Optimizing Genetic Algorithm Parameters for Atmospheric Carbon Monoxide Modeling

Meera Duggal*[1], William Daniels†[1], and Dorit Hammerling ‡[2]

[1]Department of Applied Mathematics and Statistics, Colorado School of Mines, Golden CO

January 15, 2021

Abstract

The primary source of atmospheric carbon monoxide (CO) in the Southern Hemisphere is large burn events, making CO a useful proxy for fires. Therefore, predictive CO models can help countries prepare for unusually large fire seasons. Fires are related to the climate through wind and sea surface temperatures, as they are more likely to occur when vegetation is dry. Climate indices are metrics that summarize variability in sea surface temperature and wind. A multiple linear regression model was created that uses these climate indices to predict atmospheric CO. We have created the R package *regClimateChem* to perform variable selection. This package offers three different variable selection techniques: stepwise selection, a genetic algorithm, and an exhaustive search. The exhaustive search always finds the best possible model but is computationally expensive. Stepwise selection runs quickly and is scalable but often fails to find the best model. We implemented a genetic algorithm as a compromise between computational expense and model accuracy. As a stochastic variable selection technique, the genetic algorithm has many parameters that can affect our results. Here we present a parameter optimization study for the genetic algorithm, seeking to balance computational expense and model quality. We find that a certain combination of parameters, for a four covariate case, results in 11.8% run time saving and only compromises 0.3% accuracy compared to the default settings.

Keywords: carbon monoxide, climate chemistry, multiple linear regression, variable selection,

*mduggal@mines.edu

†wdaniels@mines.edu

‡hammerling@mines.edu

genetic algorithm, optimization

Contents

1 Introduction

1.1 Motivation

In the Southern Hemisphere, the main source of CO are large burn events. As a result, CO can be used as a proxy for fires. [1]. An early warning system for large fire season would have many public safety benefits with only a few example being, it would give governments time to stock up on masks, recruit more volunteer fire fighters, and warn the public.

The CO data used in this study was collected through the Measurements Of Pollution In The Troposphere (MOPITT instrument onboard the Terra satellit). MOPITT has been gathering data for over 18 years and uses the optimal retrieval approach. The data is available at http://terra.nasa.gov/about/terrainstruments/ [1]. In order to minimize systematic and random error we select daytime, land-only retrievals from the thermal infrared product [1]. Anomalies where made by subtracting a spatial and climatological average of monthly CO from 2001 to 2016 from monthly average values. See Buchholz et al [1] for details. In Figure 1 we see the CO column and the anomaly produced for a specific region, Maritime Southeast Asia.

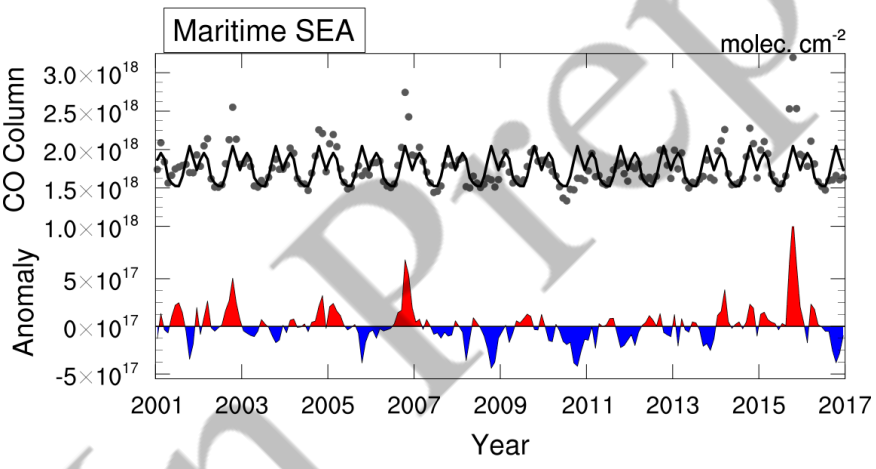


Figure 1: The plot on the top shows the monthly average of CO in the Maritime SEA region (grey dots), with the seasonal cycle (black line) [1]. The plot on the bottom shows both negative and positive anomalies.

Atmospheric CO is a useful proxy for fires in the Southern Hemisphere. Large burns events are directly related to dry vegetation and a lack of water. These conditions can be partially described with climate variability. Climate indices are used to describe variability in the climate such as the changes in air temperature, air pressure, and sea surface temperature. As a result, Buchholz et al. [1] uses four different climate indices ENSO/NINO3.4, AAO, IOD/DMI, and TSA as predictors for the multiple linear regression model. Figure 2 shows the climate indices from 2001 to 2017, the time span used in this study.

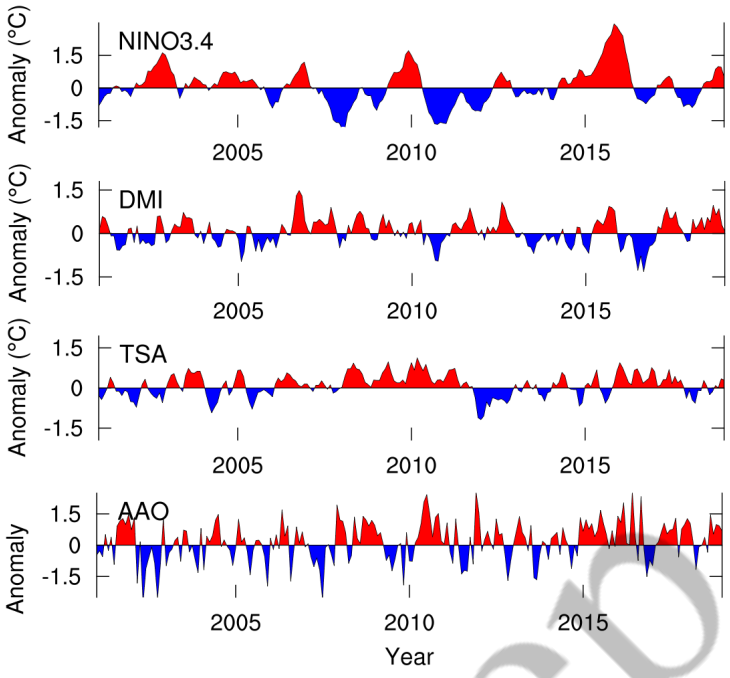


Figure 2: The four different climate indices over the time period for our study. Positive anomalies are seen in red where negative anomalies are seen in blue.

1.2 Statistical Model

A multiple linear regression model was created to model the total column CO seen by MOPITT. These models can be used for predictive purposes, such as helping countries prepare for especially extreme fire seasons. The response variable in our model is the de-seasonalized carbon monoxide anomaly and the predictor variables are the climate indices. The climate indices are lagged from 1 to 8 months because our model is intended for prediction. Each possible combination of lag values (one for each index) is called a "lagset".

For a region in the Southern Hemisphere the multiple linear regression model can be seen in equation (1).

$$CO(t) = \mu + \sum_{k} a_k \cdot \chi_k(t - \tau_k) + \sum_{i,j} b_{ij} \cdot \chi_i(t - \tau_i) \cdot \chi_j(t - \tau_j) \tag{1}$$

In equation (1), $CO(t)$ is the CO anomaly at given response time t, χ are the climate indices, τ is the lag value for each index in months, μ is the constant mean displacement, a_k and b_{ij} are coefficients, and the subscript k represents the climate indices and is a value in-between 1 and 4. The τ_k value, once chosen for an index, will remain the same for both the main and interaction terms.

1.3 regClimateChem

The R package *regClimateChem* was created to perform variable selection on equation (1) [2]. *regClimateChem* provides three variable selection techniques: exhaustive, stepwise, and the genetic algortihm. The genetic and exhaustive search were implemented using the *glmulti* package, and

stepwise selection was implemented via the *MASS* package [3] [4]. The different models were assessed using the Bayesian information criterion (BIC) because it is an appropriate choice to use when a predictive model is desired and because there is a large number of data points. The equation for the BIC is

$$\text{BIC} = \ln(n)k - 2\ln(\hat{L}),$$

where n is the number of observations, k is the number of covariates, and $\hat{L}$ is the maximized value of the likelihood function. The BIC is used to compare models. The first term in the BIC acts as a penalty for complexity and the second term in the BIC measures the fit of the model. Therefore, a lower BIC means that the model is better. The BIC is well suited for prediction because of how harsh the complexity penalty is (ln(n) vs 2 for AIC).

Stepwise selection is an iterative process that begins with either the null model or the full model. With each iteration, the algorithm considers removing or adding a predictor to minimize the BIC. The algorithm selects the model that minimizes the BIC. The process stops once adding or taking out a term no longer decreases the BIC. The exhaustive method computes the BIC value for all possible models to find the best one. The genetic algorithm uses different model modification techniques to find the best model, and it will be explained in more detail in section 2.

Daniels et al.[2] found that with only four predictor variables, the run time of the genetic algorithm is very similar to that of the exhaustive search in certain scenarios. Our goal is optimize the genetic algorithm so that the runtime is minimized, while still producing good models. The genetic algorithm that is offered in the *glmulti* package gives the user the ability to change different parameters to tailor it to their study. Here we present an optimization study to find the best *glmulti* parameters for the atmospheric CO application. By optimization we refer to minimizing runtime while maximizing model performance.

2 Genetic Algorithm

2.1 Introduction

The genetic algorithm is offered in *regClimateChem* and is implemented via the *glmulti package*. For the scenarios studied in Daniels et al [2], the genetic algorithm is less scalable but a more accurate option than stepwise selection. The Genetic Algorithm is stochastic and therefore based on probability.

We will now discuss the genetic algorithm in depth. Initially a **population** of models are randomly selected to begin the iterative process. The genetic algorithm modifies this population using three different techniques to create a new generation. With each successive generation produced the population size is maintained. The algorithm continues to produce new generations, improving the models, until a stopping criterion is satisfied. Although the algorithm is random, it is designed to improve the population of models with each successive generation. This is done by creating future generations based on the best models in the current generation. In our study models are assessed by the BIC. A model in the current generation is a **parent model** and a model in the next generation is a **child model**. In Figure 3 a general flow of the genetic algorithm is shown.

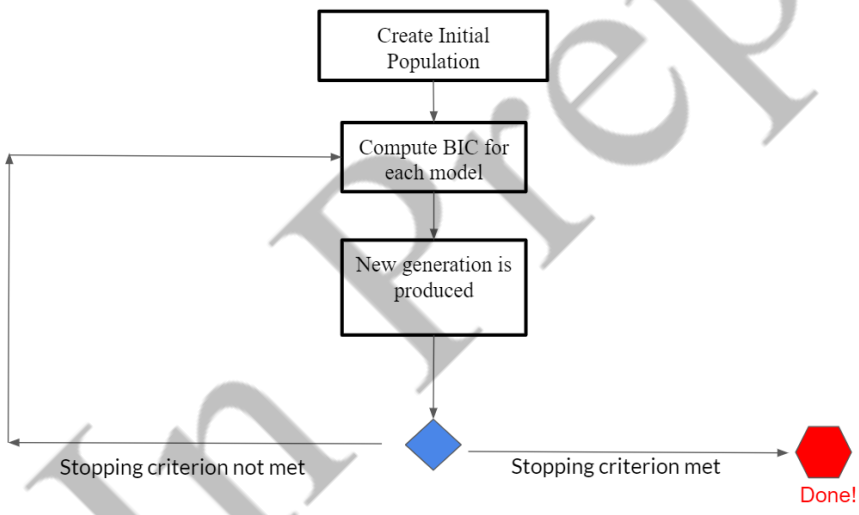


Figure 3: A general flow of the genetic algorithm.

The genetic algorithm uses a binary system to represent terms in the model. For each model, a term that is present is represented with a 1 and a term that is not present has a 0 in its place. An example of this representation can be seen below in Figure 4.

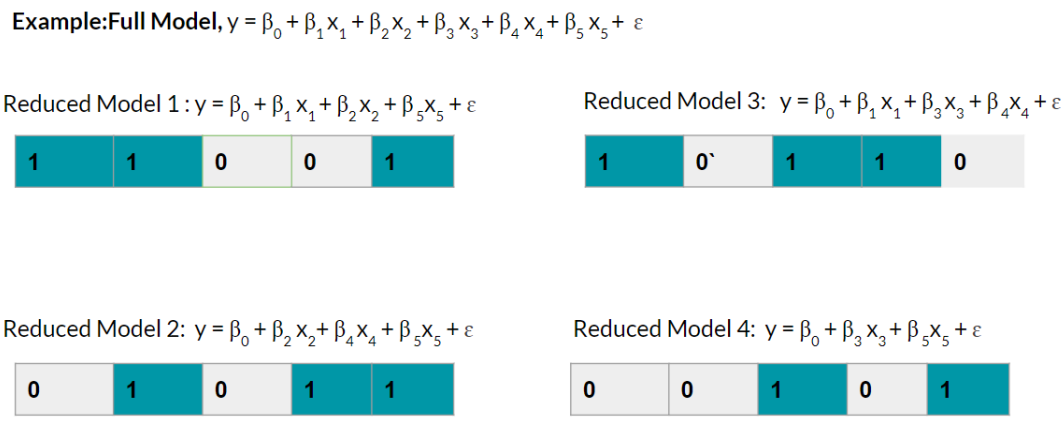


Figure 4: This is an example of the initialization process. The β_0 is always in the possible model. For sake of simplicity a simplified model is demonstrated in this example with no interaction terms present.

2.2 Model Modification Techniques

The current generation is modified using three different techniques to create the next generation of models. These techniques are:

- Asexual Reproduction
- Sexual Reproduction
- Immigration

With asexual reproduction each term in the model has the chance to mutate at a given rate called the mutation rate. This rate determines the probability that a term will mutate. When a term mutates, its binary representation switches. That is, if an included term mutates, it will no longer be included, and vice versa. A depiction of this process occurring can be seen in Figure 5

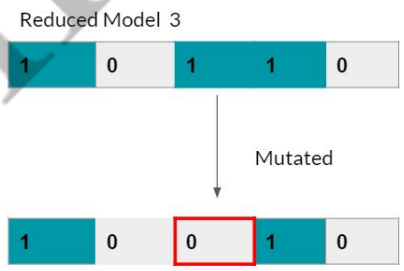


Figure 5: Example of asexual reproduction in the genetic algorithm

Another modification technique is sexual reproduction. With sexual reproduction two parents models are chosen to produce children models that will become a part of the next generation. The selection of parent models is biased according to a given information criterion. That means that "better" models are more likely to be selected as parent models. This makes future generations more likely to contain the best possible model. Each term in the child model has a 50% chance of coming from each parent model. An example of this modification method is shown in Figure 6.

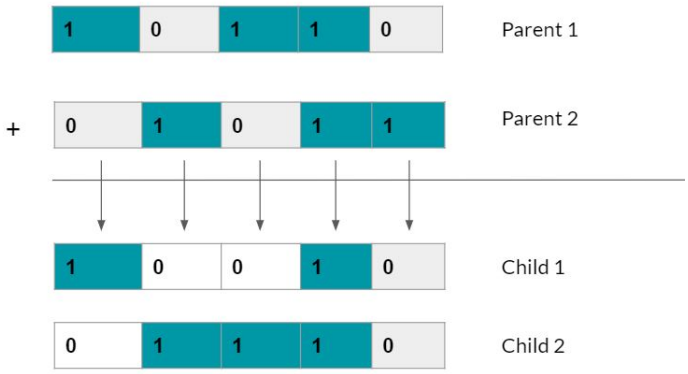


Figure 6: Example of sexual reproduction in the genetic algorithm

The last model modification technique is immigration. Immigration introduces more variability into future generations compared to the other modification techniques. Immigration does not involve a parent model. When a model is produced via immigration, each term has an equal probability of being included of excluded. This completely randomizes the new model. This process allows for a wider range of models to be introduced.

2.3 Relative Frequency of Modification Methods

The relative frequency of the three modification techniques is set by two glmulti parameters: *imm* and *sexrate*. *Imm* stands for the rate of immigration and *sexrate* is the rate of sexual reproduction. The parameter *imm* represents the proportion of models that go through immigration relative to the number of models that go through asexual reproduction. The parameter *sexrate* represents the proportion of models that go through sexual reproduction relative to the number of models that go through asexual reproduction. The default rate for both *imm* and *sexrate* are 0.3 and 0.1 respectively.

2.4 Stopping Criterion

The stopping criterion in the genetic algorithm is checked every 20 generations. The stopping criterion is made up off three different parameters, *deltaM*, *deltaB*, and *conseq*. *DeltaM* represents the target change in the mean information criterion. This means that this criterion will be satisfied once there is no longer a decrease in *deltaM* for the mean BIC which is the average of the BIC for all models produced in that generation. *DeltaB* is the target change in the best information criterion. This criterion will be met once the BIC of the best model in the current generation no longer decreases by *deltaB* when compared to the model from the previous generation. *Conseq* is the number of iterations that the genetic algorithm will run through once *deltaM* and *deltaB* are both satisfied. This is done to increase the probability that the best model has been found when the algorithm converges.

3 Optimization Study

3.1 Overview

In this study we vary glmulti parameters in order to find the parameter combination that is optimal in terms of both runtime and model accuracy. For the following study, each individual parameter value was changed and the other values where left at their default. We want to optimize over both runtime and model accuracy. Runtime refers to the total clock time it takes for the genetic algorithm to converge. Model accuracy refers to the ability of the genetic algorithm to produce the best models found by the exhaustive search. To quantify model accuracy, we look at the proportion of models that are different between the genetic algorithm and the exhaustive search.

3.2 Glmulti Parameters

The parameters that we have chosen to consider in our optimization study are in table 1:

Parameter	**Default**	**Values Studied**
Population Size	100	$5, 20, 40, 60, 80, 100$
Mutation Rate	0.001	$10^{-5}, 0.001, 0.2$
Sexual Reproduction Rate	0.1	$0.001, 0.1, 0.7$
Immigration Rate	0.3	$0.001, 0.3, 0.7$
Consecutive	5	$1, 2, 3, 4, 5$

Table 1: Each parameter we vary is shown. Along with the default and different values that were studied.

4 Four Covariate Case

We begin our optimization study with a four-covariate model, as the models in Buccholz et al [1] contain four covariates. 5 simulations where done for each that was tested. In Section 5 we test a five covariate model to see if these results scale with model size. The y-axis for the plots below are not all the same as the proportion of difference values span a large range.

4.1 Population Size

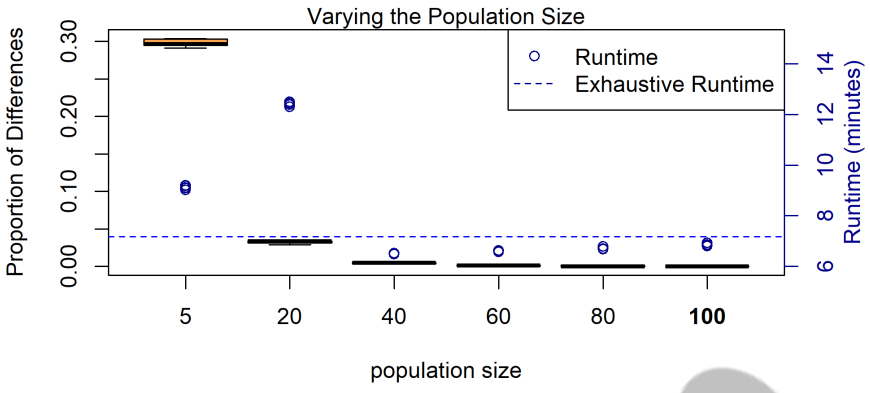


Figure 7: Results from varying population size. Default value of population size is boldfaced on the x-axis. Proportion of differences between the genetic algorithm and exhaustive search presented as box plots. Runtimes were plotted as blue circles and the exhaustive runtime is shown as a horizontal line.

A population size greater than or equal to 40 decreases run time compared to the exhaustive search, but runtime also seems to increase gradually as you go past 40. The population size of 40 minimizes the run time over the other values tested, and it also results in one of the smallest proportion of difference values. Therefore, we select 40 as the optimal population size for the four covariate case.

4.2 Immigration Rate

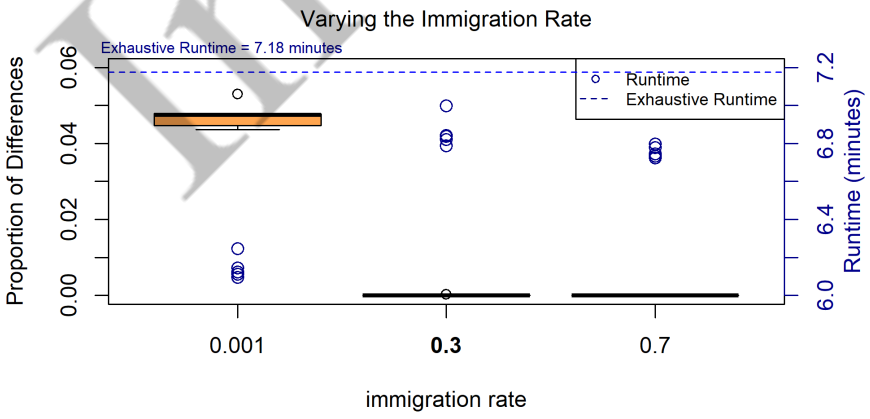


Figure 8: Results from varying the immigration rate. The default value of the immigration rate is boldfaced on the x-axis. Proportion of differences between the genetic algorithm and exhaustive search presented as box plots. Runtimes were plotted as blue circles and the exhaustive runtime is shown as a horizontal line.

Figure 8 shows that the immigration rate values greater than 0.3 results in longer run times compared to the runtimes at an immigration rate of 0.001. Although the proportion of differences at

an immigration rate of 0.001 might appear high, it is actually quite low considering the scale of the y-axis. We chose the optimal value for immigration rate to be 0.001 because it has the lowest runtime.

4.3 Sexual Reproduction Rate

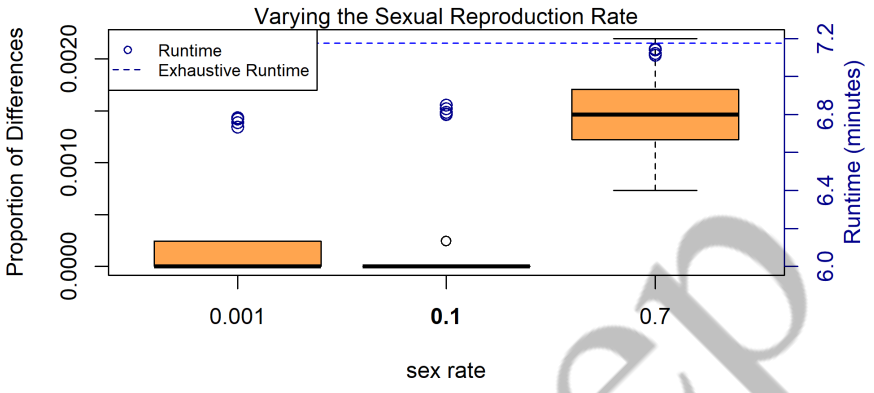


Figure 9: Results from varying the sexual reproduction rate. The default value of the sexual reproduction rate is boldfaced on the x-axis, as 0.1*. The proportion of differences between the genetic algorithm and exhaustive search are presented as box plots. Runtimes were plotted as blue circles and the exhaustive runtime is shown as a horizontal line.*

The default value of 0.1 minimizes the proportion of differences. Furthermore, there are only slight improvements in runtime when switching to a value of 0.001. Therefore, we have selected 0.1 as the optimal sexual reproduction rate parameter.

4.4 Mutation Rate

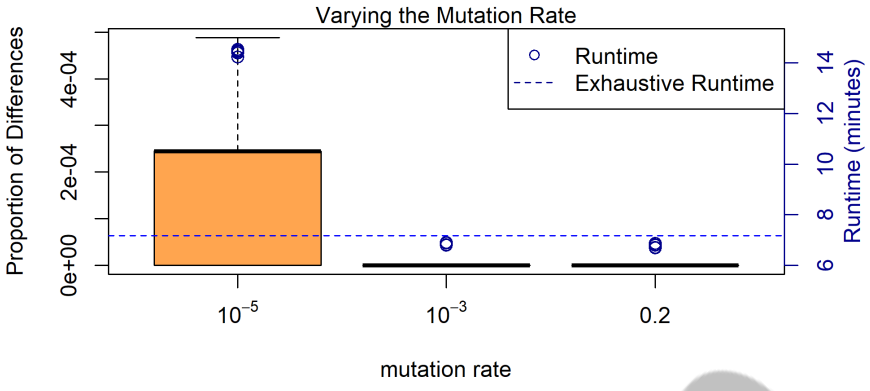


Figure 10: Results from varying the mutation rate. The default value of the mutation rate is 10^{-3}*. Proportion of differences between the genetic algorithm and exhaustive search are presented as box plots. Runtimes were plotted as blue circles and the exhaustive runtime is shown as a horizontal line.*

A mutation rate below the default drastically increases runtime while leaving the proportion of differences largely unchanged. Increasing the rate leaves the proportion of differences at essentially zero while slightly decreasing the runtime. So, we chose the optimized value for the mutation rate to be 0.2.

4.5 Consecutive Values

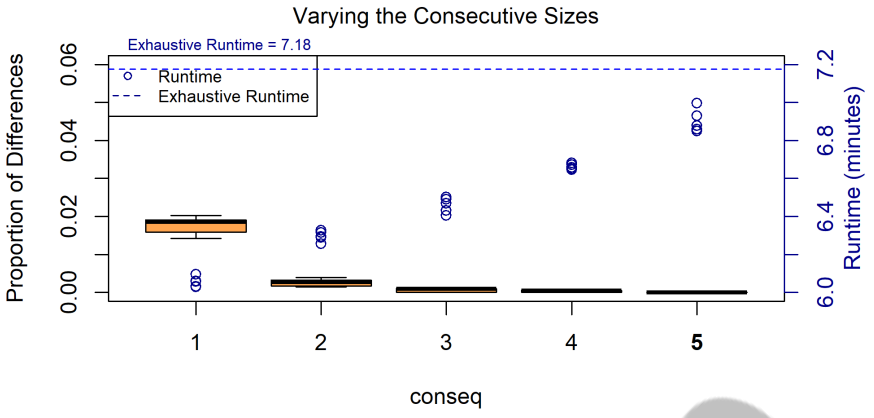


Figure 11: Results from varying different consecutive sizes. The default value of population size is boldfaced on the x-axis. The proportion of differences between the genetic algorithm and exhaustive search are presented as box plots. Runtimes were plotted as blue circles and the exhaustive runtime is shown as a horizontal line.

Increasing the consecutive value seems to linearly increase the runtime. This is expected because the consecutive value determines how many more iterations the genetic algorithm will run through *deltaB* and *deltaM* conditions are satisfied. All consecutive values that are listed have a proportion of differences that is less than 2%. We choose 2 as the optimal value because the proportion of differences is almost zero and it has a lower runtime than the default value. That being said, the run time differences are rather small.

4.6 Discussion of Results

The optimized parameters in the four covariate case are as follows.

- Population size: 40
- Mutation rate: 0.2
- Sexual reproduction rate: 0.1
- Immigration rate: 0.001
- Consecutive size: 2

In figure 12 we directly compare the default glmulti parameter values to the optimized.

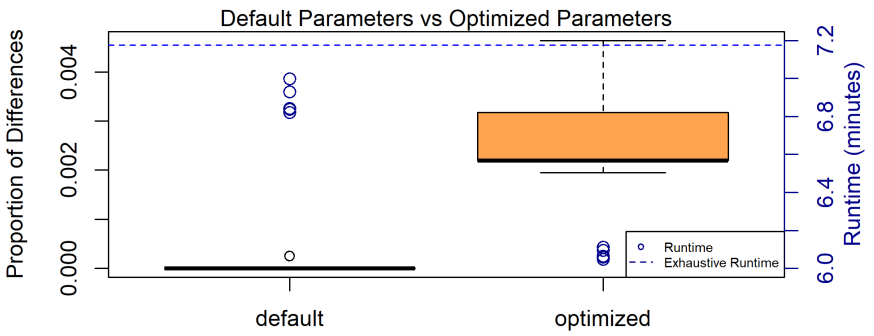


Figure 12: Comparison of default glmulti parameters to optimized parameters.

As we can see in Figure 12 using the optimized values decreases the run time an average of 11.8%. The exhasutive runtime was 7. 18 minutes. Our optimized genetic algorithm is on average 6.06 minutes so there is a decrease of over a minutes between the two model modification techniques. Also, compared to the default parameters the proportion of differences only decreases by an average of 0.28%. These results are for the four covariate case only. With only four covariates the run times are very short. Therefore, the time savings from this optimization are small. In section 5 we examine the five covariate case, which will potentially have more room for optimization, as there are many more possible models. To demonstrate the run time savings of our optimization study, we will compare the final run times of the exhaustive method, optimized genetic algorithm, and the default genetic algorithm.

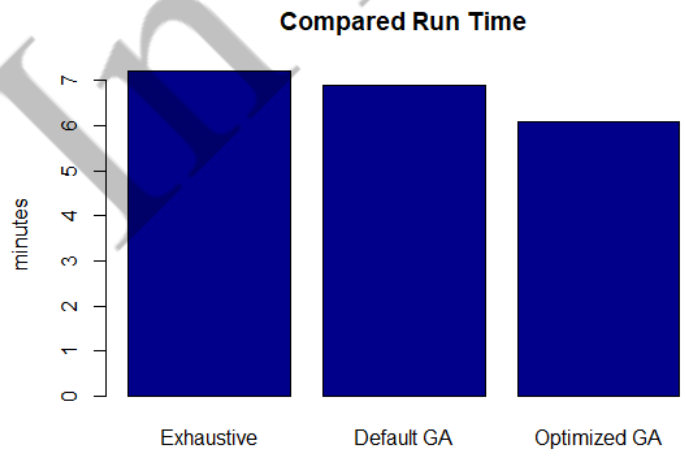


Figure 13: Runtimes for the exhaustive search, the genetic algorithm using default values, and the genetic algorithm using the optimized values.

5 Five Covariate Case

5.1 Varying a Single Parameter at a Time on a Personal Computer

To determine if our results scale to larger models, we looked at a five covariate case. First we began by varying the 5 parameters one at a time with a high, middle, and low value. This initial study were performed on a personal computer with an Intel Core i7 7th generation, 12 GB RAM, Windows 10, running 64 bit R, on R version 3.6.0. Due to longer run times, each different parameter value was only run once. Therefore we use a bar graph to represent the single value, rather than the box plots from earlier. The proportion of differences is the same as in the four covariate case: the proportion of models that differ from the exhaustive method sorted by lagset. Note that the number of possible models in the five covariate case is 1,450, hence this is why we test larger population size values.

5.1.1 Population Size

Figure 14: The default values are shown with the bolded text, as a population size of 100. The proportion of differences and runtime is seen for all the tested parameters.

A population size of 20 has a very high proportion of differences. This could because with such a small population size the algorithm will only be able to test a small number of the models at a time. The low run time at a population size of 20 could be due to the stopping criterion being hit very quickly, which is again likely related to the relatively small number of models in each generation. We can see that the proportion of differences decreases to essentially zero as you increase the population size. This could be because the genetic algorithm converges to the exhaustive search as the population size approaches the total number of possible models. We will explore a range of 100-1000 for the population size to find the optimal value.

5.1.2 Immigration Rate

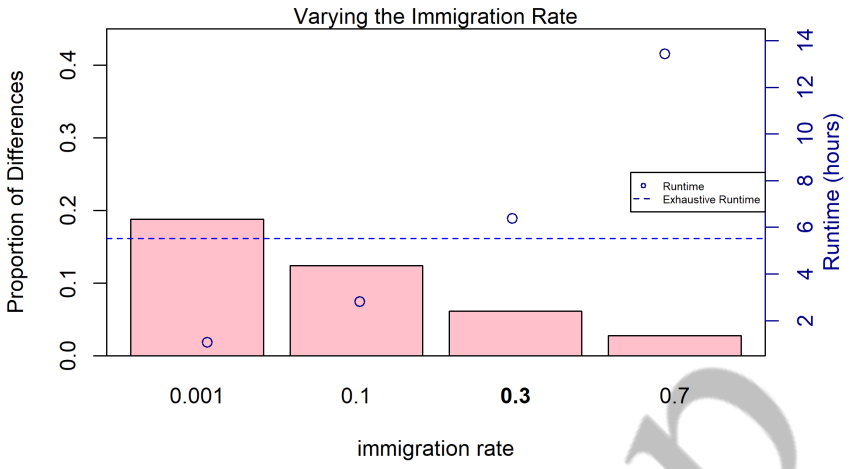


Figure 15: The default values are shown with the bolded text, as an immigration rate of 0.3*. The proportion of differences and runtime is seen for all the tested parameters.*

The immigration rate is positively correlated to runtime. Immigration introduces the highest amount of variation in the genetic algorithm. Based on the runtime results, we can therefore hypothesize that increased variability in the population of models increases runtime. This could be due to the fact that it takes longer for the stopping criterion to be met because the models are varying at higher rates. One could also conclude that high variation amongst the models leads to a low proportion of differences. This could be because the increased variability ensures that the algorithm does not get stuck in a local optimum.

5.1.3 Sexual Reproduction Rate

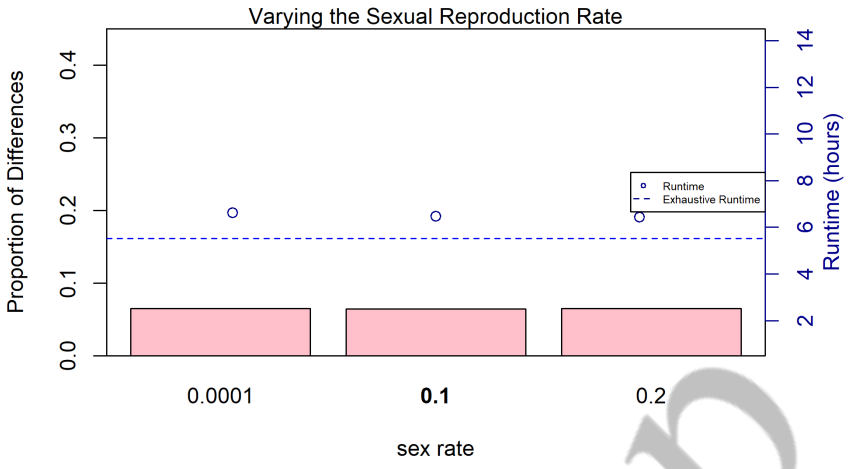


Figure 16: Two different values where tested for the parameter sexual reproduction rate, along with the default value of 0.1. *The proportion of differences is seen along with the run time in hours.*

Figure 16 shows that the proportion of differences and runtime is relatively constant across sex rate values. Therefore, we consider the sexual reproduction rate negligible as both the proportion of differences and the run time changing only slightly. This is also seen in the four covariate case.

5.1.4 Mutation Rate

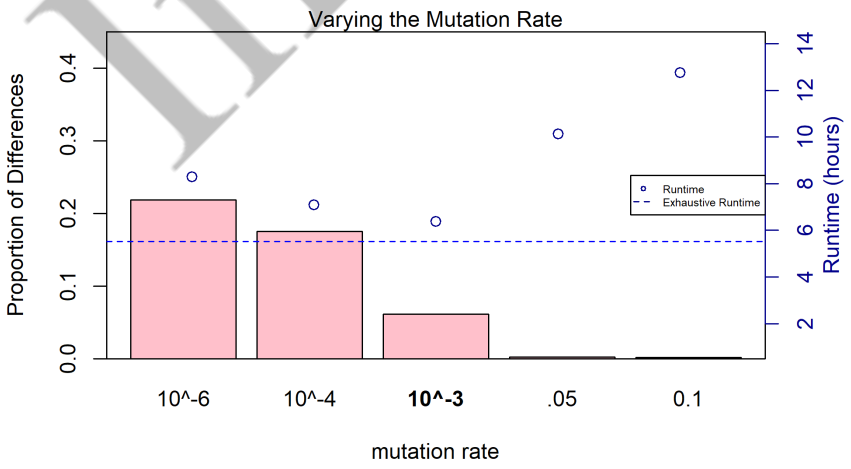


Figure 17: The default values are shown with the bolded text, as the mutation rate of 10^{-3}. *The proportion of differences and runtime is seen for all the tested parameters.*

As mutation rate increases, the proportion of differences decreases. The runtime experiences a minimum around the default vale of 10^{-3}. A good range to study further would be between 0.01 and 0.05.

5.1.5 Consecutive Values

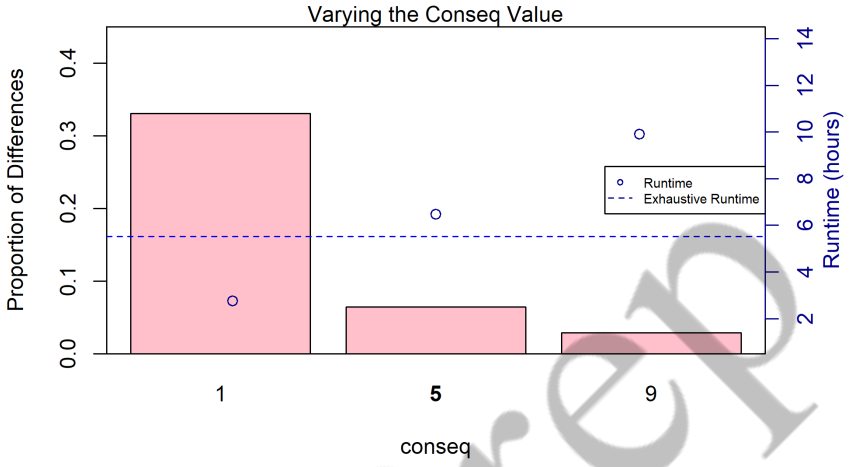


Figure 18: The default values are shown with the bolded text, as a consecutive value of 5. The proportion of differences and runtime is seen for all the tested parameters.

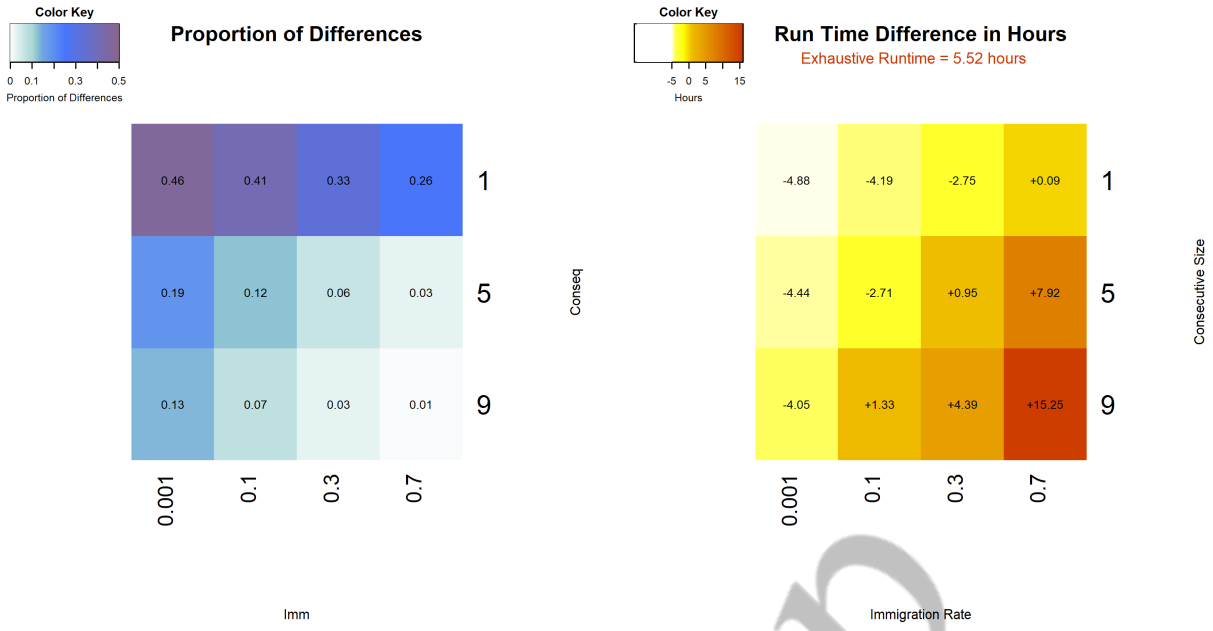
Just as in the four covariate case there is an obvious trend in both proportion of differences and the runtime. This is due to the fact that the runtime will increase with a higher consecutive rate because the genetic algorithm will have to go through more iterations before converging. The default value seems to be best and we will further investigate a tighter neighborhood surrounding the default value.

5.2 Varying Two Parameters at a Time on a Personal Computer

To determine the best possible parameters we studied the pairwise interactions between the selected glmulti parameters. This will reveal a potential interaction between the parameters. As previously seen the sexual reproduction rate was determined to be negligible so it will not be studied further.

5.2.1 Immigration Rates with Consecutive Values

The immigration rate was varied with the consecutive value to help us determine the best parameters values. The color key shows the different values for both the proportion of differences and the run time. Each cell corresponds to the same cell in the other plot, so you can easily compare the two.



(a) Proportion of Differences *(b) Runtime*

Figure 19: The default value for the genetic algorithm is an immigration value of 0.3 and a consecutive value of 5. The proportion of differences is the percentage of models that differ from the exhaustive models. The runtime difference is the runtime of the tested parameters subtracted from the runtime from the exhaustive method.

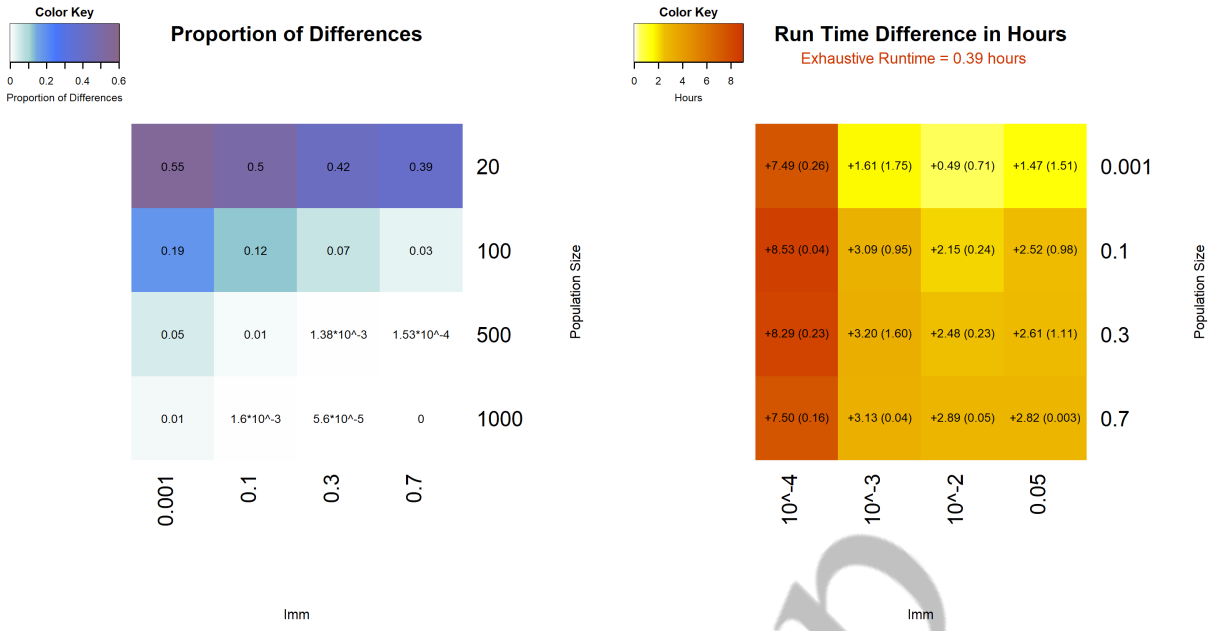
By varying the immigration rate and consecutive value we will be able to estimate how they affect one another. In this study only the immigration rate and the consecutive value where changed and every other value was left at the default. As an example of how these heatmaps are useful, we can see that consecutive value of 9 and a immigration value of 0.3 would be preferred over using a consecutive value of 5 and an immigration value of 0.7. This is because using these values results in the same proportion of differences but the runtime has a significant increase. The optimal region in this parameter space seems to be between an immigration rate of 0.001 and 0.1 and a consecutive value of 9.

5.3 Varying Two Parameters at a time on an HPC System

To further our study different parameter combinations where run on an high-performance computer (HPC system). The HPC system is called Cheyenne, which is located at a NCAR facility. Cheyenne has 145,152 processor cores, 4,032 computation nodes, and 313 TB of total system memory.

5.3.1 Population Size and Immigration Rate

First we varied population size with different immigration rate values. Below there is a heatmap seen for both proportion of differences and runtime.



(a) Proportion of Differences *(b) Runtime*

Figure 20

The first number in each tile represents the average proportion of differences or the average runtime. The runtime that is shown is the difference, an increase or decrease, in runtime based off of the exhaustive runtime on Cheyenne. This is done because as mentioned before the exhaustive method always produces the best models. The second number in the runtime heatmap represents the standard deviation amongst the 5 trials. Due to the low number of trials and high variability a table is shown below showing each individual trial's runtime.

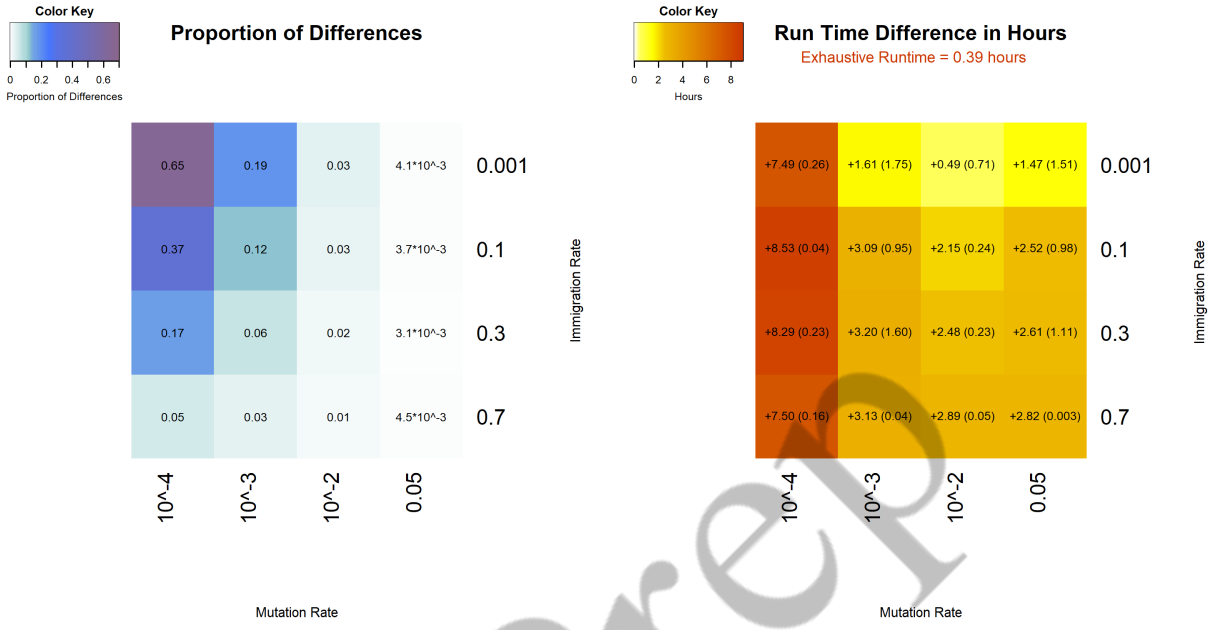
Cheyenne	0.001	0.1	0.3	0.7
20	3.81, 8.05 8.13, 8.26 8.09	3.73, 3.58 4.38, 4.47 4.33	3.92, 2.65 4.12, 4.10 4.01	2.72, 2.65 3.40, 3.33 3.32
100	2.95, 6.11 6.25, 6.04 6.04	2.72, 5.15 5.64, 5.16 5.11	3.96, 3.02 3.40, 3.65 3.60	3.34, 2.54 3.83, 3.76 3.90
500	1.02, 2.90 2.95, 2.99 2.98	1.33, 3.00 3.08, 3.02 3.04	1.37, 2.85 2.90, 2.90 2.89	1.36, 3.76 2.84, 2.85 2.83
1000	0.94, 2.60 2.46, 2.63 2.56	1.40, 2.98 2.92, 1.02 2.97	1.39, 3.31 2.84, 2.92 2.90	1.35, 2.93 2.99, 2.92 2.93

Figure 21: Different runtimes running 5 different trials on Cheyenne

For the population size and immigration rate we can see that a higher population size and higher immigration rate reduces the proportion of differences. With the goal of minimizing the runtime and keeping a small proportion of differences an optimal combination to choose would be a population size of 1000 and an immigration rate of 0.001. There seems to be low variation in the runtime as well for this parameter combination.

5.3.2 Immigration Rate and Mutation Rate

We then varied the two parameters, immigration rate and mutation rate together. As before we can see a heatmap for this parameter combination and a table containing individual runtimes.



(a) Proportion of Differences *(b) Runtime*

Figure 22: The default value for the genetic algorithm is an immigration value of 0.3 *and a mutation rate of* 10^{-3}*. Both standard deviation and difference for runtime is shown in the runtime heatmap.*

Imm and Mut	10^-4	10^-3	10^-2	0.05
0.001	+7.13, +7.76 +7.38, +7.39 +7.82	+4.60, -0.32 +0.02, +1.87 +1.88	+1.83, +0.06 -0.24, +0.40 +0.38	+4.27, +1.81 +0.08, +0.60 +0.62
0.1	+8.53, +8.50 +8.60, +8.49 +8.52	+4.98, +2.58 +2.60, +2.64 +2.63	+2.40, +2.47 +1.94, +1.96 +1.97	+4.49, +2.01 +2.00, +2.05 +2.05
0.3	+8.20, +8.53 +8.60, +8.01 +8.07	+6.37, +2.33 +2.24, +2.27 +2.79	+2.59, +2.59 +2.60, +2.59 +2.01	+4.83, +2.04 +2.06, +2.05 +2.07
0.7	+7.23, +7.64 +7.41, +7.66 +7.57	+3.10, +3.10 +3.12, +3.11 +3.21	+2.92, +2.90 +2.84, +2.83 +2.96	+2.82, +2.83 +2.80, +2.90 +2.73

Figure 23: Different runtimes running 5 different trials with immigration and mutation rate values

With a mutation rate of 0.05 it can be seen that with any combination of immigration rate, there is very low proportion of differences with any immigration rate value. The lowest runtime parameter combination is an immigration rate of 0.001 and a mutation rate of 10^{-2}.

5.3.3 Population Size and Mutation Rate

Then population size and mutation rate where varied simultaneously. The same heatmaps and tables where produced, as was one for the other parameter combinations.

(a) Proportion of Differences *(b) Runtime*

Figure 24: The default value for the genetic algorithm is a mutation rate of 10^{-3} and a population size of 100.

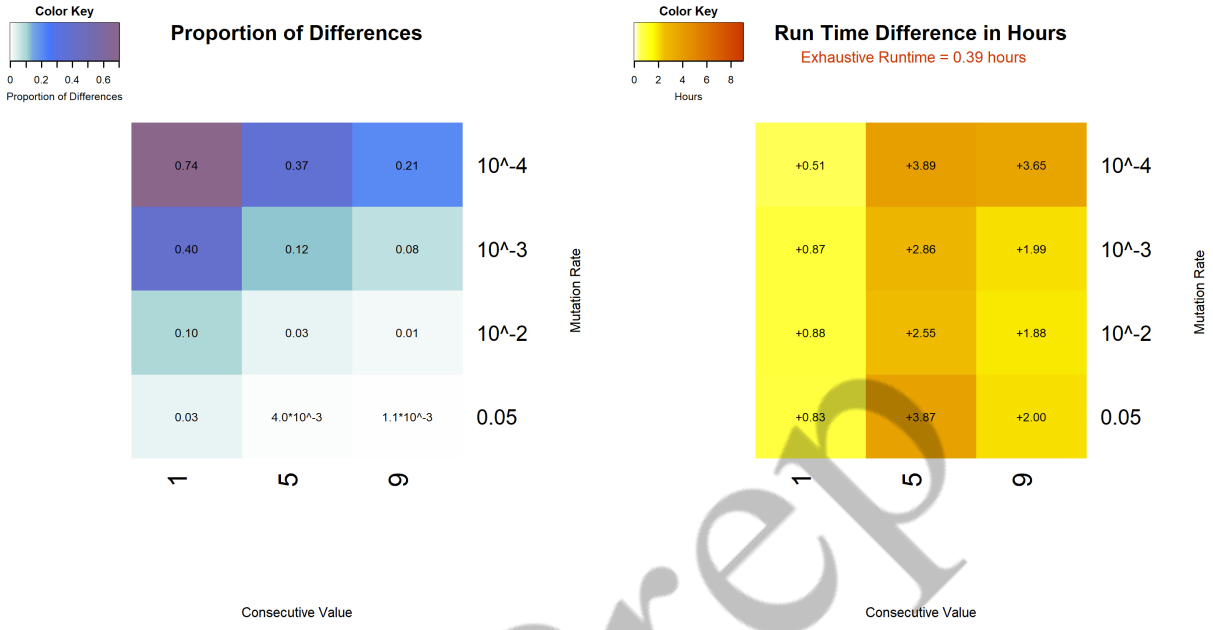
Mut / Pop	20	100	500	1000
10^-4	+3.29, +3.16 +3.19, +3.20 +3.20	+8.59, +8.65 +8.46, +8.78 +8.65	+5.19, +5.10 +4.89, +5.29 +5.42	+1.04, +1.07 +1.09, +1.08 +1.09
10^-3	+8.76, +9.48 +8.78, +8.97 +8.71	+2.06, +2.08 +2.08, +2.08 +2.08	+5.59, +5.22 +5.50, +5.67 +5.55	+0.99, +1.01 +0.99, +0.99 +0.98
10^-2	+2.82, +2.91 +2.87, +2.87 +2.86	+1.77, +1.76 +1.00, +1.74 +1.81	+1.39, +0.79 +1.44, +1.48 +1.25	+0.78, +0.80 +0.78, +0.77 +0.80
0.05	+2.67, +2.62 +2.65, +2.65 +2.64	+1.88, +1.87 +0.29, +1.85 +1.83	+1.30, +1.14 +1.31, +1.31 +1.24	+0.84, +0.82 +0.82, +0.79 +0.81

Figure 25: The runtimes from varying mutation rate and population size, in order of the trials.

With a population size of 1000 there is both low runtime and proportion of differences, regardless of the mutation rate value. It appears as if a population size of 1000 and a mutation rate of 10^{-2} gives us both a low runtime and small proportion of differences.

5.3.4 Consecutive values and Mutation Rate

We also varied mutation rate with different consecutive values. Both heatmap and a table of the runtimes can be seen below.



(a) Proportion of Differences *(b) Runtime*

Figure 26: The default value for the genetic algorithm is a consecutive value of 5 and a mutation rate of 10^{-3}.

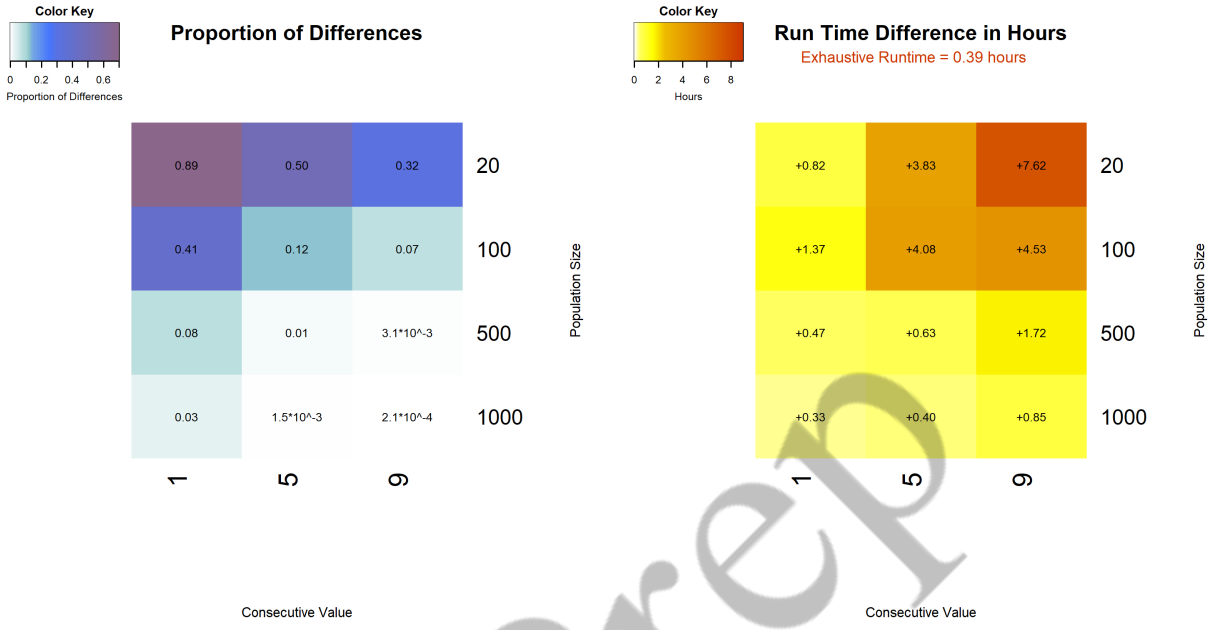
Mut / Conseq	1	5	9
10^-4	+0.72, +0.71 +0.70, +0.71 -0.29	+3.96, +3.85, +3.85, +3.95 +3.85	+4.70, +4.65 +2.96, +2.96 +2.98
10^-3	+0.83, +0.85 +0.81, +0.82 +1.07	+2.86, +2.84 +2.81, +2.96 +2.82	+1.23, +1.22 +2.51, +2.46 +2.53
10^-2	+0.81, +0.85 +0.82, +0.80 +1.11	+2.53, +2.54 +2.49, +2.62 +2.55	+1.18, +1.18 +2.39, +2.30 +2.37
0.05	+0.75, +0.74 +0.77, +0.79 +1.11	+3.90, +3.85 +3.86, +3.89 +3.87	+1.32, +1.32 +2.45, +2.47 +2.44

Figure 27: The runtimes from varying mutation rate and conseuctive values, in order of the trials.

With a consecutive value of 1 the runtime decreases significantly compared to using a value of 5 and 9. As before we can see a higher mutation rate is associated with better proportion of differences. Using a consecutive value of 1 and a mutation rate of 10^{-2} seems to have a low runtime and a low proportion of differences.

5.3.5 Population Size and Consecutive Size

The sixth and final parameter combination that we tested was population size and consecutive size. Both heatmaps and a table of the different run times was produced for this parameter combination.



(a) Proportion of Differences *(b) Runtime*

Figure 28: The default value for the genetic algorithm is a consecutive value of 5 and a population size of 100.

Pop / Conseq	1	5	9
20	-0.37, +0.96 +1.16, +1.18 +1.16	-0.33, +4.79 +4.96, +4.86 +4.86	+4.19, +8.37 +8.40, +8.52 +8.63
100	+1.32, +1.31 +1.42, +1.41 +1.38	+3.89, +3.81 +4.31, +4.20 +4.18	+5.49, +5.40 +3.83, +3.94 +3.96
500	+0.47, +0.47 +0.46, +0.47 +0.50	+0.64, +0.66 +0.61, +0.61 +0.65	+1.71, +1.72 +1.73, +1.73 +1.69
1000	+0.37, +0.45 -0.07, +0.47 +0.41	+0.52, +0.52 +0.0006, +0.51 +0.45	+1.09, +1.02 +0.05, +1.03 +1.05

Figure 29: The runtimes from varying population size and consecutive values, in order of the trials.

There is low runtimes and proportion of differences for any consecutive value when using a population size of 1000. When there is a population size of 1000 the proportion of differences does not increase much when using a consecutive value of 1 compared to using one of 5. Thus, a good parameter combination would be a population size of 1000 and a consecutive value of 1.

5.3.6 Discussion of Results

Based on the pairwise studies above, we have selected the following parameter values for use in the five covariate case.

- Population size: 1000
- Mutation rate: 10^{-2}
- Immigration rate: 0.001
- Consecutive value: 5

These are the optimized values that resulted when doing the study on Cheyenne (HPC system at NCAR). It appears that on Cheyenne the exhaustive method runs a lot quicker than on my personal computer. Where these values appear to be the best for Cheyenne, results for a personal computer may differ.

We are going to directly compare the optimized glmulti parameters with the default ones.

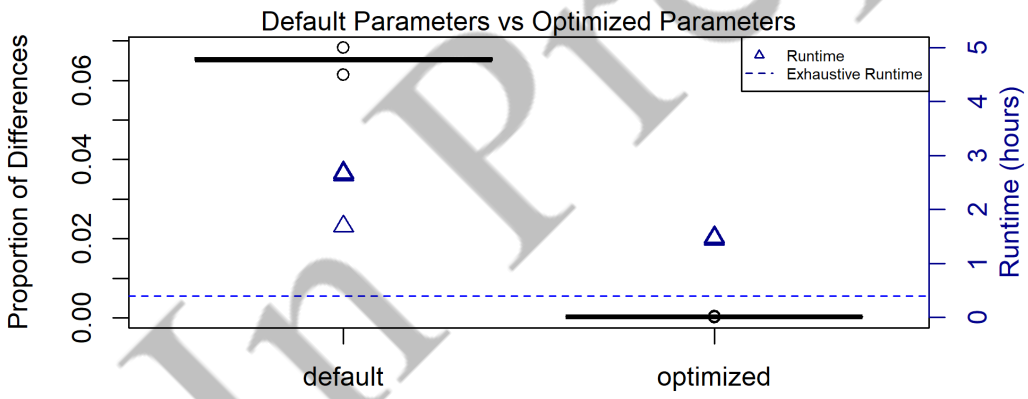


Figure 30: Comparison of default glmulti parameters to optimized parameters

From Figure 30 we can see that the optimized parameters chosen, compared to the default, not only decreased the proportion of differences but also the runtime. The runtime decreased by an average of 1 hour and 3.6 seconds and the proportion of differences decreased by an average of .4%. As we can see from the dashed line the optimized parameters failed to decrease the runtime to be lower than the exhaustive method on Cheyenne. Again, this results might differ on a personal computer.

To compare the different runtimes we show a bar graph of the exhaustive method, the default genetic algorithm, and the optimized genetic algorithm.

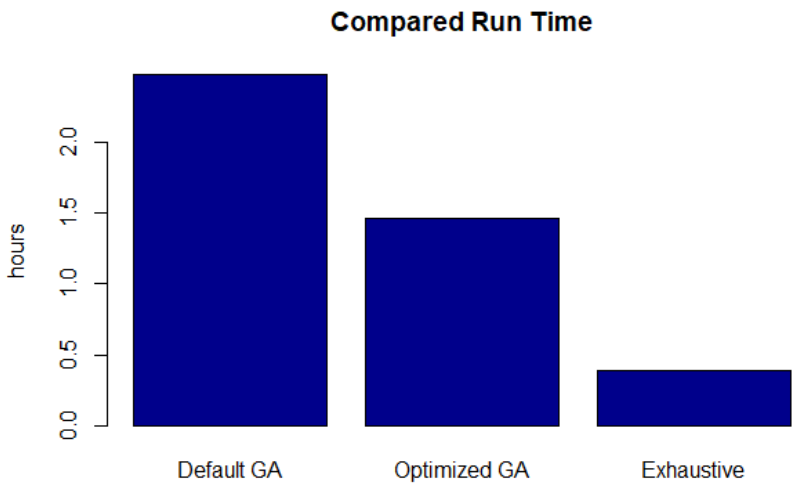


Figure 31: Runtimes for the default genetic algorithm, optimized genetic algorithm, and the exhaustive method.

6 Conclusion

In the four covariate case we where able to successfully optimize the genetic algorithm so its runtime was lower than the exhaustive method. This was done with the following parameters: a population size of 40, mutation rate of 0.2, sexual reproduction rate of 0.1, immigration rate of 0.001, and a consecutive size of 2. This study was ran on my personal computer.

For the five covariate case our study was moved to the HPC system Cheyenne. We optimized the genetic algorithm with the following parameters: population size of 1000, mutation rate of 10^{-2}, immigration rate of 0.001, and a consecutive value of 5. In the five-covariate case we can see that the optimized genetic algorithm's runtime is less than the default. That being said, the exhaustive method ended up being faster than the optimized genetic algorithm. This was not our originally intended result, but these findings are interesting and useful. This tells us that when using an HPC system, exhaustive searches outperform even an optimized genetic algorithm.

References

[1] R. R. Buchholz, D. Hammerling, H. M. Worden, M. N. Deeter, L. K. Emmons, D. P. Edwards, and S. A. Monks. Links between carbon monoxide and climate indices for the southern hemisphere and tropical fire regions. *Journal of Geophysical Research: Atmospheres*, 123, 2018.

[2] William Daniels, Dorit Hammerling, and Rebecca Buchholz. regclimatechem: An r package for data driven variable selection applied to atmospheric carbon monoxide. Technical report, Colorado School of Mines, 2000.

[3] V. Calcagno and C. de Mazancourt. glmulti: An R package for easy automated model selection with (generalized) linear models. *Journal of Statistical Software*, 34(12):29, 2010.

[4] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.