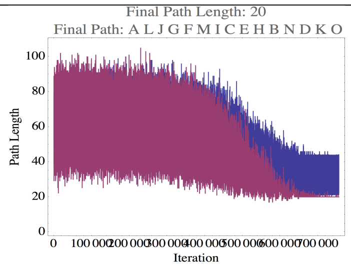
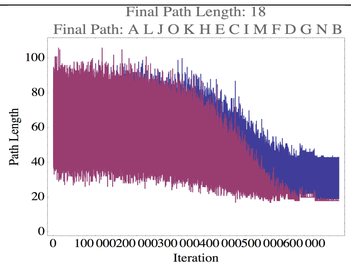
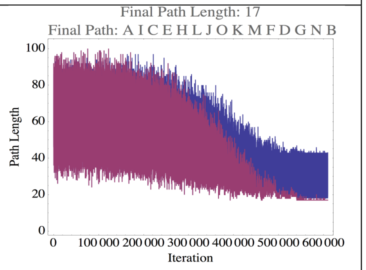
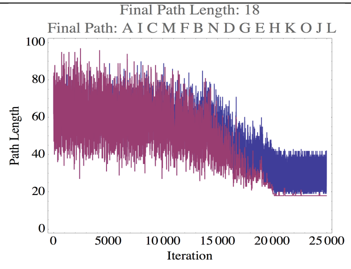
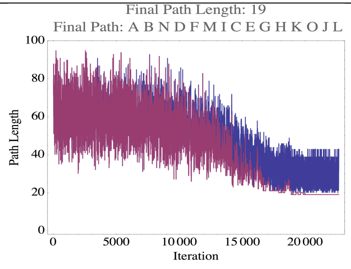
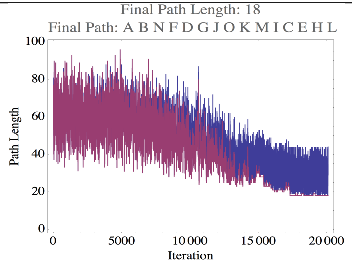
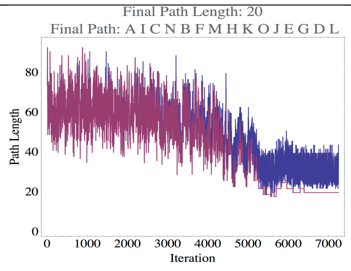
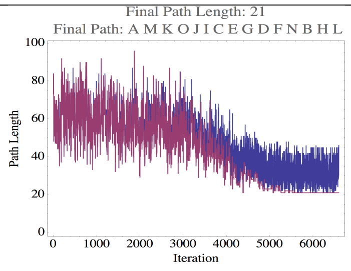
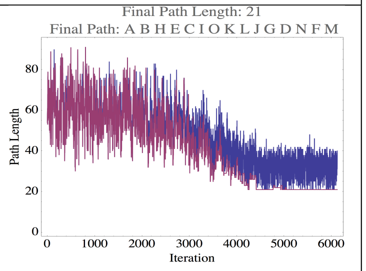
Mikhail Gaerlan
21 April 2017
Assignment 2 STA 243 Lee

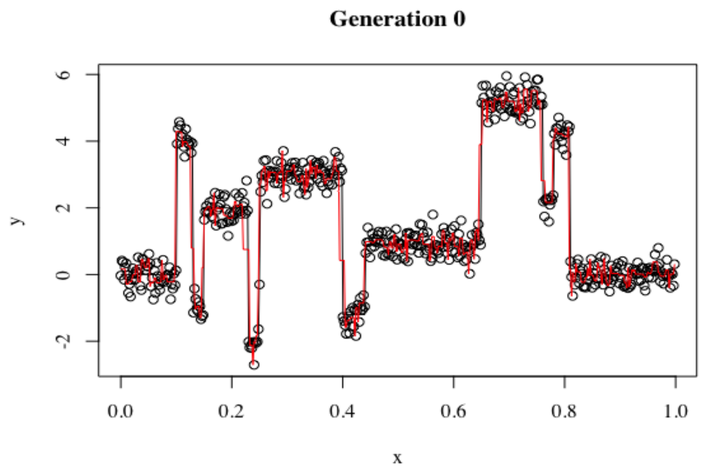
1 Simulated Annealing

The following table shows the results of the simulated annealing algorithm with various parameters. Each stage contains 100 iterations. The algorithm would stop when either the maximum number of 10,000 stages was reached or the probability of replacing a worse solution was less than 10^{-50}. The blue data represents the path length of the randomly picked $\theta^* \in N(\theta_k)$ where $N(\theta)$ was the neighborhood of θ where two elements of θ were swtiched. The pink data represents the path length of θ^* such that $\theta_{k+1} = \theta^*$, e.g. when $f(\theta^*) - f(\theta_k) < 0$. The figures show clearly that as the temperature decreases, the entropy or randomness of the solutions decreases and cools down towards a minimum. The smaller p values and τ_1 values cause the system to cool down faster. The best solutions found are the sequences (A, I, C, E, H, L, J, O, K, M, F, D, G, N, B) and (A, J, O, K, M, F, N, G, D, L, H, B) both with a path length of 17.

	$\tau_1 = 400$	$\tau_1 = 200$	$\tau_1 = 100$
$p = 0.999$			
$p = 0.970$			
$p = 0.900$			

2 Genetic Algorithms

The initial result for the genetic algorithm is shown in Figure 2. While the model appears to fit the data, the data has been over-fit and very noisy as well.



The figures in the table on the next page, show the results of the genetic algorithm with different criteria and different algorithm parameters. The algorithm was able to minimize the MDL and AIC. The model achieved getting fewer pieces in the function, but the models found by the algorithm do not seem to fit the data very well.

```
par(family = 'serif')
setwd("/Users/mikhailgaerlan/Box Sync/Education/UC Davis/2016-2017
Spring/STA 243 Computational Statistics/Assignments/Assignment 2")
rm(list=ls())

#=========
#    1
#=========
#Distance Between Cities
distances = matrix(
   #1,2,3,4,5,6,7,8,9,0,1,2,3,4,5
   #A,B,C,D,E,F,G,H,I,J,K,L,M,N,O
  c(0,1,2,4,9,8,3,2,1,5,7,1,2,9,3,#A,1
    1,0,5,3,7,2,5,1,3,4,6,6,6,1,9,#B,2
    2,5,0,6,1,4,7,7,1,6,5,9,1,3,4,#C,3
    4,3,6,0,5,2,1,6,5,4,2,1,2,1,3,#D,4
    9,7,1,5,0,9,1,1,2,1,3,6,8,2,5,#E,5
    8,2,4,2,9,0,3,5,4,7,8,3,1,2,5,#F,6
    3,5,7,1,1,3,0,2,6,1,7,9,5,1,4,#G,7
    2,1,7,6,1,5,2,0,9,4,2,1,1,7,8,#H,8
    1,3,1,5,2,4,6,9,0,3,3,5,1,6,4,#I,9
    5,4,6,4,1,7,1,4,3,0,9,1,8,5,2,#J,10
    7,6,5,2,3,8,7,2,3,9,0,2,1,8,1,#K,11
    1,6,9,1,6,3,9,1,5,1,2,0,5,4,3,#L,12
    2,6,1,2,8,1,5,1,1,8,1,5,0,9,6,#M,13
    9,1,3,1,2,2,1,7,6,5,8,4,9,0,7,#N,14
    3,9,4,3,5,5,4,8,4,2,1,3,6,7,0 #O,15
  ),
  nrow=15,
  ncol=15,
  byrow=TRUE
)

#Objective Function
obj = function(theta){
  result = distances[1,theta[1]]
  for (i in 1:(length(theta)-1)){
    result = result + distances[theta[i],theta[i+1]]
  }
  result = result + distances[1,theta[length(theta)]]
  return(result)
}
alpha = function(tau,p){
  result = p*tau
  return(result)
}
beta = function(m){
  result = 100
  return(result)
}
```

```
neighborhoodsample = function(theta){
  switch = sample(1:length(theta),2)
  newtheta = theta
  newtheta[switch[1]] = theta[switch[2]]
  newtheta[switch[2]] = theta[switch[1]]
  return(newtheta)
}

#Simulated Annealing Parameters
for (p in c(0.999,0.97,0.9)){
  for (tau in c(400,200,100)){
    printl = TRUE
    writel = TRUE
    maxiter = 10000
    prob = 10^(-50)
    resultsname =
paste("resultsp",toString(p*1000),"t",toString(tau),".csv",sep = "")
    allname = paste("allp",toString(p*1000),"t",toString(tau),".csv",sep
= "")

    theta = sample(2:15,14)
    #theta = c(9,3,5,10,15,11,13,6,14,7,4,12,2,8)
    #theta = 2:15
    if (writel){
      write(sprintf("%d %d %d %s",obj(theta),0,0,paste(theta,collapse =
" ")),file = resultsname,append = FALSE)
      write(sprintf("%d %d %d %s",obj(theta),0,0,paste(theta,collapse =
" ")),file = allname,append = FALSE)
    }
    #Begin Simulated Annealing
    break_outer = FALSE
    for (j in 1:maxiter){
      for (m in 1:beta(j)){
        newtheta = neighborhoodsample(theta)
        if (printl) print(sprintf("%2s %3d",paste(newtheta,collapse="
"),obj(newtheta)))
        if (writel) write(sprintf("%d %d %d %s",obj(newtheta),j-1,m-
1,paste(newtheta,collapse=" ")),file = allname,append = TRUE)
        delta = obj(newtheta) - obj(theta)
        if (exp(-delta/tau) < prob){
          break_outer = TRUE
          break
        }
        if (delta <= 0){
          theta = newtheta
          if (writel) write(sprintf("%d %d %d %s",obj(theta),j-1,m-
1,paste(theta,collapse=" ")),file = resultsname,append = TRUE)
        } else if (runif(1) < (exp(-delta/tau))){
          theta = newtheta
          if (writel) write(sprintf("%d %d %d %s",obj(theta),j-1,m-
1,paste(theta,collapse=" ")),file = resultsname,append = TRUE)
```

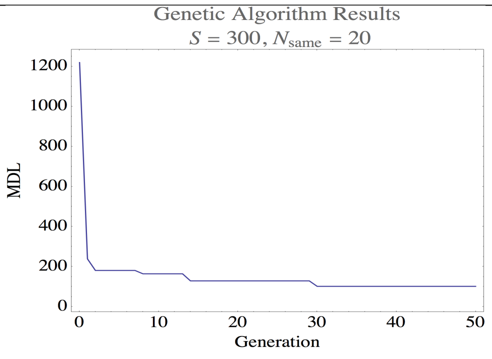
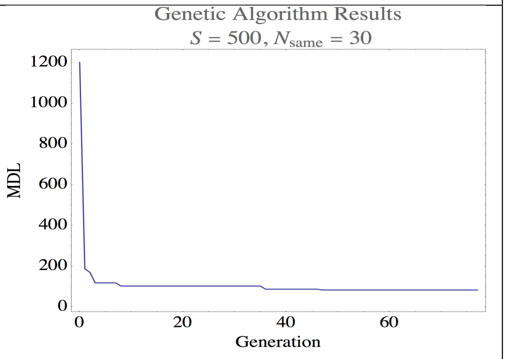
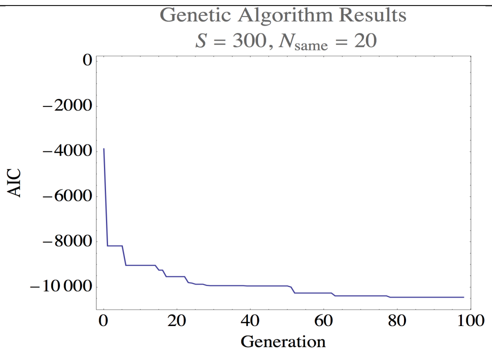
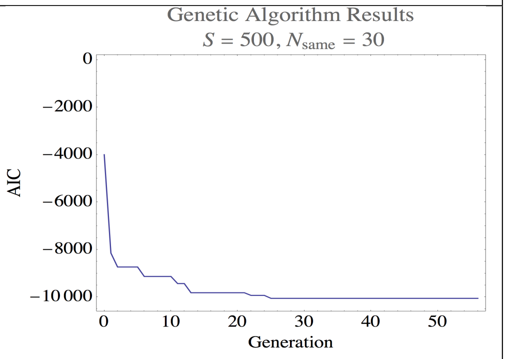
```
        } else {
          if (writel) write(sprintf("%d %d %d %s",obj(theta),j-1,m-
1,paste(theta,collapse=" ")),file = resultsname,append = TRUE)
        }
      }
      if (break_outer) break
      tau = alpha(tau,p)
    }
    print(sprintf("%2s %3d",paste(theta,collapse=" "),obj(theta)))
  }
}
#print(obj(c(9,3,5,10,15,11,13,6,14,7,4,12,8,2)))
```

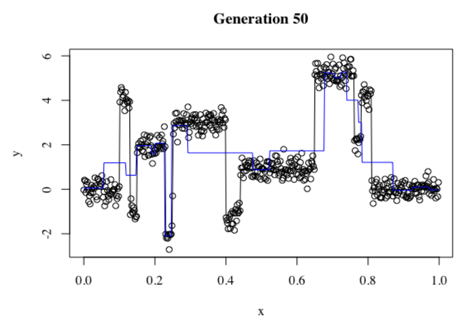
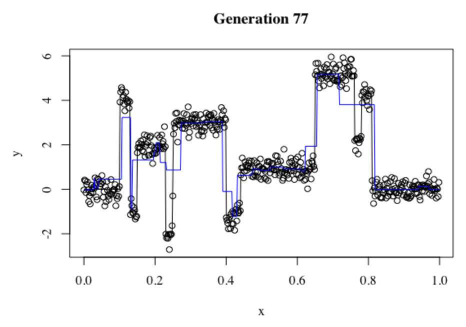
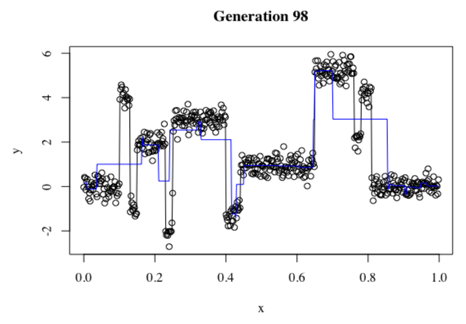
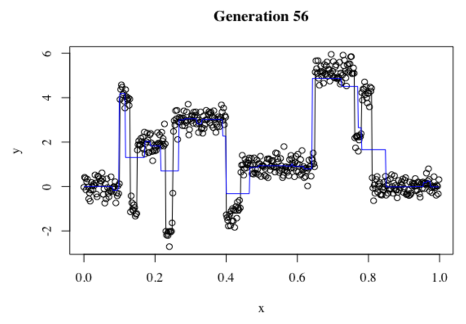
```
par(family = 'serif')
setwd("/Users/mikhailgaerlan/Box Sync/Education/UC Davis/2016-2017
Spring/STA 243 Computational Statistics/Assignments/Assignment 2")
rm(list=ls())

#=========
#    2
#=========
#======================
# Test Data and Function
#======================
truefunction = function(x){
  t = c(0.1,0.13,0.15,0.23,0.25,0.4,0.44,0.65,0.76,0.78,0.81)
  h = c(4,-5,3,-4,5,-4.2,2.1,4.3,-3.1,2.1,-4.2)
  temp = 0
  for (i in 1:11){
    temp = temp+h[i]/2*(1+sign(x-t[i]))
  }
  return(temp)
}
n = 512
x = (0:(n-1))/n
f = truefunction(x)
set.seed(0401)
y = f+rnorm(f)/3
plot(x,y)
lines(x,f)

#======================
# Program functions
#======================
getparams = function(chromo){
  pieces = sum(chromo)+1
  breaks = 0*1:(pieces-1)
  breakindex = 0*1:(pieces-1)
  heights = 0*1:pieces
  j = 1
  for (n in 1:length(chromo)){
    if (chromo[n] == 1){
      breaks[j] = x[n+1]
      breakindex[j] = n+1
      if (j == 1){
        heights[j] = mean(y[1:n])
      } else{
        heights[j] = mean(y[(breakindex[j-1]+1):breakindex[j]])
      }
      sum = y[n]
      j = j + 1
    }
  }
  heights[pieces] = mean(y[(breakindex[pieces-1]):length(x)])
  return(c(pieces,breaks,heights,breakindex))
}

modelfunction = function(x,t,h){
  temp = 0
  for (j in 1:length(h)){
    if (j == 1){
      if ((x >= 0)&&(x < t[j])){
        temp = h[j]
        break
      }
    } else if (j == length(h)){
      if ((x >= t[j-1])&&(x <= 1)){
        temp = h[j]
        break
      }
    } else {
      if ((x >= t[j-1])&&(x < t[j])){
        temp = h[j]
        break
      }
    }
  }
  return(temp)
}

mdl = function(chromo){
  params = getparams(chromo)
  limits = params[2:(params[1])]
  heights = params[(params[1]+1):(2*params[1])]
  indices = params[(2*params[1]+1):length(params)]
  nj = 1:params[1]
  nj[1] = indices[1]-1
  for (i in 2:(params[1]-1)){
    nj[i] = indices[i]-indices[i-1]
  }
  nj[params[1]] = n+1-indices[params[1]-1]
  ymodel = getmodel(chromo,limits,heights)
  return(params[1]*log(n)+(1/2)*sum(log(nj))+(n/2)*log((1/n)*sum((y-
ymodel)^2)))
}

aic = function(chromo){
  params = getparams(chromo)
  limits = params[2:(params[1])]
  heights = params[(params[1]+1):(2*params[1])]
  ymodel = getmodel(chromo,limits,heights)
  return(n*log((1/n)*sum(y-ymodel)^2)+2*params[1]*log(n))
}
```

```
getmodel = function(chromo,limits,heights){
  ymodel = 0*x
  for (i in 1:length(x)){
    ymodel[i] = modelfunction(x[i],limits,heights)
  }
  return(ymodel)
}

rank = function(population,fitness){
  rankings = 0*fitness
  sortedfitness = sort(fitness,decreasing=TRUE)
  for (i in 1:dim(population)[1]){
    for (j in 1:dim(population)[1]){
      if (sortedfitness[i]==fitness[j]){
        rankings[j] = i
        break
      }
    }
  }
  return(rankings)
}

#=======================
# Main Program
#=======================
pop_size = 300
stop_gen = 20
cross_prob = 0.9
mut_prob = 0.05
printl = TRUE
criteria = FALSE
#TRUE for MDL
#FALSE for AIC
if (criteria){
  file_name = sprintf("mdl_gen_data_%d_%d.csv",pop_size,stop_gen)
} else{
  file_name = sprintf("aic_gen_data_%d_%d.csv",pop_size,stop_gen)
}

#----------------------------
# Population Initialization
#----------------------------
if (printl) print("Generation 0")
population = array(0,dim=c(pop_size,n-1))
fitness = array(0,dim=c(pop_size))
for (i in 1:pop_size){
  population[i,1:(n-1)] = sample(c(0,1),n-1,replace=TRUE)
  if (criteria){
    fitness[i] = mdl(population[i,1:(n-1)])
  } else{
    fitness[i] = aic(population[i,1:(n-1)])
  }
}
rankings = rank(population,fitness)

#Best Model
for (i in 1:pop_size){
  if (rankings[i] == pop_size){
    print(sprintf("Best Fitness: %f",fitness[i]))
    write(sprintf("%d %f",0,fitness[i]),file=file_name,append = FALSE)
    chromo = population[i,1:(n-1)]
    params = getparams(chromo)
    limits = params[2:(params[1])]
    heights = params[(params[1]+1):(2*params[1])]
    ymodel = getmodel(chromo,limits,heights)
    plot(x,y)
    title(main = sprintf("Generation 0"))
    lines(x,f)
    lines(x,ymodel,col='red')
    break
  }
}


#----------------------------
# Begin Genetic Algorithm
#----------------------------
stopping = 0
ngen = 1
while (stopping < stop_gen){
  if (printl) print(sprintf("Generation %d",ngen))
  offspring = array(0,dim=c(pop_size,n-1))
  off_fitness = array(0,dim=c(pop_size))

  #----------------------------
  # Creating new generation
  #----------------------------
  if (printl) print("Making offspring")
  for (i in 1:pop_size){
    if (runif(1) < cross_prob){
      parents =
sample(1:pop_size,2,replace=FALSE,prob=rankings/sum(rankings))
      for (j in 1:(n-1)){
        if (runif(1) < 1/2) {
          offspring[i,j] = population[parents[1],j]
        } else{
          offspring[i,j] = population[parents[2],j]
        }
      }
    } else {
      parent =
sample(1:pop_size,1,replace=FALSE,prob=rankings/sum(rankings))
```

```
      offspring[i] = population[parent[1]]
      for (j in 1:(n-1)){
        if (runif(1) < mut_prob){
          if (offspring[i,j] == 0){
            offspring[i,j] = 1
          } else{
            offspring[i,j] = 0
          }
        }
      }
    }
  }
  for (i in 1:pop_size){
    if (criteria){
      off_fitness[i] = mdl(offspring[i,1:(n-1)])
    } else{
      off_fitness[i] = aic(offspring[i,1:(n-1)])
    }
  }

  #----------------------------
  # Elitist
  #----------------------------
  #Combine parents and offspring
  total_pop = rbind(population,offspring)
  total_fit = rbind(fitness,off_fitness)
  total_rank = rank(total_pop,total_fit)
  #Keep only n best individuals
  off_rankings = array(0,dim=c(pop_size))
  for (i in 1:pop_size){
    for (j in 1:(2*pop_size)){
      if (total_rank[j] == (2*pop_size+1-i)){
        offspring[i] = total_pop[j]
        off_fitness[i] = total_fit[j]
        off_rankings[i] = pop_size+1-i
      }
    }
  }

  for (i in 1:pop_size){
    if (off_rankings[i] == pop_size){
      print(sprintf("Best Fitness: %f",off_fitness[i]))
      write(sprintf("%d %f",ngen,off_fitness[i]),file=file_name,append =
TRUE)
      chromo = offspring[i,1:(n-1)]
      params = getparams(chromo)
      limits = params[2:(params[1])]
      heights = params[(params[1]+1):(2*params[1])]
      ymodel = getmodel(chromo,limits,heights)
      plot(x,y)
      title(main = sprintf("Generation %d",ngen))
      lines(x,f)
      lines(x,ymodel,col='red')
      break
    }
  }

  i = 1
  while (i <= pop_size){
    if (off_rankings[i]==pop_size){
      j = 1
      while (j <= pop_size){
        if (rankings[j]==pop_size){
          if (off_fitness[i] == fitness[j]){
            stopping = stopping + 1
            if (printl) print(sprintf("Same for %d
generations",stopping))
            i = pop_size
            j = pop_size
          } else{
            stopping = 0
            if (printl) print(sprintf("Not the same..."))
            i = pop_size
            j = pop_size
          }
        }
        j = j + 1
      }
    }
    i = i + 1
  }

  population = offspring
  fitness = off_fitness
  rankings = off_rankings
  ngen = ngen + 1
}

#Best Model
for (i in 1:pop_size){
  if (rankings[i] == pop_size){
    chromo = population[i,1:(n-1)]
    params = getparams(chromo)
    limits = params[2:(params[1])]
    heights = params[(params[1]+1):(2*params[1])]
    ymodel = getmodel(chromo,limits,heights)
    plot(x,y)
    title(main = sprintf("Generation %d",ngen-1))
    lines(x,f)
    lines(x,ymodel,col='blue')
    break
  }
```

```
}
```

	$S = 300, N_{\text{stop}} = 20$	$S = 500, N_{\text{stop}} = 30$
MDL		
AIC		

	$S = 300, N_{\text{stop}} = 20$	$S = 500, N_{\text{stop}} = 30$
MDL		
AIC	 Generation 98 y x	 Generation 56 y x