Replicating a Model:

Adaptation as Information Restriction: The Hot Stove Effect

By Jerker Denrell and James G. March

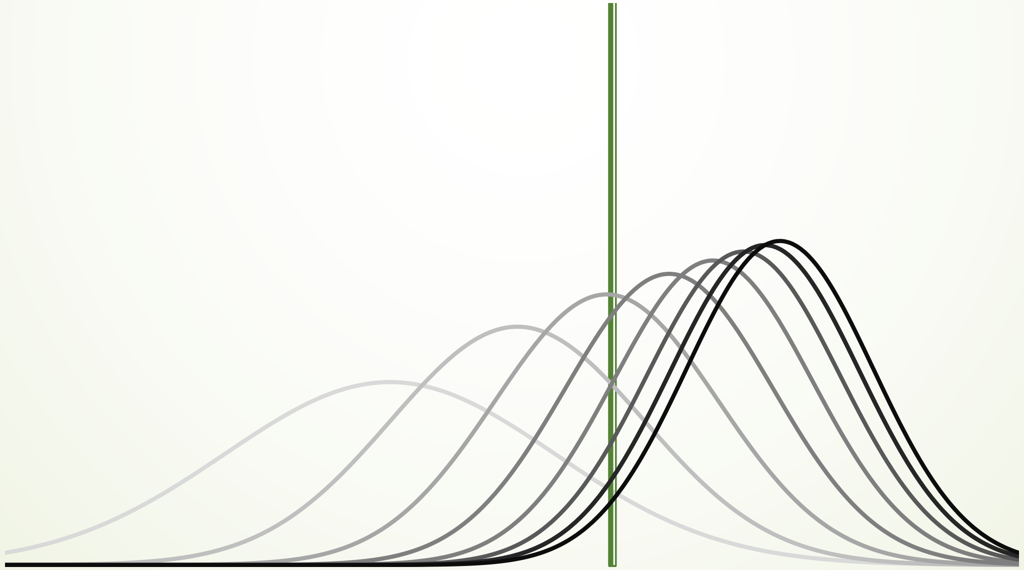
Presented by James Paine
February 22, 2019

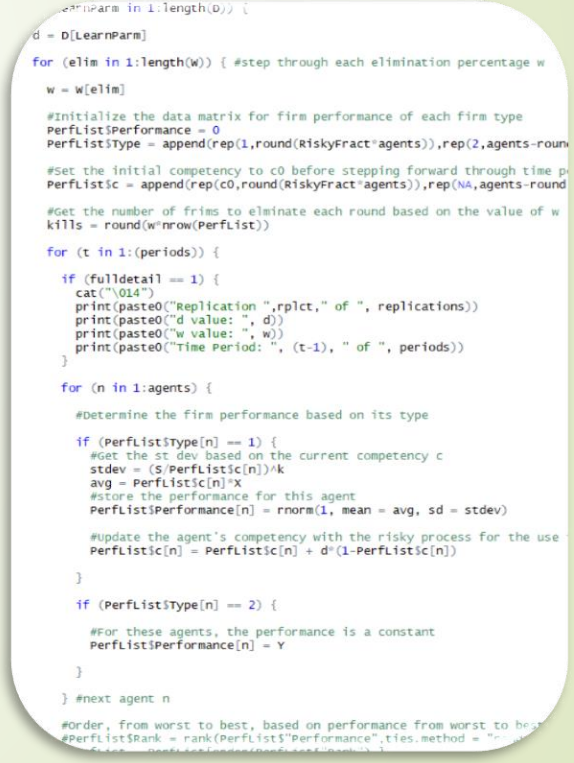

Adaptation as Information Restriction:
The Hot Stove Effect

Jerker Denrell • James G. March

Abstract

Adaptation as Sequential Sampling



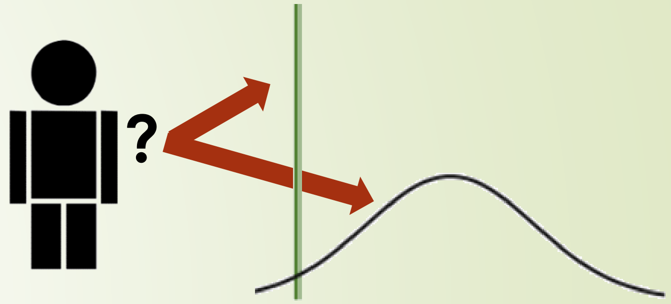


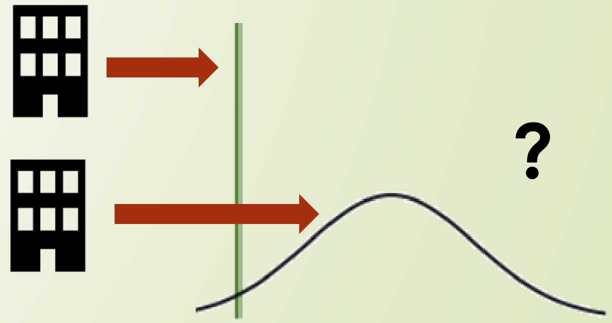
Paper Background

- Published in Organization Science in 2001
- “Hot Stove” effect can be summarized as the over-correction (or basis) away from risky behavior due to early negative experiences
- Modeling is based on individual/firm choosing one of two alternatives
  - One with ‘certain’ outcomes
  - Another with ‘risky’ (normally distributed) outcomes
- ‘Aspiration level’ as benchmark of outcomes for agent-level simulations
- Relative performance with thresholding as benchmark for multi-agent (or multi-firm) simulations
- ‘Competence’ as mechanism for experiential learning improvements

Model Details

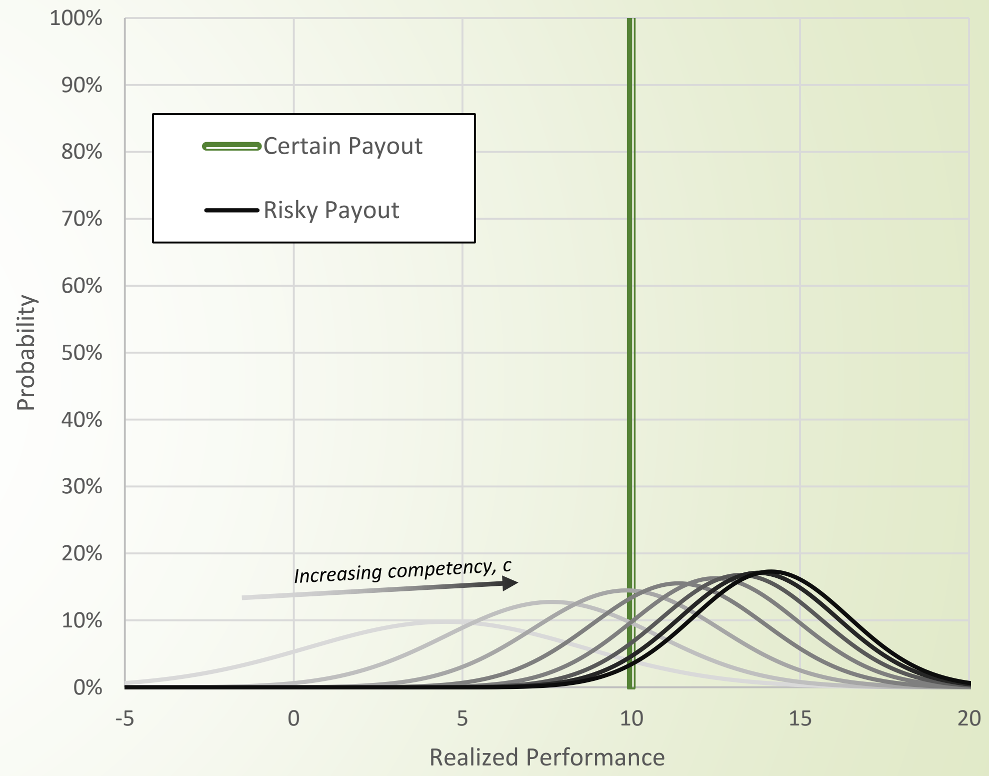
- Experiential Learning
  - 5,000 simulated individuals draw from either a 'certain' or 'risky' (normally distributed) payout
  - Payout histories are recorded as the 'aspiration level'
  - IF the payout exceeds the aspiration level, the probability of choosing that choice increases *linearly as a function of a parameter 'a'*
  - Explores the number of individuals still making 'risky' choices after 50 time steps as a function of the linear parameter *a*
- Competitive Selection
  - 100 simulated firms are categorized as always either drawing from the risky or certain payouts
  - Overall performance is assed after each period, with some fraction *w* of the worst performers being replaced
  - New firm are populated each round proportionally (or randomly)
  - Explores the number of firms of the risky type still surviving at the end of 50 rounds as a function of the parameter *w*



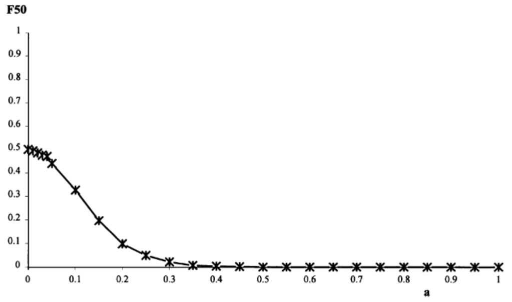
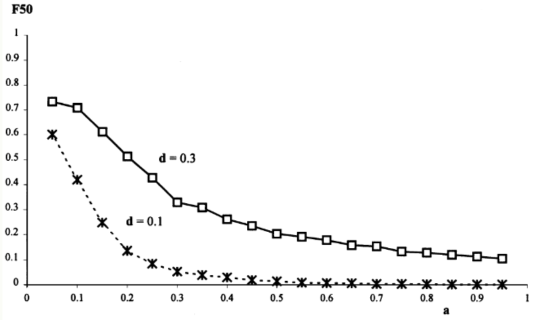
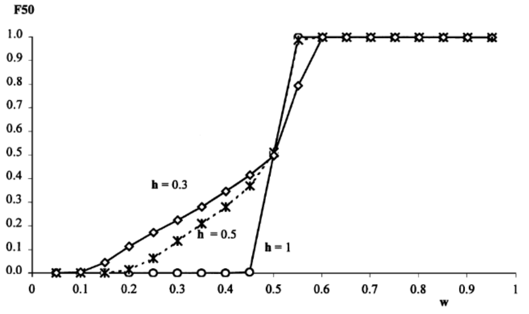
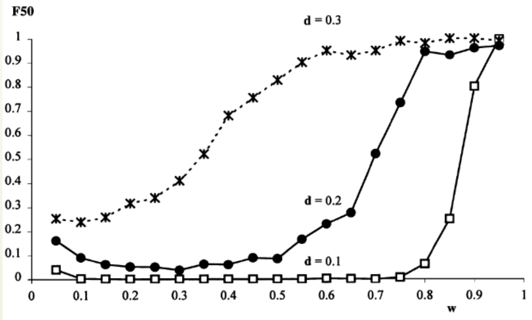


Model Details

- Competency as an Extension to both models
  - Competency c_t is analogous to learning by doing
  - After an individual/firm draws from the risky payout, their competency increases
    - $c_{t+1} = c_t + d(1 - c_t)$
    - This increases the expectation $\mu_t = c_t X$
    - And reduces the variance $\sigma_t = \left(\frac{S}{c_t}\right)^k$
  - The more times an individual samples the risky distribution, or the longer a risky firm survives, the better the expectation
  - Varying *d* varies the speed at which competency approaches 1



Model Details

	Without Competency Mechanism	With Competency Mechanism
Experiential	**Figure 1** **The Fraction of 5,000 Individuals Who Choose the Risky Alternatives at the End of Period 50 as a Function of** *a*. **Based on Averages from 25 Sets of 5,000 Simulations Where** $X = 10$, $Y = 10$, $S = 10$, $b = 0.5$  	**Figure 2** **The Fraction of 5,000 Individuals Who Choose the New Alternative at the End of Period 50 as a Function of** *a* **and** *d*. **Each Line Is Based on Averages from 19 Sets of 5,000 Simulations Where** $X = 15$, $Y = 10$, $S = 5$, $k = 0.5$, $c_0 = 0.3$, **and** $b = 0.5$ 
Competitive Selection	**Figure 3** **The Fraction of the Population Who Choose the Risky Alternative at the End of Period 50 as a Function of** *w* **and** *h*. **Each Line Is Based on Averages from 19 Sets of 1,000 Simulations Where** $X = 10$, $Y = 10$, **and** $S = 10$. **The Constant Population Size is 100.**  	**Figure 4** **The Fraction of the Population Who Choose the New Alternative at the End of Period 50 as a Function of** *w* **and** *d* **when** $h = 0.5$. **Each Line Is Based on Averages from 19 Sets of 100 Simulations Where** $X = 15$, $Y = 10$, $S = 5$, $k = 0.5$, **and** $c_0 = 0.3$. **The Constant Population Size is 100.** 

Replication

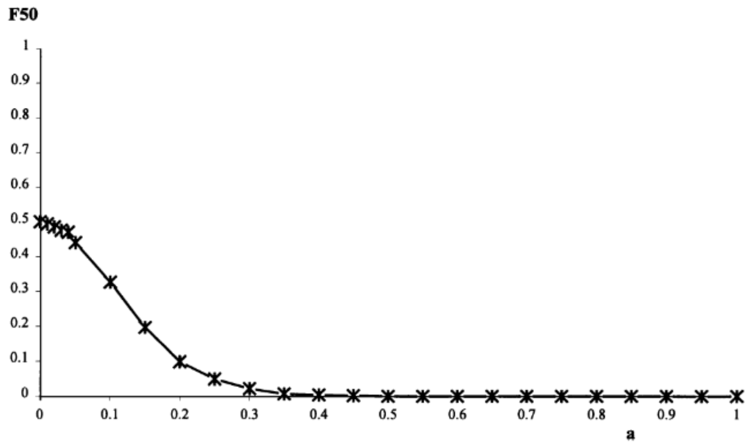
- Recreated the 4 model variations
  - R as the programming language
  - Two structural methods used:
    - Experiential: Array based memory allocation (full history retained)
    - Competitive: Data Frame based memory allocation (history overwritten)
  - Code written to be flexible to explore full parameter spaces
  - Additional mechanisms and triggers incorporated
    - Competitive: cumulative performance and inter-generational competency transfer
- Matched the model terminology and structure from Denrell and March
- For all but one, used the published parameter values and visually compared my results to Denrell and March

```
d = D[LearnParm]

for (elim in 1:length(W)) { #step through each eliminat

  w = W[elim]

  #Initialize the data matrix for firm performance of e
  PerfList$Performance = 0
  PerfList$Type = append(rep(1,round(RiskyFract*agents)

  #Set the initial competency to c0 before stepping fo
  PerfList$c = append(rep(c0,round(RiskyFract*agents))

  #Get the number of frims to elminate each round base
  kills = round(w*nrow(PerfList))

  for (t in 1:(periods)) {

    if (fulldetail == 1) {
      cat("\014")
      print(paste0("Replication ",rplct," of ", replica
      print(paste0("d value: ", d))
      print(paste0("w value: ", w))
      print(paste0("Time Period: ", (t-1), " of ", per
    }

    for (n in 1:agents) {

      #Determine the firm performance based on its type

      if (PerfList$Type[n] == 1) {
        #Get the st dev based on the current competency
        stdev = (S/PerfList$c[n])^k
        avg = PerfList$c[n]*X
        #store the performance for this agent
        PerfList$Performance[n] = rnorm(1, mean = avg,

        #Update the agent's competency with the risky p
        PerfList$c[n] = PerfList$c[n] + d*(1-PerfList$c

      }

      if (PerfList$Type[n] == 2) {

        #For these agents, the performance is a constan
        PerfList$Performance[n] = Y

      }

    } #next agent n
```

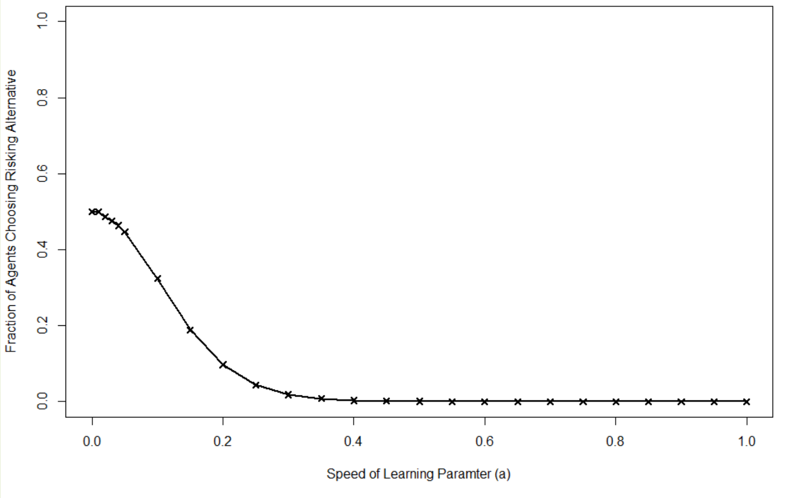
Experiential Model Recreations – Directional Results

Paper Outputs

Figure 1 **The Fraction of 5,000 Individuals Who Choose the Risky Alternatives at the End of Period 50 as a Function of** a**. Based on Averages from 25 Sets of 5,000 Simulations Where** $X = 10$, $Y = 10$, $S = 10$, $b = 0.5$

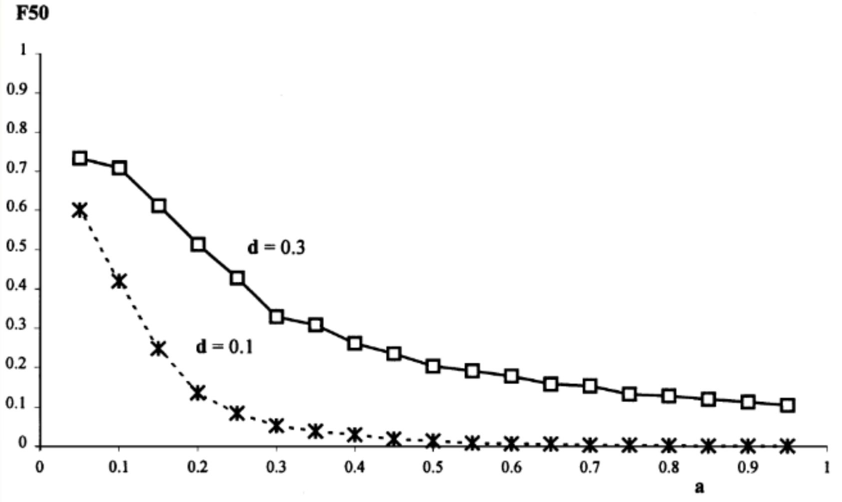


Replication outputs

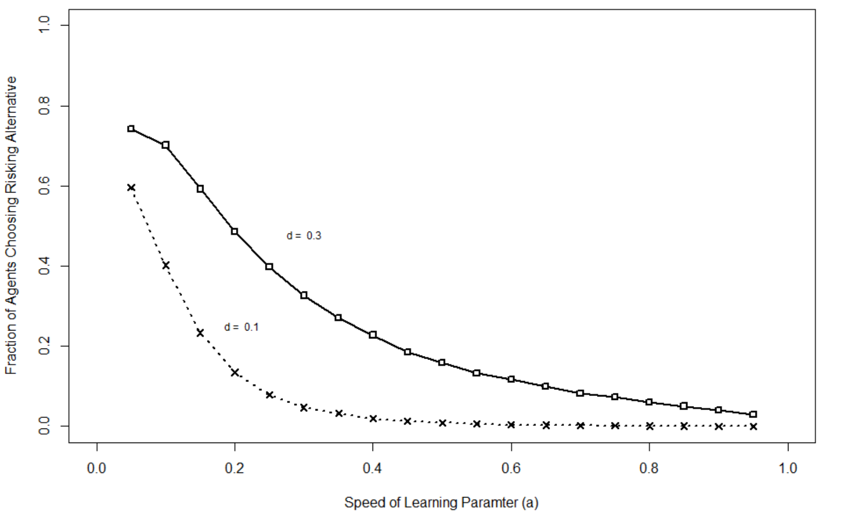


Paper Outputs

Figure 2 **The Fraction of 5,000 Individuals Who Choose the New Alternative at the End of Period 50 as a Function of** a **and** d**. Each Line Is Based on Averages from 19 Sets of 5,000 Simulations Where** $X = 15$, $Y = 10$, $S = 5$, $k = 0.5$, $c_0 = 0.3$, **and** $b = 0.5$



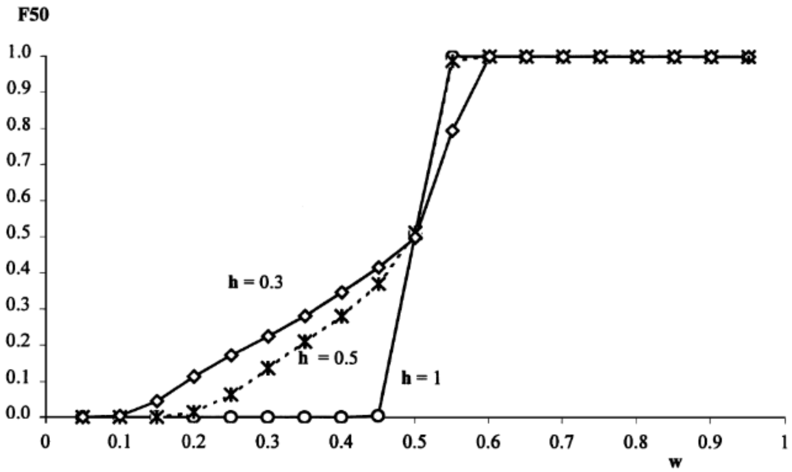
Replication outputs



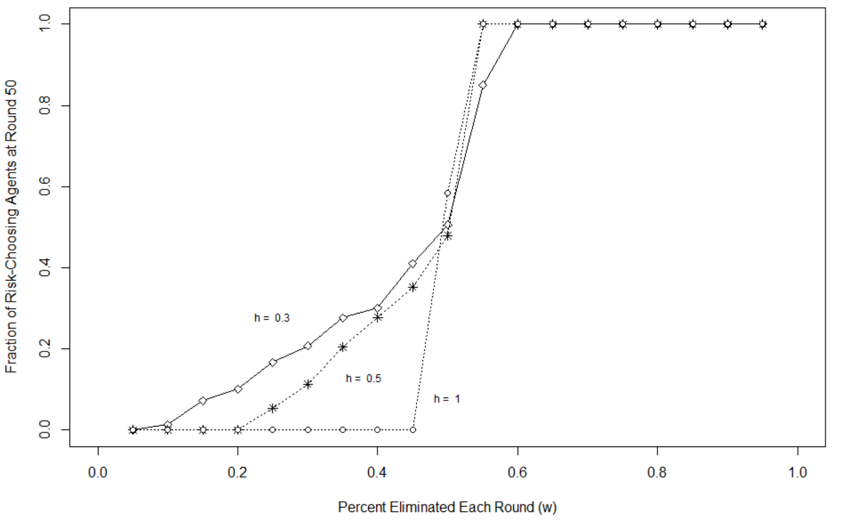
Competitive Selection Model Recreations – Directional Results

Paper Outputs

Figure 3 **The Fraction of the Population Who Choose the Risky Alternative at the End of Period 50 as a Function of *w* and *h*. Each Line Is Based on Averages from 19 Sets of 1,000 Simulations Where $X = 10$, $Y = 10$, and $S = 10$. The Constant Population Size is 100.**

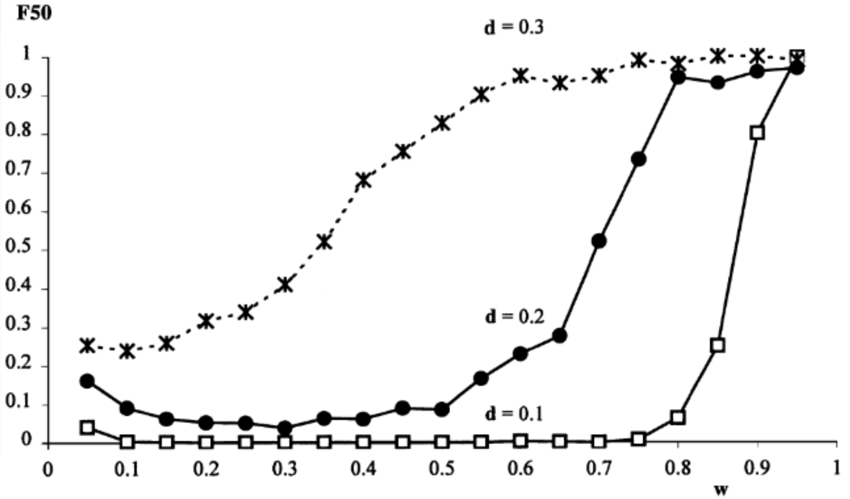


Replication outputs

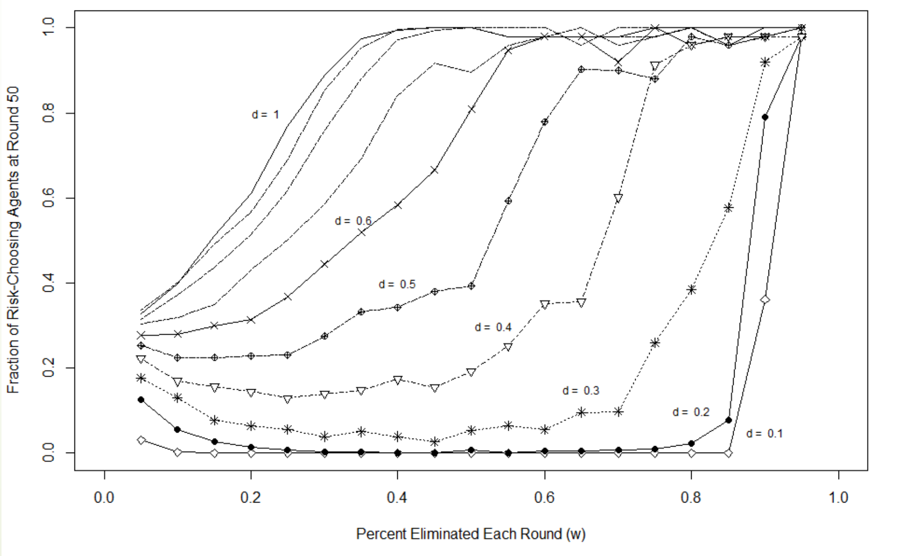


Paper Outputs

Figure 4 **The Fraction of the Population Who Choose the New Alternative at the End of Period 50 as a Function of *w* and *d* when $h = 0.5$. Each Line Is Based on Averages from 19 Sets of 100 Simulations Where $X = 15$, $Y = 10$, $S = 5$, $k = 0.5$, and $c_0 = 0.3$. The Constant Population Size is 100.**



Replication outputs*



R Script Demos

	Without Competency Mechanism	With Competency Mechanism
Experiential	1-BasicHotStove.R	2-CompetenceHotStove.R
Competitive Selection	3-SurvivalHotStove.R	4-CompetenceSurvivalHotStove.R

Critiques on Model Formulation

- Exact number of replications of the model is obfuscated, and only implied from the graph descriptions
  - For the experiential learning models, the 5000 agents are implied to be run only once
  - For the Competitive Selection models, the first model is implied to be repeated 10 times while the second appears to have been *run only once*
- In experiential learning, the aspirations are one-sided and probability updates are linear
  - Aspirations are relative to choice 1 only (not immediately clear in the paper)
  - Changes to probability are only *if* aspiration is exceed/missed, and then change by a constant factor no matter the distance of the realization from the aspiration
  - Not immediately clear on how to extend to >2 choices

The Fraction of 5,000 Individuals Who Choose the Risky Alternatives at the End of Period 50 as a Function of *a*. Based on Averages from 25 Sets of 5,000 Simulations Where $X = 10$, $Y = 10$, $S = 10$, $b = 0.5$

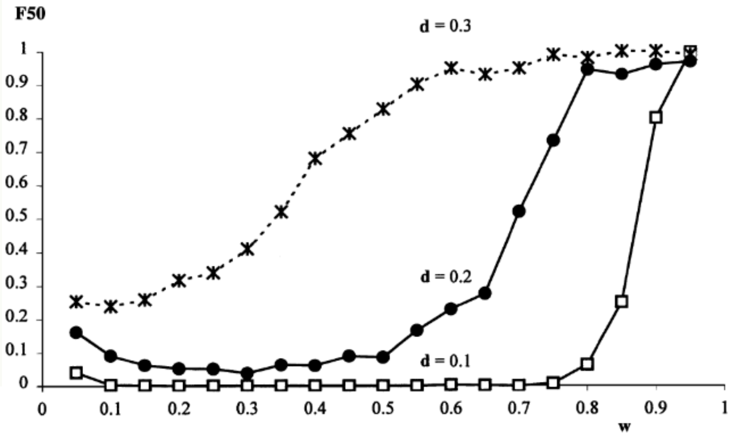
$$\mathrm{L}_{t+1} = \mathrm{L}_t(1-\mathrm{b}) + \mathrm{O}_t\mathrm{b}$$

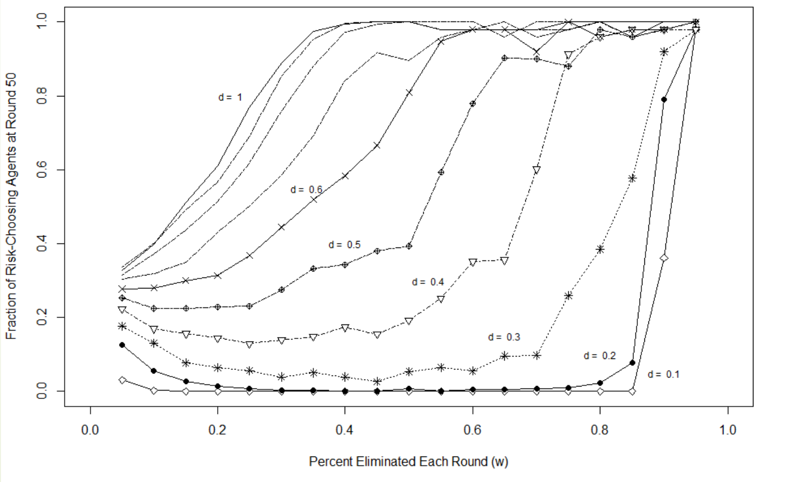
$$P_{t+1} = \begin{cases} P_t + a(1-P_t) & if\ O_t > L_t \\ (1-a)P_t & if\ O_t < L_t \\ P_t & o.w. \end{cases}$$

Critiques on Paper Reproducibility

- Figure 4 (Competitive Selection with competency learning) was difficult to reproduce
  - Authors appear to only have run the model once
  - Full envelop of survivability as a function of w and *d* missing from the paper
    - When looking at the full envelop, my model appears to be off by a factor of two w.r.t. *d* values
    - Have recreated the model multiple times from scratch, and still have the same discrepancy
  - However when looking at the full envelop in my simulation, the directional results hold

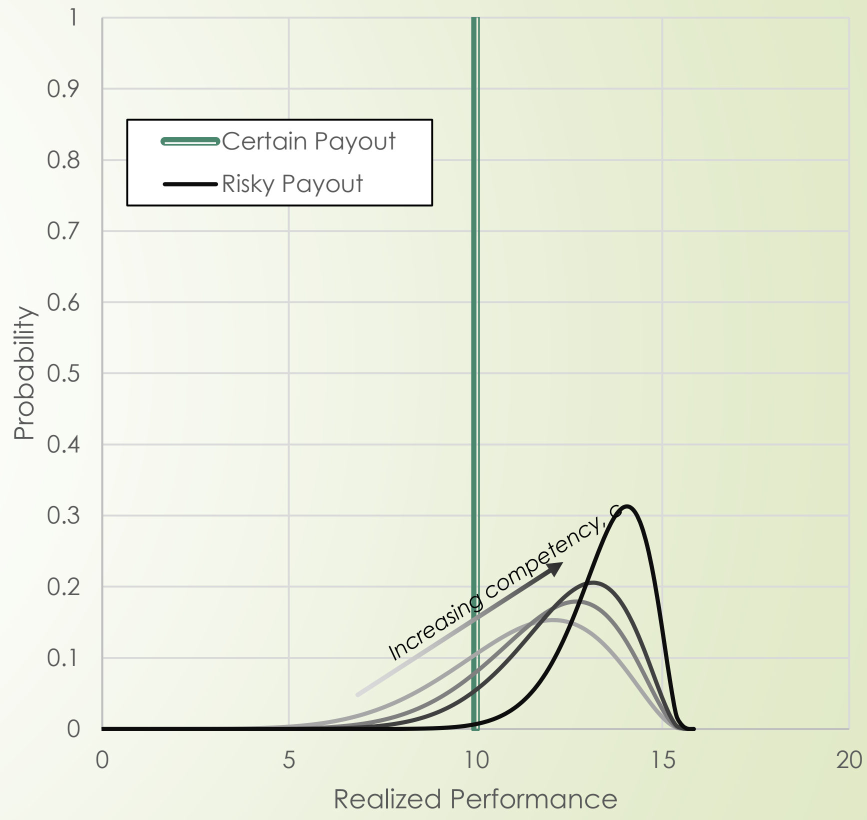
Figure 4 **The Fraction of the Population Who Choose the New Alternative at the End of Period 50 as a Function of** *w* **and** *d* **when** $h = 0.5$. **Each Line Is Based on Averages from 19 Sets of 100 Simulations Where** $X = 15$, $Y = 10$, $S = 5$, $k = 0.5$, **and** $c_0 = 0.3$. **The Constant Population Size is 100.**





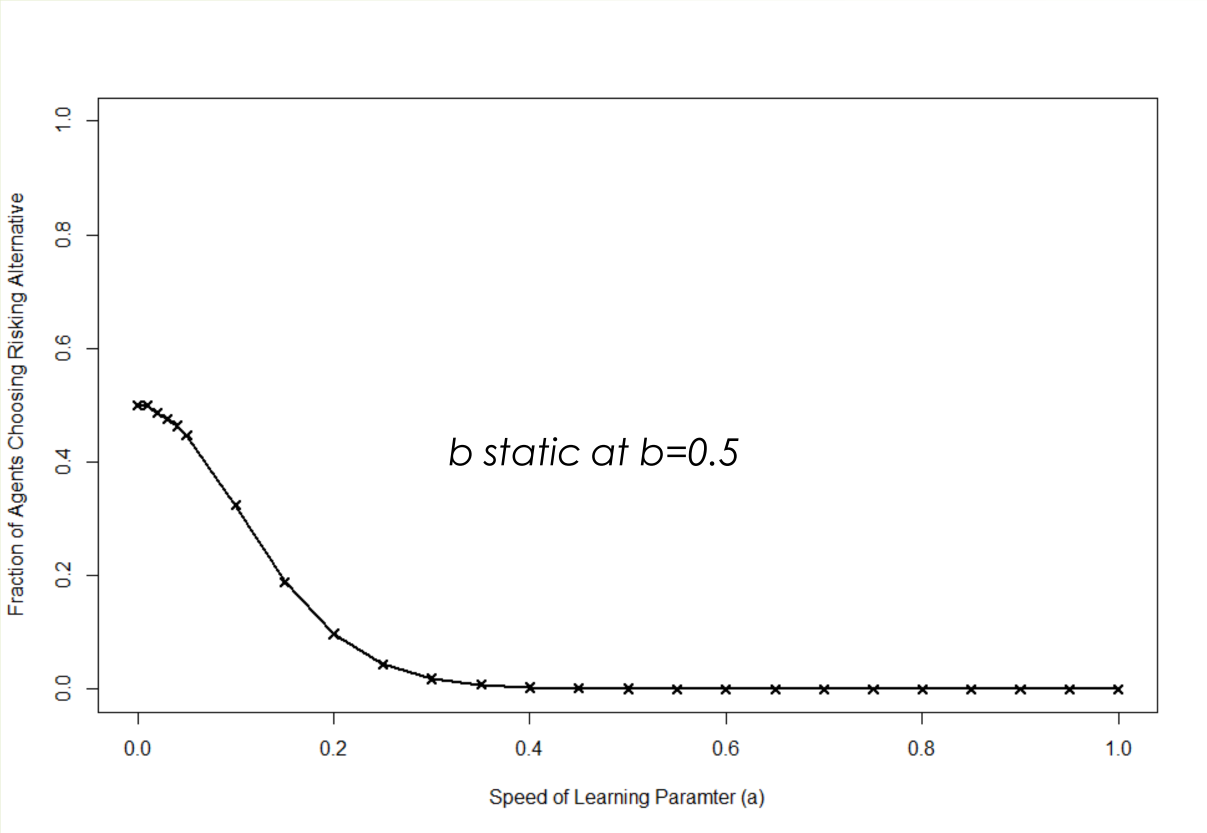
Extensions and Suggestions

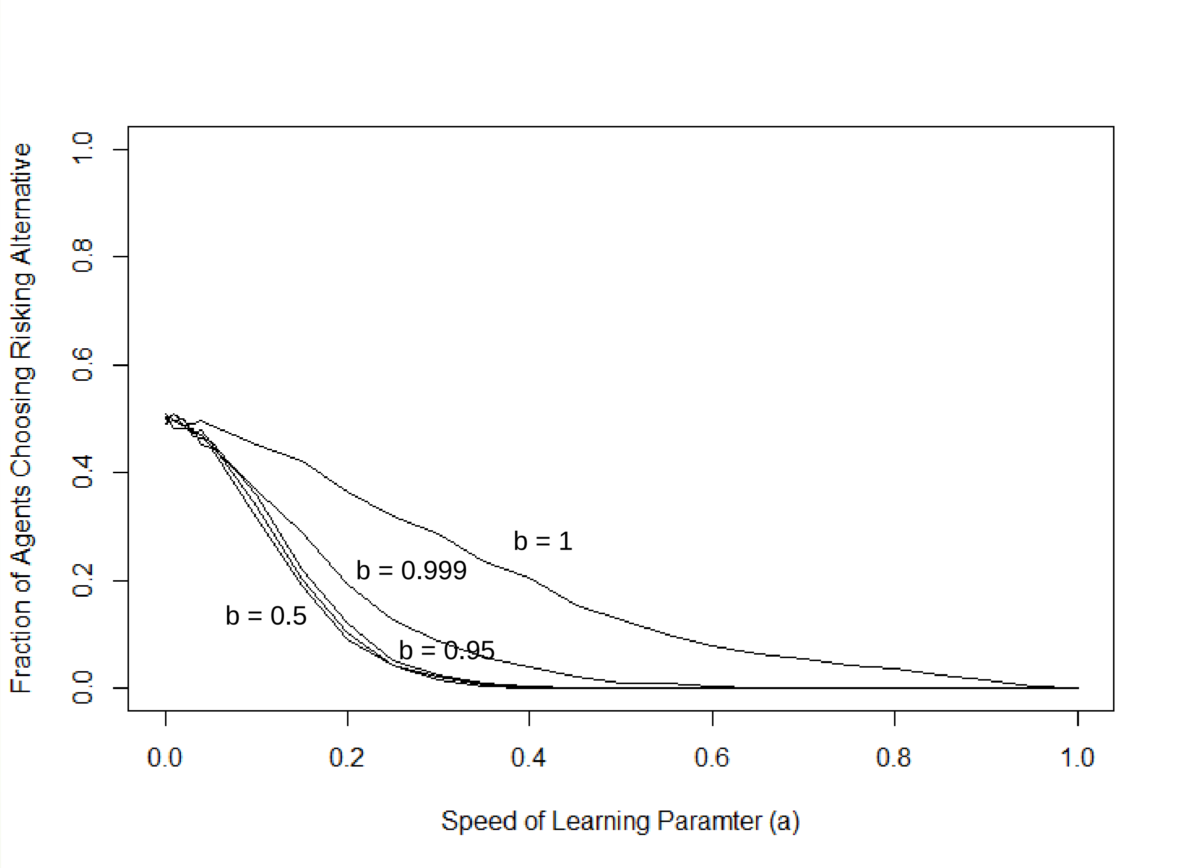
- Create more generalized envelopes of parameter interactions
- Vary at least over all parameters mentioned in the model
  - The authors do not vary over b and only talk about varying over k
- Show all results, specifically learning transference in the competitive survival models
- Make competitive survival more realistic
  - Start with low number of risky firms
  - Perhaps use beta distribution to reflect more realistic starting state for new risky process
  - Evaluate based on *cumulative* performance, not period-by-period
- Explore dynamics of the system in the time periods before t=50



Extensions and Suggestions

- Varying over additional parameters, like b as seen below, yields additional behavior





Increasing b (rate at which aspiration incorporates new information) yields threshold near 1 at which 'Risky' agent suddenly becomes much more prevalent in the system

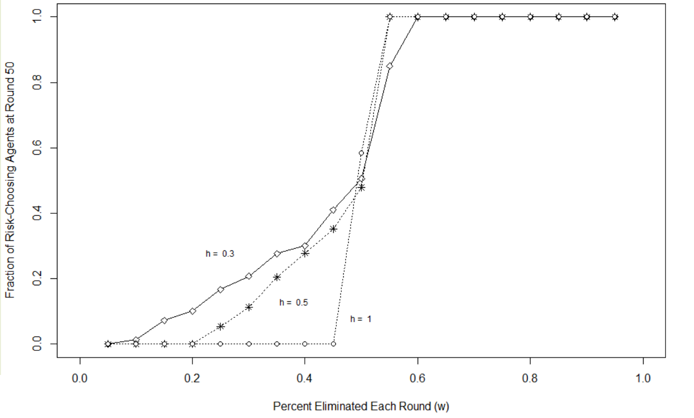
Extensions and Suggestions

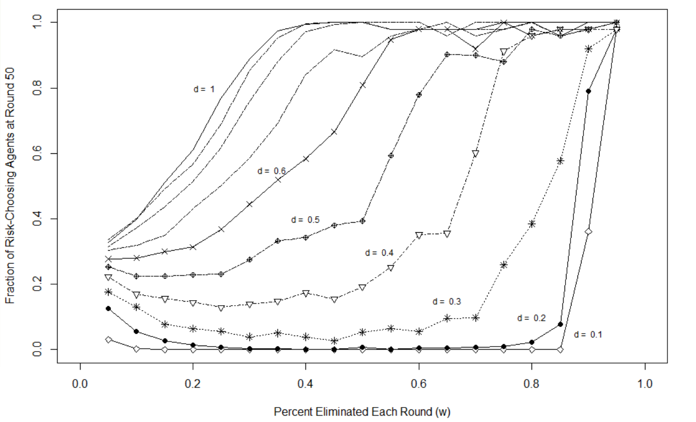
- Evaluate cumulative survival based on *cumulative* performance

Without Competency Building

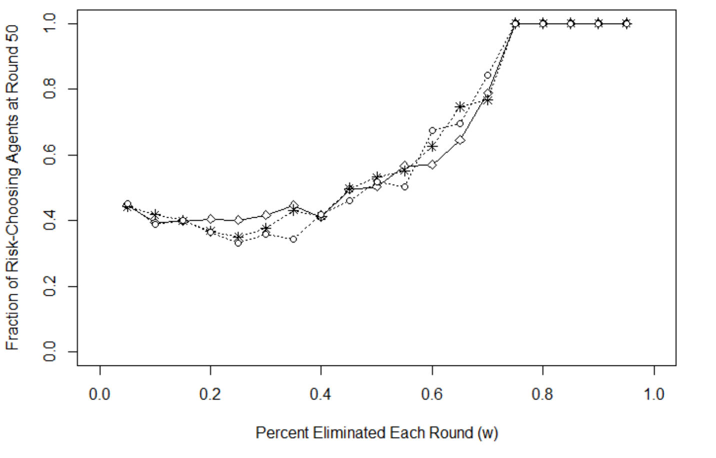
With Competency Building

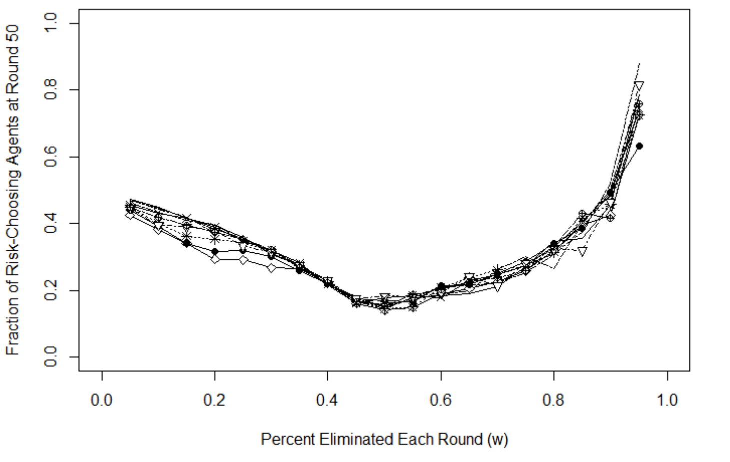
Period-By-Period





Cumulative





d and h matter less in aggregate when performance is cumulative, and the behavior is more generalizable

```r
#Model Replication
#Adaptation as Information Restriction
#Jerker Denrell, James G. March, 2001

#Created by James Paine
#15.879 - Simulation Models in Social and Behavioral Sciences
#February 22, 2019

#####PART 1 - EXPERIENTIAL LEARNING WITHOUT COMPETENCY BUILDING#####


###System Parameters

#Expectation of the 'risky' alternative
X = 10
#Standard deviation of the 'risky alternative
S = 10

#Expectation of the 'certain' alternative
Y = 10

#Rate at which aspirations adjust to experience
b= 0.5


###Simulation Paramters
periods = 50
agents = 5000
replications = 2

A = seq(from = 0, to = 0.045, by = ((0.05-0)/5))
A = append(A,seq(from = 0.05, to = 1, by = ((1-0)/20)))
A

#Note P is the probability that the RISKY alternative is chosen
#This is a two-choice system, so the probability of the certain
#  choice is 1-P for any timestep t


#Note on array indicides
#P[n,t,a] and L[n,t,a] and k[n,t,a] where
# n = agent number
# t = period number
# P = probability of choosing the risky choice
# example: P[50,20] = 0.25 means that agent #50 had a 25% chance of
#   choosing the risky choice at time period 20

#IN the below sampling, choice 1 is the 'Risky' alternative

#Initialize the probability array P and the aspiration array L
#initalize the GLOBAL probability and q array spaces
  P = array(data = NA, dim = c(agents,periods+1,length(A),replications))
  L = array(data = NA, dim = c(agents,periods+1,length(A),replications))
  k = array(data = NA, dim = c(agents,periods+1,length(A),replications))
  FractX = array(data = NA, dim = c(periods+1,length(A),replications))

##Initialize the simluation
```

```
  #Initialize the probabilities at t=0
  P[,1,,]=0.5

  #Initialize the asipriations at t=0
  L[,1,,] = P[,1,,]*X+(1-P[,1,,])*Y

#debug param
n=1
t=1
SoL = 1
rep = 1
#

for (rep in 1:replications) {

  for (SoL in 1:length(A)) {

    a = A[SoL]

    for (t in 1:(periods+1)) {

      cat("\014")
      print(paste0("Replication ", rep," of ", replications))
      print(paste0("a value: ", a))
      print(paste0("Time Period: ", (t-1), " of ", periods))


      for (n in 1:agents) {

        #print(paste0("Agent: ", n, " of ", agents))

        #Probabilistically get agent n's choice

k[n,t,SoL,rep]=sample(c(1:2),1,replace=FALSE,prob=c(P[n,t,SoL,rep],(1-
P[n,t,SoL,rep])))

        #Get the realized value from either the risky or certain
distributions
        if (k[n,t,SoL,rep] == 1) {
          O = rnorm(1, mean = X, sd = S)
        } else {
          O = Y
        }


        if ((t+1)<=(periods+1)) {

          #Get the aspirations for next round
          L[n,t+1,SoL,rep] = L[n,t,SoL,rep]*(1-b)+O*b

          #Update the probability of choosing the risky alternative based on
the
          #   current aspiration L and the realized value O

          P[n,t+1,SoL,rep] = P[n,t,SoL,rep]
```

```
          #"...if the first alternative is tried..."
          if (k[n,t,SoL,rep] == 1) {
            #"...AND yields an outcome better than the asipration at time t"
            if (O > L[n,t,SoL,rep]) {
              #"The probability increases in the following way:"
              P[n,t+1,SoL,rep]=P[n,t,SoL,rep] + a*(1-P[n,t,SoL,rep])

            #"...OR yields an outcome that is worse than the asipriation
            } else if (O < L[n,t,SoL,rep]) {
              P[n,t+1,SoL,rep] = (1-a)*P[n,t,SoL,rep]

            }
          }

          #...or if the second alternative is tried..."
          if (k[n,t,SoL,rep] == 2) {
            #"...AND yields an outcome worse than the asipration at time t"
            if (O < L[n,t,SoL,rep]) {
              #"The probability increases in the following way:"
              P[n,t+1,SoL,rep]=P[n,t,SoL,rep] + a*(1-P[n,t,SoL,rep])

              #"...OR yields an outcome that is better than the asipriation
            } else if (O > L[n,t,SoL,rep]) {
              P[n,t+1,SoL,rep] = (1-a)*P[n,t,SoL,rep]

            }

          }

        }


      } # Next Agent n

      # Determine fractions of choices at each time step (t) and Speed of
Learning (a) value
      NumX = sum(k[,t,SoL,rep] == 1)
      NumY = sum(k[,t,SoL,rep] == 2)
      FractX[t,SoL,rep] = NumX/(NumX+NumY)


    } # Next time period t


  } # Next parameter a value

} # Next replication


#Average each replication and determine the standard deviation
FractX_avg = apply(FractX, c(1,2), mean)
FractX_sd = apply(FractX, c(1,2), sd)

#Get Confidence intervals for plotting
CI = 0.999
SD_mod = qnorm(1-CI,lower.tail=FALSE)/sqrt(replications)
FractX_UCI = FractX_avg + SD_mod*FractX_sd
```

```
FractX_LCI = FractX_avg - SD_mod*FractX_sd
FractX_UCI[FractX_UCI>1] = 1
FractX_LCI[FractX_LCI<0] = 0


#Full plot of all rounds of last replication
Rep = replications
matplot(A,t(FractX[,,Rep]), typ = "l", pch=4, col = "grey", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative"
)


#Plot of last round of specific replication
Round = periods
Rep = 2
plot(A,FractX[Round,,Rep], typ = "l", pch=4, col = "black", lwd=2, lty =1,
     xlim=c(0,1), ylim=c(0,1),
     xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative"
)
points(A,FractX[Round,,Rep], typ = "p", pch=4, lwd=2)


#Full plot all rounds of the mean of all replications
matplot(A,t(FractX_avg[,]), typ = "l", pch=4, col = "grey", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative"
)

##FIGURE 1 RECREATION
Round = periods
plot(A,FractX_avg[Round,], typ = "l", pch=4, col = "black", lwd=2, lty =1,
     xlim=c(0,1), ylim=c(0,1),
     xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative"
)
points(A,FractX_avg[Round,], typ = "p", pch=4, lwd=2)

#arrows(A,FractX_LCI[Round,], A, FractX_UCI[Round,], length=0.05, angle=90,
code=3)
```

```
#Model Replication
#Adaptation as Information Restriction
#Jerker Denrell, James G. March, 2001

#Created by James Paine
#15.879 - Simulation Models in Social and Behavioral Sciences
#February 22, 2019

#####PART 2 - EXPERIENTIAL LEARNING WITH COMPETENCY BUILDING#####


###System Parameters

#Expectation of the 'risky' or 'new' alternative
X = 15
#Standard deviation of the 'risky' or 'new' alternative
S = 5

#Expectation of the 'certain' alternative
Y = 10

#Rate at which aspirations adjust to experience
b= 0.5

#Initial Competence of agents with the 'risky alternative
c0 = 0.3

#Rate of standard deviation decrease as a function of competence (see page
527)
k = 0.5


###Simulation Paramters
periods = 50
agents = 5000
replications = 5

#Display full iteration details - adds time, only use for small replications
fulldetail = 0

#Create arrays to range over for later plotting
A = seq(from = .05, to = 0.95, by = ((0.95-0.05)/18))
D = c(0.1,0.3)

#Note P is the probability that the RISKY alternative is chosen
#This is a two-choice system, so the probability of the certain
#  choice is 1-P for any timestep t


#Note on array indicides
#P[n,t,a] and L[n,t,a] and choice[n,t,a] where
# n = agent number
# t = period number
# P = probability of choosing the risky choice
# example: P[50,20] = 0.25 means that agent #50 had a 25% chance of
#   choosing the risky choice at time period 20
```

```
#IN the below sampling, choice 1 is the 'Risky' alternative

#Initialize the probability array P and the aspiration array L
#initalize the GLOBAL probability and q array spaces
  P = array(data = NA, dim =
c(agents,periods+1,length(A),length(D),replications))
  L = array(data = NA, dim = c(agents,periods+1,length(A),
length(D),replications))
  choice = array(data = NA, dim = c(agents,periods+1,length(A),
length(D),replications))
  FractX = array(data = NA, dim = c(periods+1,length(A),
length(D),replications))
  c = array(data = NA, dim = c(agents,periods+1,replications))
##Initialize the simluation

  #Initialize the probabilities at t=0
  P[,1,,,]=0.5

  #Initialize the asipriations at t=0
  L[,1,,,] = P[,1,,,]*X+(1-P[,1,,,])*Y

  #Initialize the competencies at t=0
  c[,1,] = c0

n=1
t=1
SoL = 1
LearnParm = 1
SoL = 1

starttime = proc.time()

for (rep in 1:replications) {

  if (fulldetail != 1) {
    cat("\014")
    print(paste0("Replication ",rep," of ", replications))
    print(paste0("Elapsed time since last replication: ",(proc.time() -
starttime)[3],"s"))
    print(paste0("Avg Replication time: ",(proc.time() - starttime)[3]/rep,"s
per replication"))
    print(paste0("Est Time Remaining: ",((proc.time() -
starttime)[3]/rep)*(replications-rep),"s"))


  }

  for (LearnParm in 1:length(D)) {

    d = D[LearnParm]

    for (SoL in 1:length(A)) {  #Note: a is the 'speed of learning'

      a = A[SoL]

      for (t in 1:(periods+1)) {
```

```
        if (fulldetail == 1) {
          cat("\014")
          print(paste0("Replication ",rep," of ", replications))
          print(paste0("d value: ", d))
          print(paste0("a value: ", a))
          print(paste0("Time Period: ", (t-1), " of ", periods))
        }

        for (n in 1:agents) {

          #print(paste0("Agent: ", n, " of ", agents))

          #Probabilistically get agent n's choice
          # Note, here choice 1 is the 'risky' or 'new' alternative

choice[n,t,SoL,LearnParm,rep]=sample(c(1:2),1,replace=FALSE,prob=c(P[n,t,SoL,
LearnParm,rep],(1-P[n,t,SoL,LearnParm,rep])))

          #Get the realized value from either the risky or certain
distributions
          if (choice[n,t,SoL,LearnParm,rep] == 1) {

            #Get the st dev based on the current competency c
            stdev = (S/c[n,t,rep])^k
            avg = c[n,t,rep]*X
            #Get the performance experienced
            O = rnorm(1, mean = avg, sd = stdev)
          } else {
            O = Y
          }


          if ((t+1)<=(periods+1)) {

            #Get the aspirations for next round
            L[n,t+1,SoL,LearnParm,rep] = L[n,t,SoL,LearnParm,rep]*(1-b)+O*b

            #Update the probability of choosing the risky alternative based
on the

            #   current aspiration L and the realized value O

            P[n,t+1,SoL,LearnParm,rep] = P[n,t,SoL,LearnParm,rep]

            #"...if the first/new alternative is tried..."
            if (choice[n,t,SoL,LearnParm,rep] == 1) {

              #"...Competence increases with each utilization"
              c[n,t+1,rep]=c[n,t,rep] + d*(1-c[n,t,rep])

              #"...AND yields an outcome better than the asipration at time
t"
              if (O > L[n,t,SoL,LearnParm,rep]) {
                #"The probability increases in the following way:"
                P[n,t+1,SoL,LearnParm,rep]=P[n,t,SoL,LearnParm,rep] + a*(1-
P[n,t,SoL,LearnParm,rep])

              #"...OR yields an outcome that is worse than the asipriation
```

```
          } else if (O < L[n,t,SoL,LearnParm,rep]) {
            P[n,t+1,SoL,LearnParm,rep] = (1-a)*P[n,t,SoL,LearnParm,rep]

          }
        }

        #...or if the second/existing alternative is tried..."
        if (choice[n,t,SoL,LearnParm,rep] == 2) {

          c[n,t+1,rep]=c[n,t,rep]

          #"...AND yields an outcome worse than the asipration at time t"
          if (O < L[n,t,SoL,LearnParm,rep]) {
            #"The probability increases in the following way:"
            P[n,t+1,SoL,LearnParm,rep]=P[n,t,SoL,LearnParm,rep] + a*(1-
P[n,t,SoL,LearnParm,rep])

            #"...OR yields an outcome that is better than the asipriation
          } else if (O > L[n,t,SoL,LearnParm,rep]) {
            P[n,t+1,SoL,LearnParm,rep] = (1-a)*P[n,t,SoL,LearnParm,rep]

          }

        }

      }


    } # Next Agent n

    # Determine fractions of choices at each time step (t) and Speed of
Learning (a) value
    NumX = sum(choice[,t,SoL,LearnParm,rep] == 1)
    NumY = sum(choice[,t,SoL,LearnParm,rep] == 2)
    FractX[t,SoL,LearnParm,rep] = NumX/(NumX+NumY)


   } # Next time period t


  } # Next speed of learning parameter a value

 } # Next learning paramter d value

} # Next replication


#Average each replication and determine the standard deviation
FractX_avg = apply(FractX, c(1,2,3), mean)
FractX_sd = apply(FractX, c(1,2,3), sd)

#Plot specific repitition and round combo
rep = 1
Round = 50
plot(A,FractX[Round,,1,rep], typ = "l", col = "black", lwd=2, lty =1,
     xlim=c(0,1), ylim=c(0,1),
```

```
     xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative")
  points(A,FractX[Round,,1,rep], typ = "p", pch=4, lwd=2)
  lines(A,FractX[Round,,2,rep], type = "l", col = "black", lwd=2, lty=1)
  points(A,FractX[Round,,2,rep], typ = "p", pch=0, lwd=2)
  text(c(0.21,0.3),c(0.25,0.48),paste("d = ", D), cex = .75)


##FIGURE 2

Round = 50
plot(A,FractX_avg[Round,,1], typ = "l", col = "black", lwd=2, lty =3,
     xlim=c(0,1), ylim=c(0,1),
     xlab = "Speed of Learning Paramter (a)", ylab = "Fraction of Agents
Choosing Risking Alternative")
points(A,FractX_avg[Round,,1], typ = "p", pch=4, lwd=2)
lines(A,FractX_avg[Round,,2], type = "l", col = "black", lwd=2, lty=1)
points(A,FractX_avg[Round,,2], typ = "p", pch=22, lwd=2, bg = "white")
text(c(0.21,0.3),c(0.25,0.48),paste("d = ", D), cex = .75)
```

```
#Model Replication
#Adaptation as Information Restriction
#Jerker Denrell, James G. March, 2001

#Created by James Paine
#15.879 - Simulation Models in Social and Behavioral Sciences
#February 22, 2019

#####PART 3 - COMPETITIVE SURVIVAL WITHOUT COMPETENCY BUILDING#####

###System Parameters

#Expectation of the 'risky' or 'new' alternative
X = 10
#Standard deviation of the 'risky' or 'new' alternative
S = 10

#Expectation of the 'certain' alternative
Y = 10

###Simulation Paramters
replications = 5
firms = 2
periods = 50
agents = 100
RiskyFract = 0.5

#See page 529 for descriptions of each reproduction mechanism
#  1 = Uniformly random
#  2 = proportional to number of surviving firms
#  3 = proportional to total performance of surviving firms
#  4 = proportional to average performance of surviving firms
ReproductionMechanism = 2

#Set to FALSE to match the paper
#Do the surviving firms keep their previous performance round-by-round?
CummulativePerformance = FALSE

#Display full iteration details - adds time, only use for small replications
fulldetail = 0

#Create arrays to range over for later plotting
W = seq(from = .05, to = 0.95, by = ((0.95-0.05)/18))
H = c(0.3,0.5,1)


#t = time period (note, the paper is indexed relative to t=0, but here it's
relative to t=1)
#w = fraction of the population eliminated each time period
#h = affects the sensitivity of reproduction to the aggregate performance of
a type (see page 529)

#Create a data frame to keep track of the performance of each firm
PerfList = data.frame(matrix(NA, ncol =4, nrow = agents), stringsAsFactors =
FALSE)
colnames(PerfList) = c("Agent","Type","Performance","Rank")
```

```
#Create an array to keep track of the fraction of X (or 1) type firms at the
end of each run
FractX = array(data=NA, dim=c(length(W),length(H), replications))

t=1
n = 1
w = 0.3
h = 0.3
sens = 1
elim = 1

starttime = proc.time()

for (rep in 1:replications) {

  if (fulldetail != 1) {
    cat("\014")
    print(paste0("Replication ",rep," of ", replications))
    print(paste0("Elapsed time since last replication: ",(proc.time() -
starttime)[3]))
    print(paste0("Avg Replication time: ",(proc.time() - starttime)[3]/rep,"
per replication"))
    print(paste0("Est Time Remaining: ",((proc.time() -
starttime)[3]/rep)*(replications-rep)))


  }

  for (sens in 1: length(H)) {

    h = H[sens]

    for (elim in 1:length(W)) { #step through each elimination percentage w

      w = W[elim]

      #Initialize the data matrix for firm performance of each firm type
      PerfList$Agent = seq.int(1,agents)
      PerfList$Performance = 0
      PerfList$Type = append(rep(1,round(RiskyFract*agents)),rep(2,agents-
round(RiskyFract*agents)))

      #Get the number of frims to elminate each round based on the value of w
      kills = round(w*nrow(PerfList))

      for (t in 1:(periods+1)) {

        if (fulldetail == 1) {
          cat("\014")
          print(paste0("Replication ",rep," of ", replications))
          print(paste0("h value: ", h))
          print(paste0("w value: ", w))
          print(paste0("Time Period: ", (t-1), " of ", periods))
        }

        for (n in 1:agents) {
```

```
          #Determine the firm performance based on its type


          if (CummulativePerformance == TRUE) {
            CummPerf = 1
          } else{
            CummPerf = 0
          }

          if (PerfList$Type[n] == 1) {
            PerfList$Performance[n] = (CummPerf*PerfList$Performance[n]) +
rnorm(1, mean = X, sd = S)
          }

          if (PerfList$Type[n] == 2) {
            PerfList$Performance[n] = (CummPerf*PerfList$Performance[n]) + Y
          }
        } #next agent n

        #Order based on performance from worst to best
        PerfList$Rank = rank(PerfList$"Performance",ties.method = "random")
        PerfList = PerfList[order(PerfList$"Rank"),]

        SurviveList = PerfList[(kills+1):agents,]

        #Determine the various performance factors for the survivors that
affect reproduction

        T1 = sum((SurviveList$Type == 1)*SurviveList$Performance)
        T2 = sum((SurviveList$Type == 2)*SurviveList$Performance)

        N1 = sum((SurviveList$Type == 1))
        N2 = sum((SurviveList$Type == 2))

        A1 = T1/N1
        A2 = T2/N2

        #Define reproduction probability based on user choices

        if (ReproductionMechanism == 1) { #uniformly random replacement
          r1 = 1/firms
        } else {  #replacements proportional to the amount/performance of
firms

          #Avoid erroneous rates when one population is totally eliminated
          if (N1 == 0) {
            r1 = 0
          } else if (N2 == 0) {
            r1 = 1
          } else {

            if (ReproductionMechanism == 2) {
              r1 = N1^h/(N1^h+N2^h)
            } else if (ReproductionMechanism ==3) {
              r1 = T1^h/(T1^h+T2^h)
            } else if (ReproductionMechanism ==4) {
              r1 = A1^h/(A1^h+A2^h)
```

```
          }
        }
      }

      #Generate new firms based on the reproduction probabilities
      NewTypes = sample(c(1:2),kills,replace=TRUE,prob=c(r1,1-r1))

      #Record the new firm types
      PerfList[1:kills,]$Type = NewTypes
      #Reset the new firms performance to 0
      PerfList[1:kills,]$Performance = 0

    } # Next time period t

    FractX[elim,sens,rep] = sum(PerfList$Type == 1)/agents

  } # next elimination percentage w

 } # next sensitivity factor h

} # next replication

endtime = proc.time()

ElaspedTime = endtime-starttime
ElaspedTime

#Average each replication and determine the standard deviation
FractX_avg = apply(FractX, c(1,2), mean)
FractX_sd = apply(FractX, c(1,2), sd)

rep=1
matplot(W,FractX[,,rep], typ = "l", pch=4, col = "black", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Percent Eliminated Each Round (w)", ylab = "Fraction of Risk-
Choosing Agents at Round 50"
)

matplot(W,FractX_avg, typ = "l", pch=4, col = "black", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Percent Eliminated Each Round (w)", ylab = "Fraction of Risk-
Choosing Agents at Round 50"
)

##Plot of Figure 3 on page 530
plot(W,FractX_avg[,1], typ = "l", col = "black", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Percent Eliminated Each Round (w)", ylab = "Fraction of Risk-
Choosing Agents at Round 50")
    points(W,FractX_avg[,1], typ = "p", pch=23, lwd=1, bg = "white")
    lines(W,FractX_avg[,2], type = "l", col = "black", lwd=1, lty = 3)
    points(W,FractX_avg[,2], typ = "p", pch=8, lwd=1.5)
    lines(W,FractX_avg[,3], type = "l", col = "black", lwd=1, lty = 3)
    points(W,FractX_avg[,3], typ = "p", pch=21, lwd=1.5, bg = "white")
    #text(c(0.25,0.38,0.5),c(0.28,0.13,0.08),paste("h = ", H), cex = .75)
```

```
#Model Replication
#Adaptation as Information Restriction
#Jerker Denrell, James G. March, 2001

#Created by James Paine
#15.879 - Simulation Models in Social and Behavioral Sciences
#February 22, 2019

#####PART 4 - COMPETITIVE SURVIVAL WITH COMPETENCY BUILDING#####

###System Parameters

#Expectation of the 'risky' or 'new' alternative
X = 15
#Standard deviation of the 'risky' or 'new' alternative
S = 5

#Expectation of the 'certain' alternative
Y = 10

###Simulation Paramters
replications = 5
firms = 2
periods = 50
agents = 100
RiskyFract = 0.5
h = 0.5
k = 0.5
c0 = 0.3

#See page 529 for descriptions of each reproduction mechanism
#  1 = Uniformly random
#  2 = proportional to number of surviving firms
#  3 = proportional to total performance of surviving firms
#  4 = proportional to average performance of surviving firms
ReproductionMechanism = 2

#Set to FALSE to match the paper
#Do the surviving firms keep their previous performance round-by-round?
CummulativePerformance = TRUE

LearningTransfer = FALSE

#Display full iteration details - adds time, only use for small replications
fulldetail = 0

#Create arrays to range over for later plotting
W = seq(from = .05, to = 0.95, by = ((0.95-0.05)/18))
#D = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0)
D = c(0.1,0.4,0.6)

#t = time period (note, the paper is indexed relative to t=0, but here it's
relative to t=1)
#w = fraction of the population eliminated each time period
#h = affects the sensitivity of reproduction to the aggregate performance of
a type (see page 529)
```

```
#Create a data frame to keep track of the performance of each firm
PerfList = data.frame(matrix(NA, ncol =3, nrow = agents), stringsAsFactors =
FALSE)
colnames(PerfList) = c("Type","Performance","c")

#Create an array to keep track of the fraction of X (or 1) type firms at the
end of each run
FractX = array(data=NA, dim=c(length(W),length(D), replications))

starttime = proc.time()

for (rplct in 1:replications) {

  if (fulldetail != 1) {
    cat("\014")
    print(paste0("Replication ",rplct," of ", replications))
    print(paste0("Elapsed time since last replication: ",(proc.time() -
starttime)[3],"s"))
    print(paste0("Avg Replication time: ",(proc.time() -
starttime)[3]/rplct,"s per replication"))
    print(paste0("Est Time Remaining: ",((proc.time() -
starttime)[3]/rplct)*(replications-rplct),"s"))


  }


  for (LearnParm in 1:length(D)) {

    d = D[LearnParm]

    for (elim in 1:length(W)) { #step through each elimination percentage w

      w = W[elim]

      #Initialize the data matrix for firm performance of each firm type
      PerfList$Performance = 0
      PerfList$Type = append(rep(1,round(RiskyFract*agents)),rep(2,agents-
round(RiskyFract*agents)))

      #Set the initial competency to c0 before stepping forward through time
periods
      PerfList$c = append(rep(c0,round(RiskyFract*agents)),rep(NA,agents-
round(RiskyFract*agents)))

      #Get the number of frims to elminate each round based on the value of w
      kills = round(w*nrow(PerfList))

      for (t in 1:(periods)) {

        if (fulldetail == 1) {
          cat("\014")
          print(paste0("Replication ",rplct," of ", replications))
          print(paste0("d value: ", d))
          print(paste0("w value: ", w))
          print(paste0("Time Period: ", (t-1), " of ", periods))
        }
```

```
        for (n in 1:agents) {

          if (CummulativePerformance == TRUE) {
            CummPerf = 1
          } else{
            CummPerf = 0
          }


          #Determine the firm performance based on its type

          if (PerfList$Type[n] == 1) {
            #Get the st dev based on the current competency c
            stdev = (S/PerfList$c[n])^k
            avg = PerfList$c[n]*X
            #store the performance for this agent
            PerfList$Performance[n] = (CummPerf*PerfList$Performance[n]) +
rnorm(1, mean = avg, sd = stdev)

            #Update the agent's competency with the risky process for the use
in the next round (if they survive)
            PerfList$c[n] = PerfList$c[n] + d*(1-PerfList$c[n])

          }

          if (PerfList$Type[n] == 2) {

            #For these agents, the performance is a constant
            PerfList$Performance[n] = (CummPerf*PerfList$Performance[n]) + Y

          }

        } #next agent n

        #Order, from worst to best, based on performance from worst to best
        #PerfList$Rank = rank(PerfList$"Performance",ties.method = "random")
        #PerfList = PerfList[order(PerfList$"Rank"),]
        PerfList = PerfList[order(PerfList$"Performance"),]

        #Get list of suriving firms (bottom of ordered list)
        SurviveList = PerfList[(kills+1):agents,]

        #Determine the various performance factors for the survivors that
affect reproduction

        T1 = sum((SurviveList$Type == 1)*SurviveList$Performance)
        T2 = sum((SurviveList$Type == 2)*SurviveList$Performance)

        N1 = sum((SurviveList$Type == 1))
        N2 = sum((SurviveList$Type == 2))

        A1 = T1/N1
        A2 = T2/N2

        #Define reproduction probability based on user choices
```

```
        if (ReproductionMechanism == 1) { #uniformly random replacement
          r1 = 1/firms
        } else {  #replacements proportional to the ammount/performance of
firms

          #Avoid erroneous rates when one population is totally eliminated
          if (N1 == 0) {
            r1 = 0
          } else if (N2 == 0) {
            r1 = 1
          } else {

            if (ReproductionMechanism == 2) {
              r1 = N1^h/(N1^h+N2^h)
            } else if (ReproductionMechanism ==3) {
              r1 = T1^h/(T1^h+T2^h)
            } else if (ReproductionMechanism ==4) {
              r1 = A1^h/(A1^h+A2^h)
            }

          }
        }

        #Generate new firms based on the reproduction probabilities
        NewTypes = sample(c(1:2),kills,replace=TRUE,prob=c(r1,1-r1))

        #Record the new firm types
        PerfList[1:kills,]$Type = NewTypes
        #Reset the new firms performance to NA
        PerfList[1:kills,]$Performance = 0

        #Reset new firms of type 1 to the baseline compentency with the
process

        AvgC = mean(SurviveList[SurviveList$Type==1,]$c)

        if (LearningTransfer == TRUE) {
          NewC = AvgC
        } else {
          NewC = c0
        }

        NewComp = replace(replace(NewTypes, NewTypes==2, NA),NewTypes==1,
NewC)
        PerfList[1:kills,]$c = NewComp

        sum(PerfList$Type == 1)/agents

      } # Next time period t

      FractX[elim,LearnParm,rplct] = sum(PerfList$Type == 1)/agents


    } # next elimination percentage w

  } # next learning speed paramter d
```

```
} # next replication

endtime = proc.time()

(ElaspedTime = endtime-starttime)

#Average each replication and determine the standard deviation
FractX_avg = apply(FractX, c(1,2), mean)
FractX_sd = apply(FractX, c(1,2), sd)

##Plot of Figure 4 on page 530


matplot(W,FractX_avg[,], typ = "l", col = "black", lwd=1, lty =1,
     xlim=c(0,1), ylim=c(0,1),
     xlab = "Percent Eliminated Each Round (w)", ylab = "Fraction of Risk-
Choosing Agents at Round 50")

##Figure 4

plot(W,FractX_avg[,1], typ = "l", col = "black", lwd=1, lty =1,
        xlim=c(0,1), ylim=c(0,1),
        xlab = "Percent Eliminated Each Round (w)", ylab = "Fraction of Risk-
Choosing Agents at Round 50")
    lines(W,FractX_avg[,2], type = "l", col = "black", lwd=1, lty = 1)
    lines(W,FractX_avg[,3], type = "l", col = "black", lwd=1, lty = 3)
    lines(W,FractX_avg[,4], type = "l", col = "black", lwd=1, lty = 4)
    lines(W,FractX_avg[,5], type = "l", col = "black", lwd=1, lty = 6)
    lines(W,FractX_avg[,6], type = "l", col = "black", lwd=1, lty = 1)
    lines(W,FractX_avg[,7], type = "l", col = "black", lwd=1, lty = 6)
    lines(W,FractX_avg[,8], type = "l", col = "black", lwd=1, lty = 6)
    lines(W,FractX_avg[,9], type = "l", col = "black", lwd=1, lty = 6)
    lines(W,FractX_avg[,10], type = "l", col = "black", lwd=1, lty = 1)
    points(W,FractX_avg[,1], typ = "p", pch=23, lwd=1, bg = "white")
    points(W,FractX_avg[,2], typ = "p", pch=21, lwd=1.5, bg = "black")
    points(W,FractX_avg[,3], typ = "p", pch=8, lwd=1.5)
    points(W,FractX_avg[,4], typ = "p", pch=25, lwd=1.5, bg = "white")
    points(W,FractX_avg[,5], typ = "p", pch=10, lwd=1.5)
    points(W,FractX_avg[,6], typ = "p", pch=4, lwd=1.5)

text(c(0.9,0.8,0.65,.53,.4,.34,.22),c(0.05,0.1,0.15,.3,.4,.55,.8),paste("d =
", D[c(1:6,10)]), cex = .75)
```