An Evolutionary Approach to Improved Performance of Merged Companies

By

Raymond I. Ro

M.S., Electrical Engineering, 1995
North Carolina State University

B.S., Computer Engineering, 1993
B.S., Electrical Engineering, 1993
West Virginia University

Submitted to the System Design and Management Program in Partial Fulfillment of Requirements for the Degree of Masters of Science In Engineering and Business Management

AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, MAY 2001

© 2001 Raymond I. Ro, All Rights Reserved

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author _____

Raymond I. Ro
System Design and Management Program

Certified by _____

James H. Hines
Senior Lecturer, Sloan School of Management
Thesis Supervisor

Accepted by _____

Stephen C. Graves
LFM/SDM Co-Director
Abraham Siegel Professor of Management

Accepted by _____

Paul A. Lagace
LFM/SDM Co-Director
Professor of Aeronautics & Astronautics and Engineering Systems

An Evolutionary Approach to Improved Performance of Merged Companies

By

Raymond I. Ro

Submitted to the System Design and Management Program in Partial Fulfillment of Requirements for the Degree of Masters of Science In Engineering and Business Management

1 Abstract

Mergers and acquisitions (M&As) have become a key strategy for many companies. Possible reasons for M&As include entrance into a new market, growth, intellectual capital, and the elimination of competitors. However, over 80% of mergers have failed to create value for the shareholders and the merging companies. One reason for this failure might be in the integration of the companies. A current method is the use of a transition team to carry out the integration issues. These teams look at issues such as restructuring the company or creating best of practice policies. Sometimes consulting firms are used to develop strategies for the merger. They may focus on different strategies, such as generating company synergies. Our belief is that the lack of value created may be due to the lack of knowledge transfer. By improving this transfer of knowledge, individuals within the company can operate more effectively. These improvements in the overall performance of the organization should bring more value to the company and shareholders. Here, we suggest the use of an evolutionary approach, as suggested by Hines and House, where members from both organizations are mixed together onto teams to work on projects. We believe that by repeatedly mixing individuals from both organizations to work on team projects, knowledge transfer will improve and the company will evolve to a better state. However, since it may not be possible to fully mix two companies together in this manner, it is suggested that the use of a small number of mixed teams will be just as effective as fully mixing two companies.

Advisor: James H. Hines
Senior Lecturer, Sloan School of Management

2 Acknowledgements

I am grateful to the many individuals that have contributed to this thesis.

To begin with, I wish to express my sincere thanks to my advisor, Professor Jim Hines. His guidance, inspiration, wonderful insights, support, and unique sense of humor have made the creation of this thesis an extremely fun and educational experience. In spite of his very busy schedule, he has always managed to make time to help me out with any problems.

I also wish to thank Professor Jody House from the Oregon Graduate Institute. Her help with developing the simulator was invaluable, and her participation, encouragement, and guidance on the project were also essential for the completion of this thesis.

Also, many thanks to the other members of the organizational evolution team as well as my SDM colleagues. Finally, a special thanks to my family for their encouragement through out this program.

3 Table of Contents

4 List of Figures and Tables

List of Tables

“There is no law of progress. Our future is in our own hands, to make or to mar. It will be an uphill fight to the end, and would we have it otherwise? Let no one suppose that evolution will ever exempt us from struggles. 'You forget,' said the Devil, with a chuckle, 'that I have been evolving too.'”

- William Ralph Inge

"A complex system that works is invariably found to have evolved from a simple system that worked."

- John Gall

5 Introduction

5.1 Motivation

Mergers and acquisitions (M&A) have recently become a growing trend among companies. Over 26,000 mergers occurred worldwide in 1998, which was estimated to be worth $2.4 trillion dollars. Compare this to the 11,300 mergers that had occurred in 1990, which was worth around a half trillion dollars ("After the Deal"). Not only has the number of mergers increased, the size of the mergers have grown also. As of 1998, eight of the ten largest mergers had occurred in 1998 (Spitzer 4). In 1999, Cisco had performed 18 acquisitions valued at $14.5 Billion (Barker 138). So why are companies so eager to perform these mergers? This can be for several reasons, including the acquisition of new technologies, improving their competitive advantage, growth, or other factors. In the end, companies expect to gain added value through an M&A. However, in the last 10 years, 83% of M&A deals have never created the value that the companies or shareholders had expected (Kelly 7). This is astonishing, considering the number of mergers that have taken place and the size of some mergers.

One might think that this high failure is a result of poor evaluation strategy; however, this may not be so. According to David Jemison, a professor at the University of Texas, "…organizations whose transactions have failed have typically done as well with evaluating strategy and crunching numbers as those whose have succeeded" (Spitzer 9). Another possible reason for failure may be due to how the companies are integrated. Edgar Schein believes that mergers are difficult for a cultural reason:

> Ideally we look for blending, by which the new organization takes the most functional elements of each culture. But this presumes ability to decipher the cultures, which is rarely present until long after the new organization has been formed (184).

Some companies attempt to capture these "functional elements" by having members from both companies work together on a team, often known as a transition team. These transition teams are usually tasked to reorganize the company and to decide upon best of practice policies. Policies are implicit (sometimes explicit) rules that individuals use to make decisions (Hines and House, "Harnessing" 3). Unfortunately, the task of the transition team can be difficult since they may fall into a cultural trap of falsely understanding the other company (Schein 179) or they may lack "maximum insight into their own culture" (Schein 184). In the end, it is difficult for any one individual or small group of individuals to understand all of these complexities in order to create effective policies for the company.

Another reason for the lack of value gained by mergers may be attributed to ineffective knowledge transfer of policies between individuals of the merged companies. If we are able to create an effective and simple method to improve knowledge transfer of policies between two merging companies, we can then perhaps improve the effectiveness of how these merged companies operate, thus, increasing the value of the merger. To explore this possibility, an organizational evolutionary approach was taken, as suggested by Hines and House's paper on Organizational Management ("Harnessing" 3).

Specifically, Hines suggests “five rules for evolutionary management.” One possible rule is to mix individuals together in a company so that knowledge can be transferred (Hines, “Five Rules” 4). For mergers, this would imply that all of the individuals at one company would need to relocate to the other company, or that half of the individuals from each company would need to switch. This could be very expensive to implement and may not be practical to do. What we suggest is to mix a small number of individuals from both companies onto a few project teams, known as mixed teams. We expect that these mixed teams will be just as effective in transferring knowledge as fully mixing two companies.

5.2 Goals

The work performed in this thesis attempts to apply organizational evolutionary ideas to improve the performance of mergers and acquisitions. Specifically, a small mixed team structure is proposed as an effective way to transfer knowledge across organizations. To test this proposed idea, a simulator was developed, based on an existing simulator. Insight gained from simulation will be used to recommend practical policies for applying the evolutionary method to an M&A.

The thesis output will consist of:

- An explanation of the mixed team structure.
- Overview of the simulation structure.
- Simulation code written in Matlab.
- Analysis of simulation data.
- Recommendations for applying the mixed team structure for M&As.
- Recommendations for further research and study.

5.3 Assumptions and Scope

The scope of this work is the application of organizational evolution to improve the knowledge transfer of good policies in an M&A. When individuals copy part or all of a policy from other individuals, it is known as learning. Team policies are affected by the different individual policies that each member has. Depending how a team performs relative to other teams within the company, individuals on that team will be promoted or demoted. Promotion and learning will be studied as methods to improve knowledge transfer. The process when an individual creates a new policy or comes up with a new idea is known as innovation. This will not be studied.

5.4 Work Process

The author used a practical problem and research problem appraoch (Booth, Colomb, and Williams 49). This process is shown in Figure 5-1. The practical problem of improving the effectiveness of a merger using an organizational evolutionary approach was chosen as the topic of this study. A mixed team structure was proposed as a solution to this practical problem. It is believed that this mixed team structure, compared to fully mixing two organizations, would be just as effective in improving knowledge transfer but easier to implement. To see if this was feasible, an understanding of how the knowledge transfer of policies within an organization was required. This motivated us to develop a model of the mixed team structure and the fully mixed company structure. The effects of promoting and learning were studied using these models. Several experiments were performed to gather insights. These insights were then applied to the solution of the practical problem, the use of mixed teams.

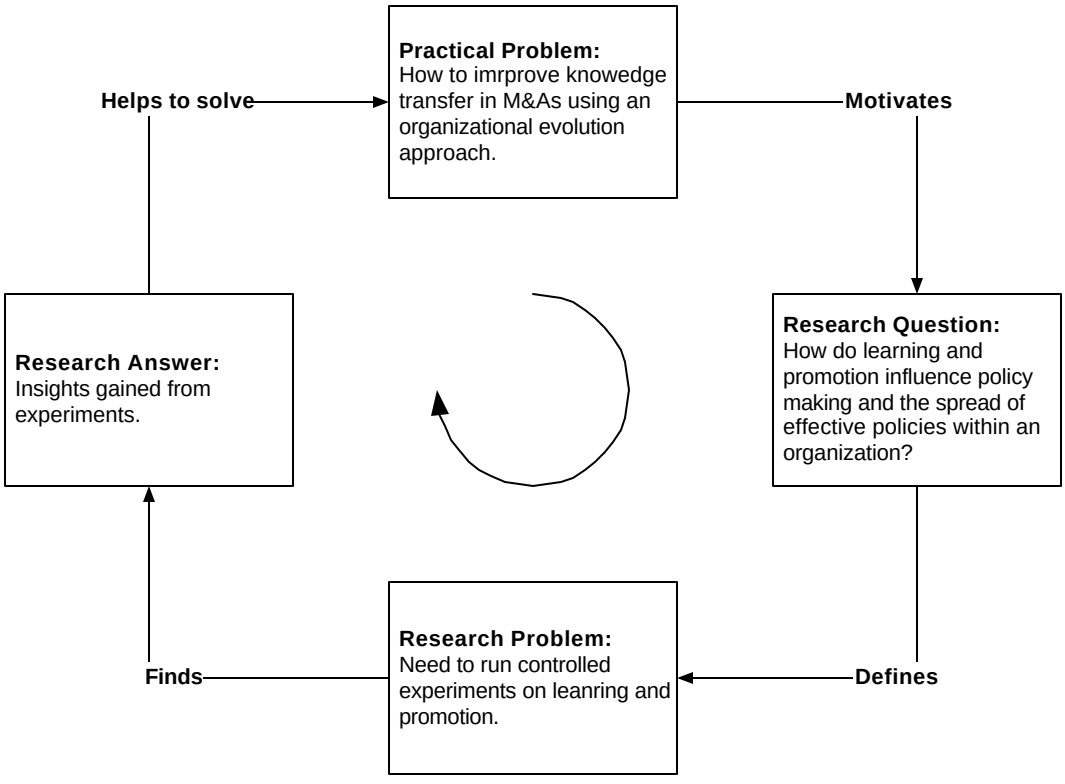

Figure 5-1. Work process for thesis includes a practical problem and a research problem.

6 Foundations

This chapter is intended to provide the reader with some basic background knowledge of how the simulator works and of other needed concepts. An introduction to genetic algorithms is given, where basic concepts and terminology will be discussed. Next, the structure of the simulator will be discussed, followed by the data types used in the simulator. Then team policymaking, promotion, and learning will be explained. A brief overview of the central limit theorem will be given. Finally a brief discussion of the use of the system dynamics project model will be provided.

6.1 Genetic Algorithms

In essence, "Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics" (Goldberg 1). Genetic Algorithms (GAs) were derived from the work of computer scientists that studied evolutionary systems in the 1950s and 1960s. Their motivation for studying evolutionary systems was to find an optimization tool for engineering problems. The basic characteristic of these evolutionary systems was to evolve a population of solutions toward an optimal value (Mitchell 2). In the 1960s, John Holland and his students at the University of Michigan developed the genetic algorithm. Two major goals of their research included:

1. "To abstract and rigorously explain the adaptive process of natural systems."
2. "To design artificial systems software that retains the important mechanisms of natural systems" (Goldberg 1).

As suggested by Melanie Mitchell, the evolutionary approach used by GAs is attractive for several reasons (4). First, the evolutionary approach is robust in nature. Given a large solution space, GAs are able to effectively search multiple solution points in that space in parallel. Unlike other optimization methods, GAs have a low probability of getting stuck at a local maximum or minimum solution. Second, the evolutionary approach allows for the design of innovative solutions to problems. Due to its random nature, GAs are able to create new solutions by producing offspring or by mutation. Third, the evolutionary approach is adaptive and will perform in a changing environment. If the fitness function changes, the population of solutions will evolve with the fitness function. Finally, the evolutionary rules are simple to implement and program, even for complex problems. The fittest solutions within the population survive. At the same time, new solutions are randomly being created within the population, with the possibility that they will be more fit.

The mechanics behind how a GA works is based on biology. Basic elements of a GA include "chromosomes," population, and a fitness function. Chromosomes contain a particular solution to an optimization problem. This solution can be a set of parameters or a point in the solution space and is usually encoded as a binary string (Goldberg 21). A set of chromosomes makes up a population. This population represents many solutions within a solution space. Each chromosome within the population is

evaluated by the fitness function to determine its fitness value. The fitness function is a measure of how fit an individual chromosome is. The idea is to maximize this fitness value. A more fit chromosome represents a solution that performs better relative to some objective function. A less fit chromosome represents a solution that performs worse relative to that objective function.

To evolve the population, basic evolutionary functions are carried out: selection, crossover, and mutation. Selection is the process where chromosomes are randomly selected for reproduction. Like biological evolution, the more fit individuals survive to produce offspring. In the GA, individuals with higher fitness values will more likely be chosen for reproduction. Once the individuals have been selected for reproduction, known as parents, the offspring is created. In reproduction, parents (pairs of individuals) transfer segments of their chromosome with each other to create two new children. This is known as a crossover. Finally, a few bits within the offspring chromosomes are flipped randomly to represent mutations. Mutations occur with a very low probability and represent a jump to a new point in the solution space. When selection, crossover, and mutation has occurred for the population, a generation has occurred. GAs will run for several generations, while the population evolves towards a more optimal state.

6.2 Simulation Structure

New code was developed and added to the original Organizational Evolution simulator to allow for simulation of merging companies. Matlab was used as the programming language, due to its flexibility and its abundance of mathematical functions. This newer code differs from the original code in that the data structures have been designed to allow for simulation of two legacy companies merging together, along with the use of mixed teams. To help improve the simulation speed, vectors are used as much as possible. Furthermore, more control is given to the user to turn on and off the learning and promotion functions at a specified number of generations. This allows one to clearly see the before and after effects of learning and promoting.

Construction of this simulator consists of an agent based modified genetic algorithm, wrapped around a system dynamics project model. The modified genetic algorithm was written in Matlab. Differences exist between the modified genetic algorithm and the standard genetic algorithm. First, the population is divided into teams, where individuals learn from other team members. Second, mutations were not implemented to simplify the analysis. Third, the fitness function evaluates the team policies, not the individual policies. Fourth, individuals are never replaced within the population. Finally, a simplified system dynamics project model was used as a fitness function. The project model was used to evaluate the team policies. More detail of this is given in the paper by House, Kain, and Hines, “Metaphor for Learning: An Evolutionary Algorithm.”

A high level flow diagram of the Matlab code is shown in Figure 6-1. This basic flow is the same for the single, fully merged company case or for the split companies with mixed team case. More detail about the company structure will be given later on. At the start of the simulation, various simulation parameters are loaded and used to create the data structures. Some of these parameters include, number of generations, team size, number of mixed teams, number of teams in the company, and probability of learning. Once the data structures for the company, teams, and mixed teams have been created, each individual is given an initial policy value. The simulator now begins the iterative part of the simulation. It first begins by checking to see if it is at the last generation or not. If so, the simulation ends, else the simulator continues onward. In the next step, the team policy is computed, based on a weighted average. Team policy computation will be discussed later on. Next, each team's policy is then evaluated. A simplified project model is used to evaluate how fit a policy is. This project model was created in Simulink, a Matlab block diagram simulator, and interfaces directly to the Matlab code. Matlab feeds the number of programmers into the project model. Simulink will then return the project completion time to Matlab. In the project model used, more programmers will complete a project faster. Thus, teams with policies of more programmers will perform better than teams with policies with fewer programmers. Teams are then ranked based on their project completion time. Next, the simulator checks to see if promotion for teams is enabled. If so, promotion values are computed for all teams and the members are then updated by this value. Individuals are then randomly assigned to new teams. Next, the simulator checks to see if learning is enabled. If so, individuals randomly select another team member to learn from. A "roulette wheel" is used to select another individual to learn from. However, an individual's status will determine how much of the roulette wheel an individual has, thus, changing their probability of being selected. An individual with high status will be given more of the wheel and improving their chances of selection. An individual with low status will be given a smaller portion of the wheel, thus making it harder for other individuals to select them for learning. This is analogous to real life, where individuals who want to do well tend to learn from people with more knowledge and experience than them. After learning has been performed, the generation is checked to see if the iterative loop is to be executed again.

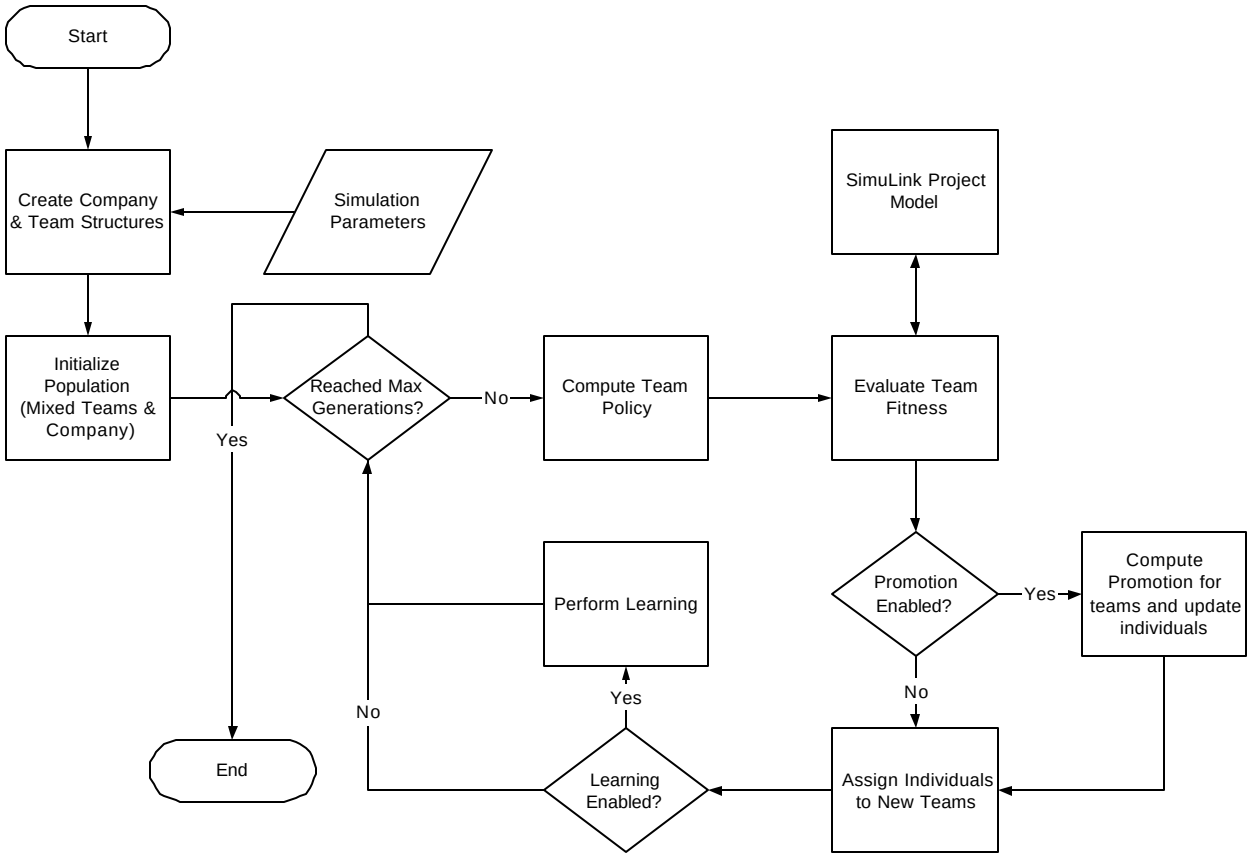

Figure 6-1. High-level flow of simulator.

6.3 Company, Team, and Individual Structures

To help organize the data in the simulator, three basic structures were created: the company structure, the team structure, and the individual structure. This allowed for the creation of a simulator that could operate as a fully merged company (single company mode) or a mixed team company.

The single company simulation mode operates as if learning and promoting occur within a single company. This is also identical to two merging companies completely immerse themselves into one single company. By doing this, we are allowing all members from both companies to mix and to work with each other on projects. This also creates a boundaryless organization, which allows for easy assignment of members from both companies onto a team. Ultimately, this looks just like a single company as shown in Figure 6-2. In this simulation mode, a company structure with *m* teams will be setup according to the specified simulation parameters. Each team will have the same number of individuals, which is also set by the simulation parameters. The team size and number of individuals per team will remain fixed throughout the simulation, but the specific individuals on a team will change each generation.

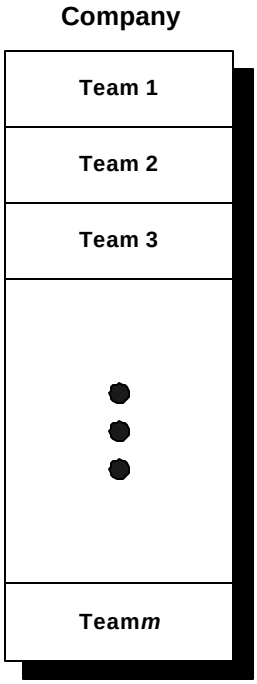

Figure 6-2. Fully merged or single company structure with *m* number of teams.

In contrast to the single company case, a different situation was created in which a small number of project teams contained a mix of individuals from both companies. Instead of diffusing knowledge by mixing all of the individuals from both companies, knowledge was spread between companies through the use of these mixed teams. In the mixed team simulation mode, the simulator generates two company structures and a mixed team structure, as shown in Figure 6-3. This mode of simulation simulates the case when two merging companies remain separate; however, each company assigns a small number of individuals to work on projects with members of the opposite company. Working on these mixed teams will hopefully be an effective way for individuals from both companies to learn from each other. These mixed teams, along with the teams in the legacy companies, will be rated on how well their project performs. This rating translates to a promotional value. Each team's promotional value will then be applied to all the team members. After the completion of a project, individuals who had been on mixed teams may rotate back to their original companies, hence, taking and spreading their new knowledge. For the simulator, the number of individuals per team, the number of mixed teams, and the number of teams per company are specified. Both companies have the same number of teams. These parameters remain constant during the simulation. Furthermore, individuals are not allowed to switch companies, i.e., an individual from company A cannot be assigned to a team in company B. However, an individual from company A can be assigned to a mixed team, which has individuals from company B. Mixed teams are represented by an equal number of members from both companies. Thus, the mixed team acts as a

conduit for transferring knowledge or policies between both companies. Once an individual returns to their home team with their newly learned policy, that information can then be transferred to other members of that company via learning.

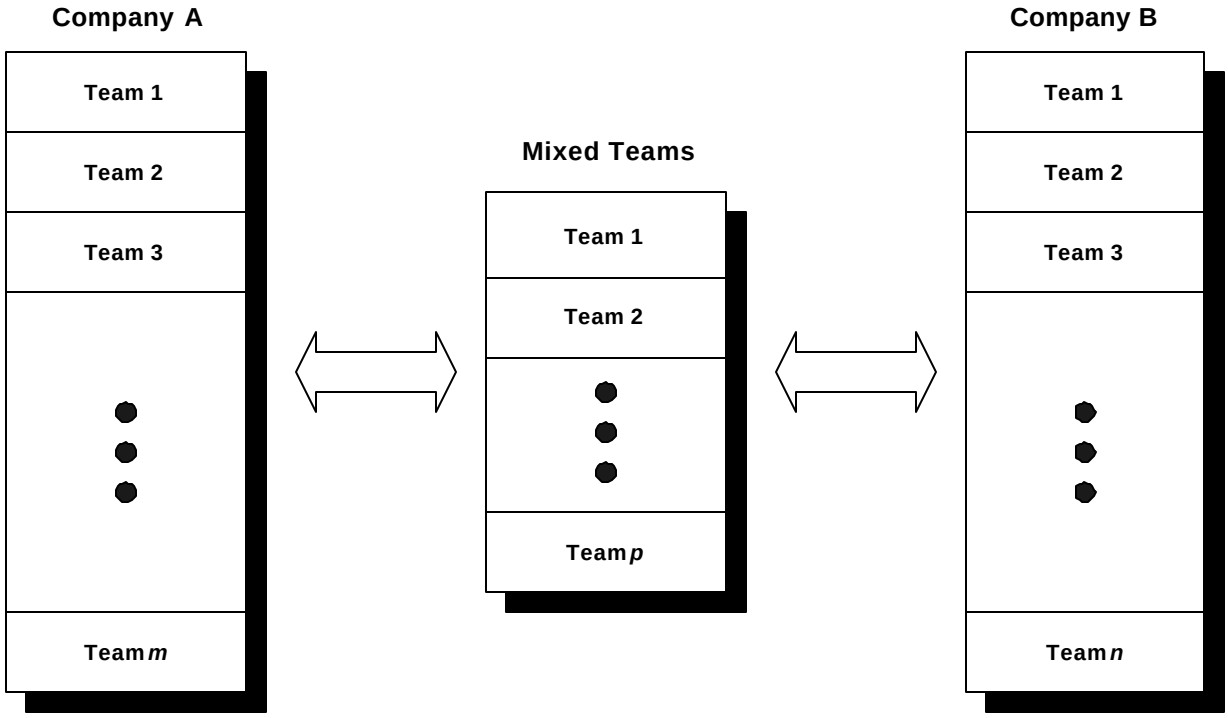

Figure 6-3. Mixed team structure with two legacy corporations. Company A has *m* teams, Company B has *n* teams, and there are *p* mixed teams.

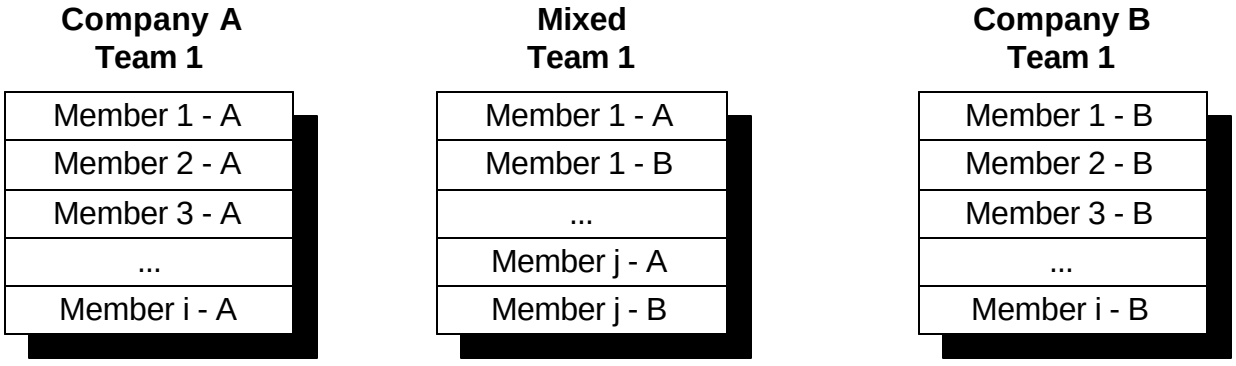

Figure 6-4. Team structure for both corporations and the mixed team.

Figure 6-4 illustrates a more detailed view of the team structure. Teams from the legacy companies consist of members from that company only. On the other hand, mixed teams have an equal representation of members from both companies. As stated before, all team sizes are the same.

Information for each individual is tracked at each generation throughout the simulation. Figure 6-5 is a list of attributes kept on each individual. The chromosome, or individual's policy (knowledge) is tracked at every generation. For this simulation, this policy is the number of programmers to assign to a project. Each individual has a unique ID to track them by and a team ID to identify which team that individual is assigned to. The individual's status[1] is also kept track of throughout the simulation. A short history of whom an individual learns from is stored in the personal network attribute. For individuals on mixed teams, identification of which home company they came from is stored. Finally, team policies and team rankings are stored.

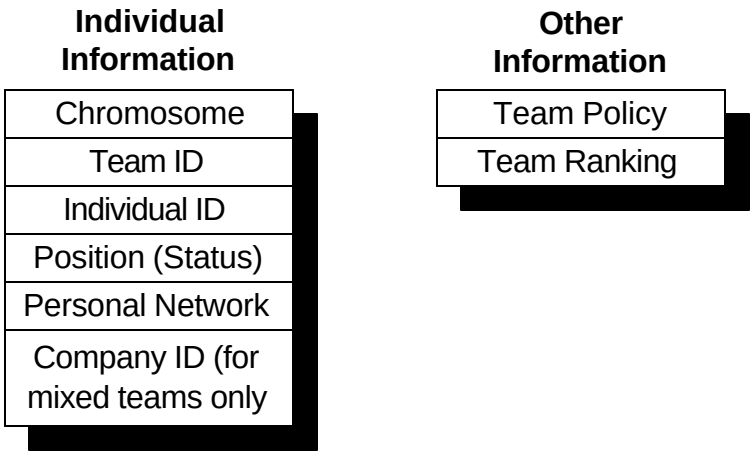

Figure 6-5. Attributes for each individual in the simulation.

6.4 Team Policy Making

Team policies are generated using a weighted average of all the team members as specified in House, Kain, and Hines (3). These weights are the individual statuses of all team members at the beginning of a project. Figure 6-6 shows a block diagram for how a team policy is created. A weighted policy value is created for each individual on the team by taking the weighting factor or status, W_i, and multiplied by his or her policy value, P_i. The weighted policy value is then added up for the team. At the same time, a sum of each individual's weighting factor is computed. The team policy value is the sum of the weighted policy value divided by the sum of the weighting factors. Thus, an individual with a high

[1] "Status" is a more general idea of what Hines and House refers to as "position." The author chooses to use status rather than position in order to minimize possible confusion with a title or a position on an organizational chart. This change in terminology does not change any underlying mathematics.

status value will have more influence on the team policy than an individual with a low status value. For example, lets say that an engineering team is composed of a principle engineer and a young engineer who is the early stages of their career. Though the young engineer may be well regarded by other engineers within the company, we would expect the principle engineer to have a much greater status than the young engineer. Thus, we would expect that the decisions and actions of this team to be heavily influenced by the principle engineer.

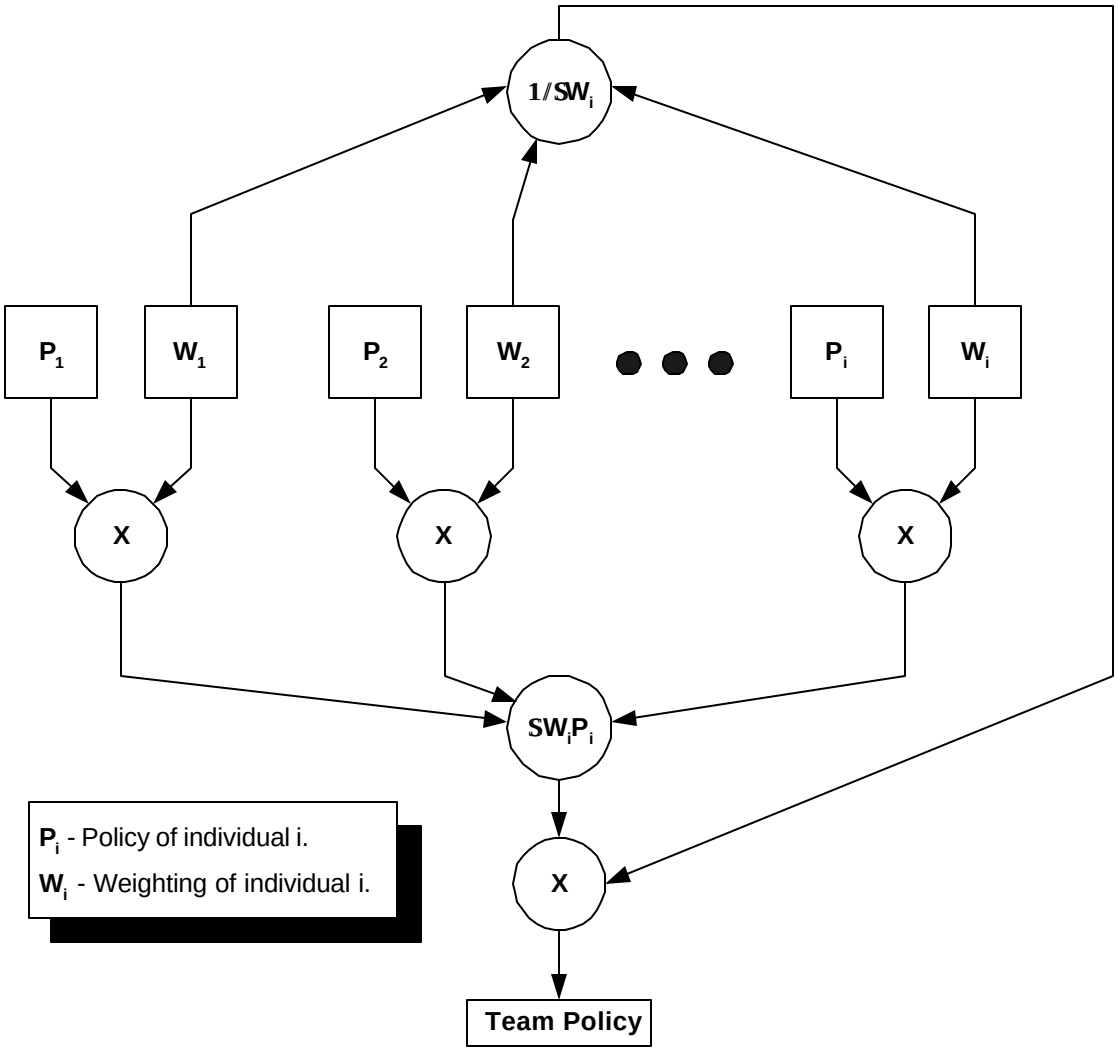

Figure 6-6. Block diagram for computing weighted team policy.

6.5 Promotion

When a team has good policies and performs well on a project, that team is looked favorably upon by other members within that company. Likewise, if a team that has poor policies and performs poorly on a project, other members of that company look down upon that team. This favorable or non-favorable impression on the team is transferred to the individuals on that team. If a team performs well, individuals on that team improve their status. If a team performs poorly, the individuals on that team lose some status. In the simulator, teams within the company are evaluated on how their policy effects the completion time of a project. The faster the completion time of the project, the better the team evaluation. Once all of the teams have completed the project, they are ranked based on the project completion time. The promotion scheme used in this simulator is the same as the one specified in the "Metaphor for learning: an evolutionary algorithm" (House, Kain, and Hines 3). This promotion uses a base 2 and is defined as:

$$Status_{new} = Status_{old} \times 2^{K}$$

Where K is defined as:

$$K = \frac{2 \times (Team\ Rank - 1)}{(Number\ Of\ Teams - 1)} - 1$$

The best performing team will double its status value, while the worst performing team will half its status value. Thus, all of the individuals on the best performing team will double their status value, while all the individuals on the worst performing team will have their status values cut in half.

6.6 Learning

Learning allows the genetic information from one individual to be transferred to another individual and is much like the crossover function in a standard genetic algorithm (House, "Metaphor" 2). In a standard crossover, two individuals exchange parts of their chromosome, which imitates a biological recombination between two single-chromosome organisms (Mitchell 3). In organizational evolution, policies are encoded into the individual's chromosome. Like crossover, learning transfers part of the genetic material; however, only the chromosome for the individual that is learning changes. This learning process is illustrated in Figure 6-7. In this example, team member 2 is selected to learn from team member i. Copies of chromosomes are made for members 2 and i. The simulator then generates a random crossover point; in this case, the crossover point is four bits into the chromosome. Next, a segment up to the transfer point of member i's chromosome is copied over member 2's chromosome. As

shown in the figure, "**1101**" from member i is copied to member 2, yielding a new policy of "**01001101**." This new chromosome is then copied back to member 2's chromosome attribute.

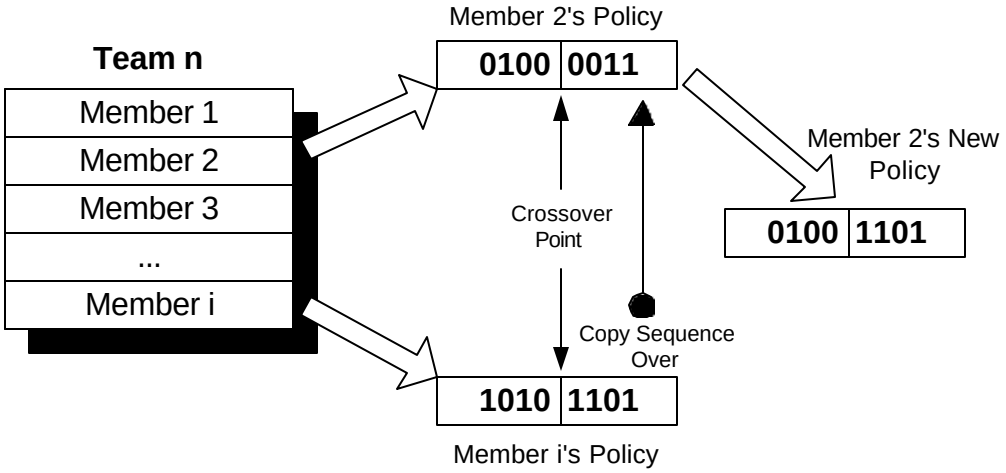

Figure 6-7. Member 2 learns from Member i.

Another very important part of the learning process is selecting whom an individual learns from. In the simulator, team members learn from another team member by a random selection. However, a team member's status will greatly influence if they are selected to be the source of learning by another memeber. At the beginning of the simulation, each individual on the team has equal status, thus, there is an equal probability for each member to be selected. As each project is completed, promotional values will be assigned to members of a team, which will change their status. As an individual achieves higher status, the probability that he or she will be selected for learning increases. Conversely, as an individual's status is reduced, the probability that that individual will be selected decreases. This probability of being selected is calculated by taking an individual's status and dividing it by the sum all the individual statuses on that team. Table 6-1 lists each individual's status and probability of being learned from for a particular team. Figure 6-8 represents the percent chance that a particular individual will be selected for learning.

Individual	Status (Weighting)	Probability of Being Selected
I-1	0.4244	0.09
I-2	0.8816	0.19
I-3	0.3137	0.07
I-4	2.0771	0.44
I-5	0.3742	0.08
I-6	0.6353	0.13

Table 6-1. Sample data for individuals on a team.

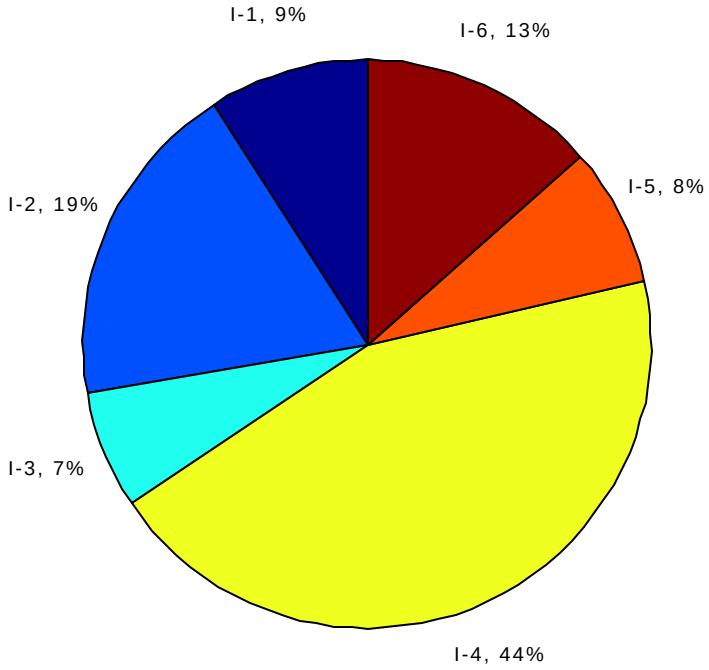

Figure 6-8. Pie chart showing the percent chance an individual will be selected to be learned from.

Suppose a team consists of a principle engineer and several new engineers. Recall that the principle engineer's status will be extremely high, while the status of the new engineer's will be quite low. Consequently, the probability that individuals on the team will select the principle engineer to learn from will be higher than the probability of selecting another new engineer. There is a small probability that the principle engineer might select a new engineer to learn from. This could improve or worsen the principle engineer's policy; however, in most cases, the principle engineer would select his or herself to learn from, in which case we do not change. Now suppose that this team had the highest ranking within the company. Then the status for each individual on a team would be doubled. When the team members are assigned to their next project and team, the probability of that individual being selected to be learned from increases. If however that team performed the worst of all teams within the company, then the status of each individual would be halved. When individuals are assigned to their next team, there is now less of a probability that they would be selected for learning. It would also be more probable for those individuals with low status to learn from other individuals with higher status on the new team.

6.7 Central Limit Theorem

Team policies are created using a weighted average. Since the statuses of all individuals are the same at the beginning of a simulation, an interesting phenomenon in the distribution of all team policies occurs. The central limit theorem is used to explain this phenomenon and will be applied later on.

Suppose we have a sequence of identically independently distributed (iid) random variables, X_1, X_2,…, that have a finite mean of μ and a finite variance of σ^2. Lets then sum up the first n random variables in the sequence, such that:

$$S_n = X_1 + X_2 + \ldots + X_n.$$

Then according to the central limit theorem, as n becomes large, the cumulative distribution function (cdf) of S_n approaches the cdf of a normal distribution (Leon-Garcia 280). As long as the X_i's are iid, then S_n has a mean of $n\mu$ and variance of $n\sigma^2$. If we are looking at averages, where

$$M_n = S_n/n,$$

then the mean is μ and variance is σ^2 (Walpole 210).

6.8 System Dynamics Project Model

The system dynamics project model is not the core of this study; however, we will briefly mention its use in the simulator. Figure 6-9 shows a simple project model created in Simulink, based on the project model described by Hines and House ("Source of Poor Policy" 9). This project model represents a software project. On this project, there are an assigned number of computer programmers that must complete a specified number of tasks in order to finish this project. The team of programmers has a productivity value, which allows them to complete each task at a certain rate. Unfortunately, programmers make mistakes that create bugs in the software code. The number of bugs is affected by the quality of work by the group. These bugs take time to discover and once they bugs found, they become rework or tasks that need to be corrected. The policy value of the teams in the simulator is to decide upon the number of programmers to assign to this software project. Valid policy values are anywhere from 11 to 20 programmers. In this simple policy, the more programmers assigned, the faster the project completes. Project completion time is used to evaluate the team policies. The shortest completion times represent the most fit group, while the longest completion time represents the least fit

group. Other parameters, such as quality, productivity, time to discover rework, and initial tasks remain constant for all projects.

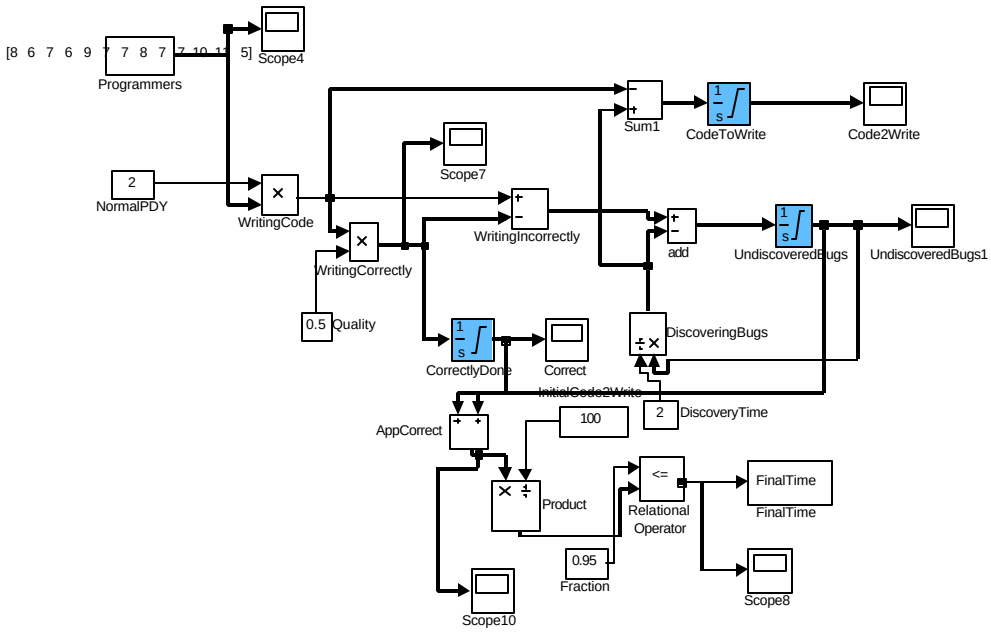

Figure 6-9. A simple project model represented in Simulink.

7 Experiments

To help gain further insight on the effectiveness of the mixed team structure, a series of experimental simulations were run. These simulations are divided into two major categories. In the first category, we will look at a single company situation, which is analogous to fully merging two companies together. The second category is the mixed team simulation, where a small number of individuals from both companies are mixed together on a team to work on a project. To help us gain more insight, each category is broken down into four situations. We look at how a company evolves with no learning and no promotion, with promotion only, with learning only, and with learning and promotion. The experiments are summarized in Table 7-1. We hope to understand the effects of learning and promotion in the mixed team structure and to see if it will evolve as well as a fully merged company.

The setup is consistent for all of the experiments. In the simulations, the teams must decide on the number of programmers to staff on a project. Each team policy will then be fed into its system dynamics project model. This is the basic project model as described earlier. Recall that the project has a faster completion time when there are more programmers assigned to the project. Teams are allowed to staff anywhere between 11 to 20 programmers. This range of programmers was constrained by the author and is not a limitation of the simulator. Thus, a team policy of 20 programmers would yield the fastest project completion time, while a team policy with the value of 11 would yield the slowest project completion time. Each individual's initial policy for staffing is generated randomly, using a uniform distribution over the valid range of 11 to 20 programmers. Recall how the individual's status will affect the weighting of the team policy. Initial status values for all individuals start off equal. After all of the teams have completed their projects, teams are assigned promotional values, and the individuals are then promoted and randomly assigned to a new project. This process is repeated again for the specified number of generations.

In both the single company and the mixed team company case, there are six members to a team. In the mixed team case, each company has 3 members per mixed team. Furthermore, there are a total of six mixed teams plus 25 additional teams per company (a total of 56 teams). In the single company case, there are 50 teams total. Finally simulations were run for a total of 100 generations.

To evaluate performance, it is convenient to have a method to determine if a company is evolving to a better state. One method of measuring this evolution is to compute the average of all the teams' policy values. If this value improves, say toward the optimal value, we can say that the organization as a whole is evolving to a better state. In our case, the average team policy value would be around 15 programmers. The optimal value is 20 programmers.

Finally, simulations of the single company case (fully mixing) and the mixed team case were studied. The single company case provided us with a baseline, in which the mixed team case was compared to.

Experiment	Main Section	Fully Merged Team	Mixed Team
No Learning and No Promotion	7.1	7.1.1	7.1.2
Promotion Only	7.2	7.2.1	7.2.2
Learning Only	7.3	7.3.1	7.3.2
Learning and Promotion	7.4	7.4.1	7.4.2
Good/Bad Policy	7.5	-	7.5

Table 7-1. A Summary of All Experiments.

7.1 No Learning and No Promotion

To better understand how promotion and learning effects the evolution of a merging company, it will be helpful to understand how a baseline case of no learning and no promotion behaves. Since no learning occurs, individuals do not change their policies. Since no promotion occurs, no one individual has more influence over the team policy than another individual, regardless if their policy is very good or very bad. Since a uniform distribution (11 to 20 programmers) is used to set the individual's polices, we would expect an average around 15 programmers to be staffed.

Randomly mixing good and poor performers on a project is somewhat realistic. One would hope that the individuals with poor policies would learn from individuals with good policies. Furthermore, it would be uncommon for a company to have all of its good people on one project and poor people on another project. This could cause some projects to be extremely successful, while the other projects fail. In an engineering firm, it's like putting all of the principle engineers on one project, and all of the new hires on another project.

7.1.1 Fully Merged, Single Company

When a company lacks methods for improving individual's policies, we would not expect the company to evolve to a better state, nor do we expect it to get worse. Suppose there exists a company that has just fully merged (single company case), if no learning occurs, nor do we give more weight to any one individual in the decision making process, there will be no improvement in team policies. Individuals on teams that perform well do not get promoted, even though they probably carry a good policy. On the

flip side, individuals with poor policies do not become demoted for their poor performance. Since the policies within the company are equally good as well as bad, we expect that on average, the teams will select 15 programmers on a project. This will lead to average project performance by the teams. Figure 7-1 shows the histogram for all individuals within a company, in generation 1 and generation 100.

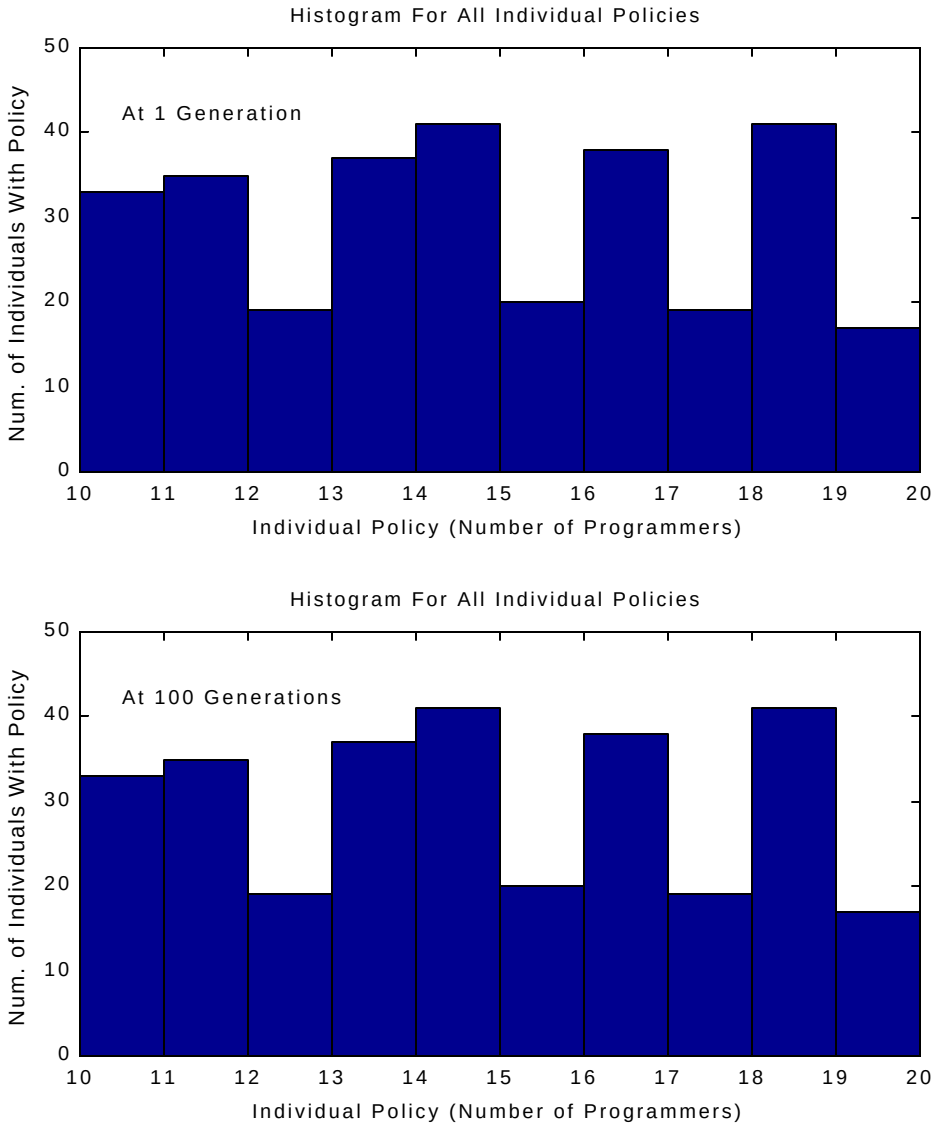

Figure 7-1. Histograms for individual policies show a uniform distribution for the first and last generation (*s_nlnp_r1p50u_4_11*).

Since no learning is going on, we expect that everybody's policy will remain the same from generation to generation. As we can see in Figure 7-1, the histogram for the first generation is identical to the histogram of the last generation.

In companies, projects tend to be run by teams, thus, a team will develop a policy based on each member's input. When a team policy is created, each member of that team has a certain amount of influence over the team policy. This influence is usually based on the individual's status. If an individual has performed well in the past by having good policies, they are usually rewarded and gain a higher status within a company. If an individual has a poor policy and has not performed well in the past, they will most likely be demoted and have a lower status within the company. Thus, if status is based on performance, than we expect individuals with good policies to be more influential over the team's policy than individuals with poor policies. However, since there is no promotion in this specific case, each individual within the company will have the same influence over their team's policy. Since no learning and promotion is being performed in this case and individuals are randomly assigned to teams, we would expect that half the members of a team to have a good policy and the other half to have poor policies. Individuals with policies that are between the good policy and bad policy have average policies. Because everyone has the same influence on the team policy, we would expect the team to create an "average" policy. Thus, this policy is just as good as it is bad. In fact, we would expect most of the teams within the company to produce a policy that is around the average. A few teams may produce team policies higher than the average, while some other teams may produce team policies below the average. If all of the projects were completed and the company policy was to randomly reassign individuals to new teams, we would expect roughly the same team policy results within the company. Since we would not expect the average policy for all the teams within the company to get better, we do not expect the company to evolve to a better state. That is, we do not expect the average of all the teams to improve.

If we look at Figure 7-2, which shows the distribution of teams from the simulation, we see that the histogram of the team policies within a company follows a normal distribution, with an average of about 15 programmers. Recall that the each individual can have a policy of anywhere from 11 to 20 programmers. Since every individual within the company has the same status, regardless of whether their policy is good or bad, the team policy is just an average of each person's policy. Because we are looking at the averages of several policies that were generated using a uniform distribution, we expect a normal distribution about the mean of the uniform distribution, due to the Central Limit Theorem. This reflects what we would expect the distribution of team policies to be, namely most team policies are average. As we expect, the histograms in Figure 7-2 have a normal distribution in generation 1 and generation 100. The average and variance for all team polices over 100 generations (projects) is shown in Figure 7-3. As expected, both the average and variance remain constant, thus no evolution.

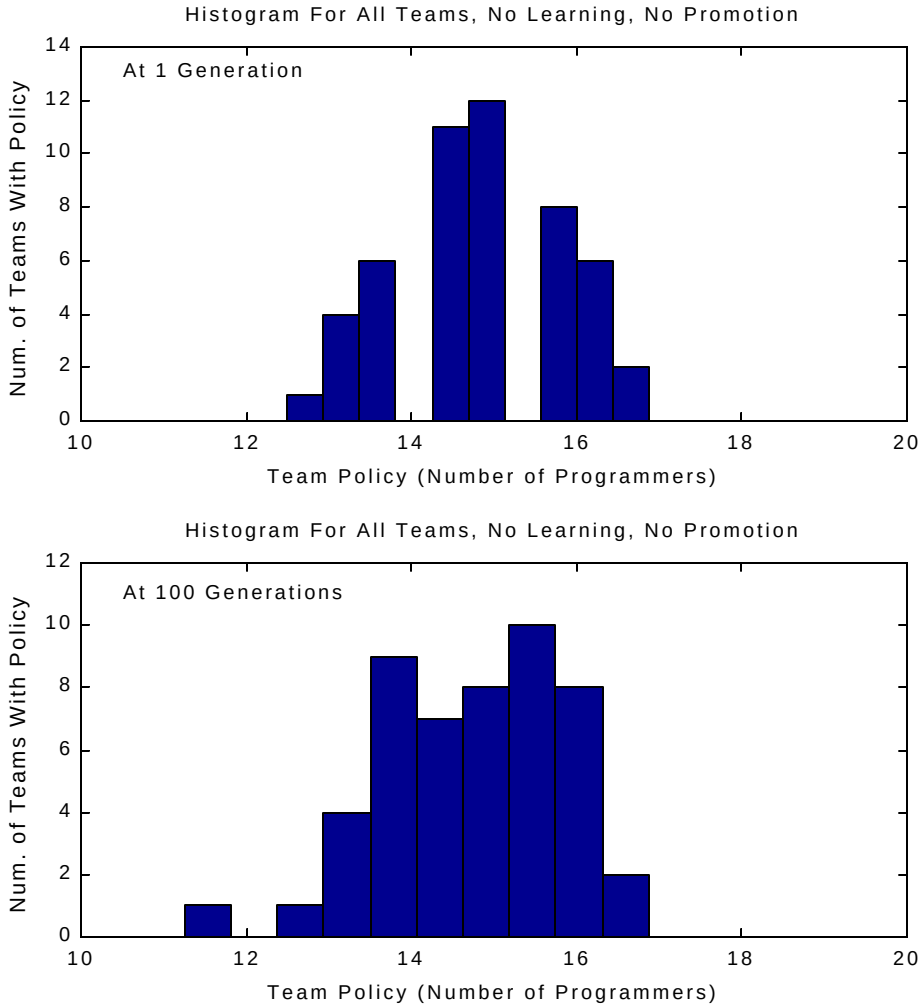

Figure 7-2. Histogram of team policies at generation 1 and at generation 100. Note how distribution is not uniform but normal. Generation 100 looks a little less normal due to sampling error (s_nlnp_r1p50u_4_11).

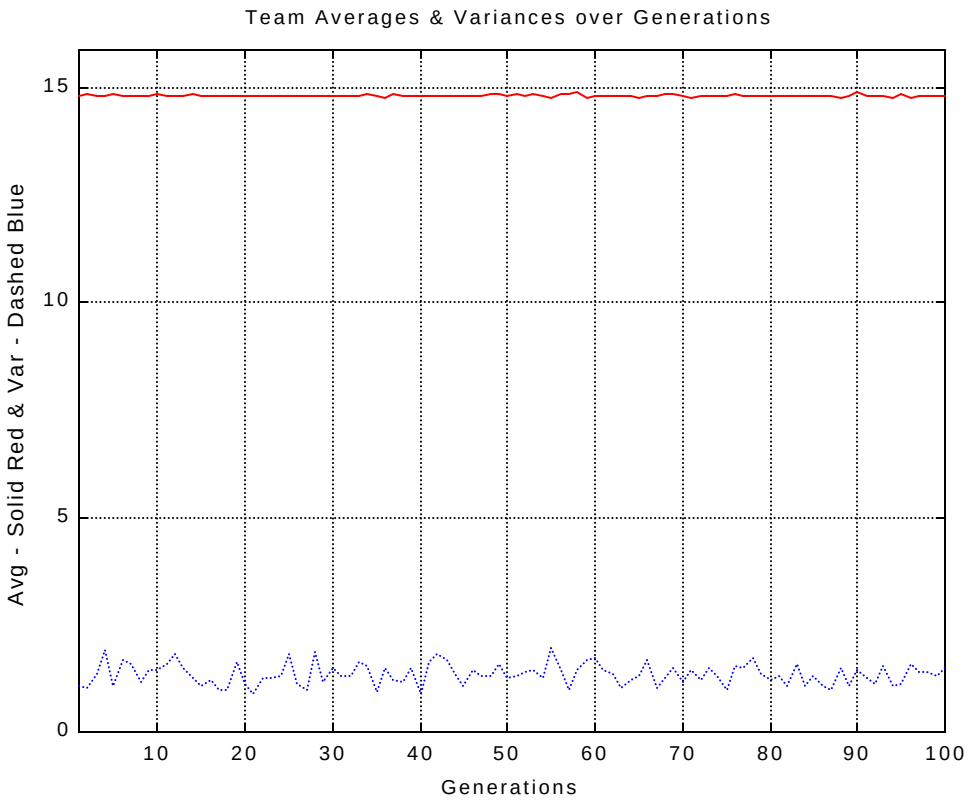

Figure 7-3. Average team policies and the variance over 100 generations (s_nlnp_r1p50u_4_11).

7.1.2 Mixed Team Company

Just as in the single or fully merged company case, it is desirable to understand how promotion and learning affects the evolution of a mixed team company. Once again, lets assume that policies for all individuals are randomly initialized using a uniform distribution over the range of 11 to 20 programmers. If this company uses a small number of mixed teams where there is no learning or promotion of individuals, we would still not expect the company to evolve. Though individuals from both companies are randomly selected to work together on team projects, new policies are not created, due to the lack of learning. Also, people with good policies have just as much say into the team policy as people with poor policies. Again, due to the random initialization of people's initial policy, we would expect each team to contain individuals with good, bad, and average policies. Thus, most of the team policies in the company would be "average," while some teams are above average and some are below. Therefore, this case is identical to the single company case, and we would expect the same results as before.

Figure 7-4 shows the histogram for all individuals within the company at generation 1 and at generation 100. Since learning does not occur, no individuals within the company will learn new policies,

thus, the histogram for individual policies should remain the same over the generations. Finally, we would expect the average of all the team policies to be around the average (15 programmers). We see that in Figure 7-6 (A-D), this is true for the whole company, each of the legacy companies, and within the mixed teams.

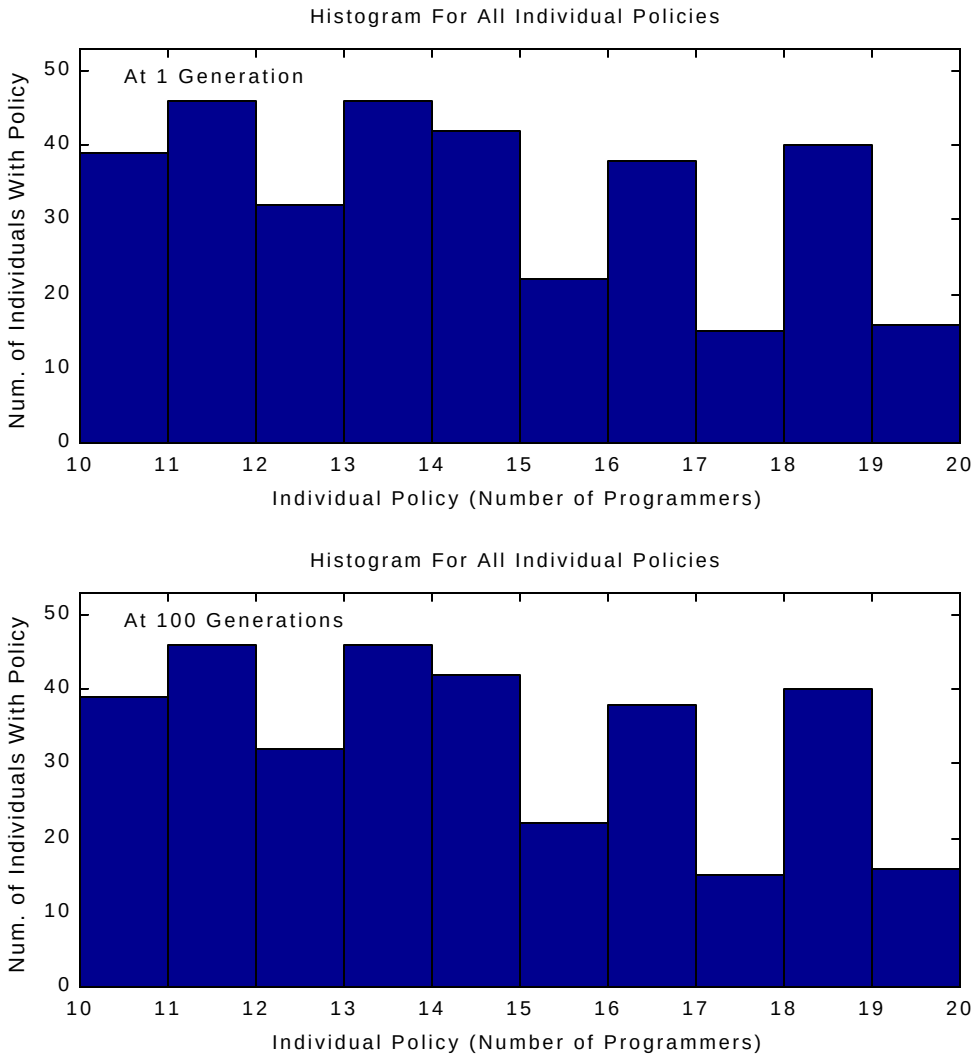

Figure 7-4. Histograms for individual policies in a mixed team company show a uniform distribution for the first and last generation. Note how the distribution remains the same over the generations (d_nlnp_r3m6p25u_4_11).

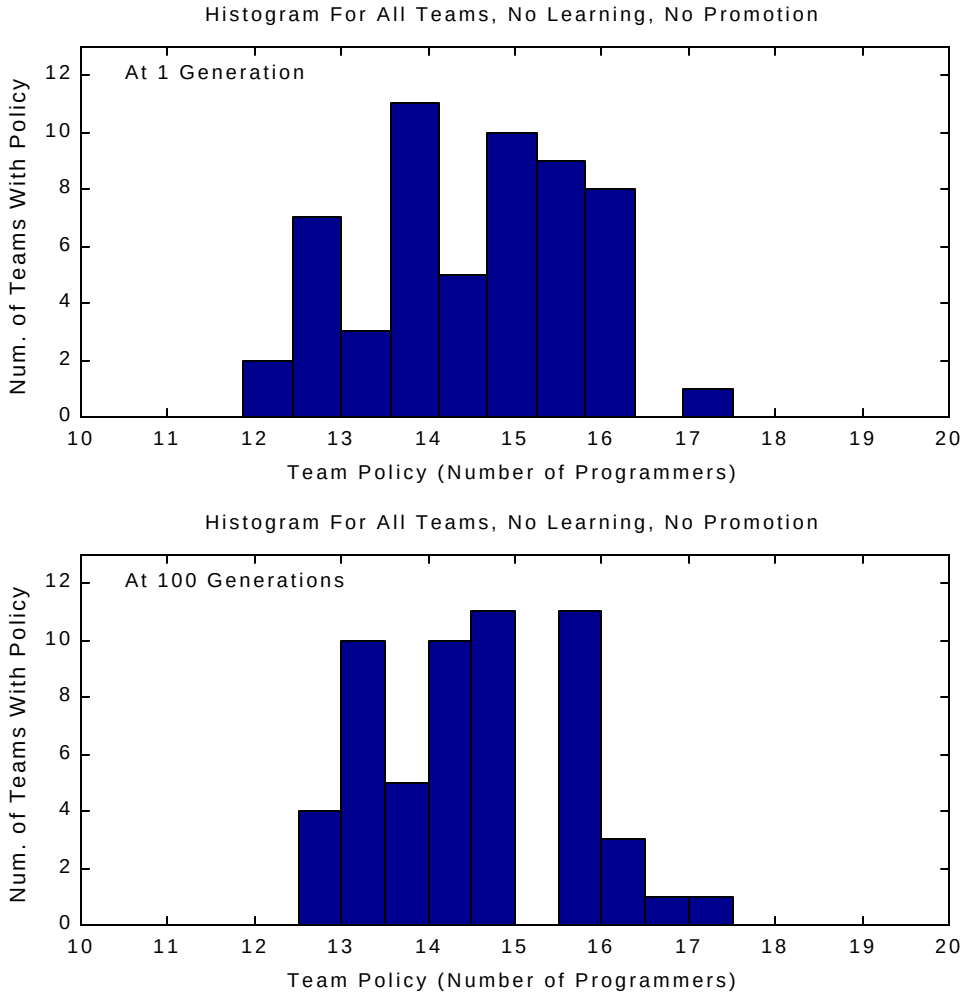

Figure 7-5. Histogram of team policies at generation 1 and at generation 100 for a mixed team company. Distribution of team policies should follow a normal distribution shape due to the central limit theorem. These distributions do not look perfectly normal because of sampling size and bin spacing (d_nlnp_r3m6p25u_4_11).

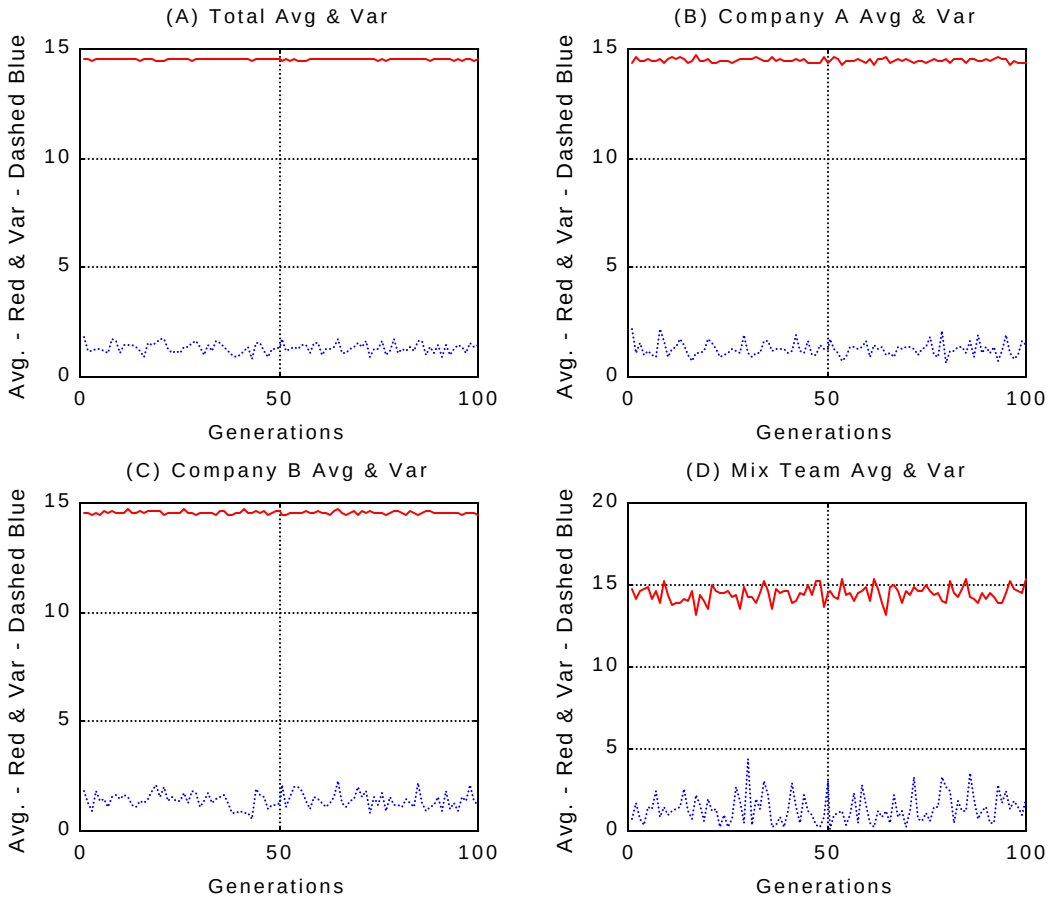

Figure 7-6. Average and variances of the team policies over 100 generations. (A) Whole company. (B) Legacy company A. (C) Legacy company B. (D) Mixed teams. The average and variance over generations is not as smooth as Company A or Company B's due to the small sample size (6 mixed teams).

Though it is nearly impossible to find a company where nobody learns from each other and individuals have equal status, it is important to understand how this company evolves over time. By understanding this very basic case, we can see how promotion and learning will affect merging companies.

7.2 Promotion Only

When a promotion scheme is used, we would expect that the policies generated by teams in a merged company to evolve to a better state. Lets assume that there is a company where no learning occurs; however, people's status within the company affects how much influence they have on a team policy.

Like before, we assume that initially there is a uniform distribution of policies over the range of 11 to 20 programmers. Since no learning occurs, no new policies are being created, and we expect the

histogram of individual policies in the first generation to be identical to the histogram of policies of the last generation. Again, this situation is not realistic, but it allows us to understand what the effects of promotion are in a controlled manner.

Promotion has an interesting effect on the team policies and how the company evolves. Initially, all individuals start off with the same status; hence, the team policies are just an average of all individual policies. As promotion occurs, individuals on teams that perform better become promoted, while individuals on teams that do not perform well become demoted. Recall from before that we expect this distribution of team policies to be a normal distribution. A few teams will be above average, because they consist of more individuals with better policies. A few teams will be below the average, because they consist of more individuals with worse policies. The rest of the teams will have about average policies. Since individuals do not learn and policies do not change between generations, we expect to have a normal distribution of team policies. However, as more generations go by, the weighting becomes more influential on the team policy creation. That is, individuals on teams that perform well are promoted, while individuals on teams that do not perform well are demoted. After several generations, we expect that some individuals in the population to have an extremely high status, while other individuals to have an extremely low status. Individuals with extremely high statuses will dominate the team policy, while the individuals with extremely low statuses will become insignificant in the creation of the team policy. As a result, we expect the average of all team policies to improve but never reach the optimal value. This is because there are teams that consist of individuals with just poor policies, thus, pulling the average of the team policies down. What is interesting is that even though promotions are based on teams, this process still identifies how individuals perform.

7.2.1 Fully Merged, Single Company

In this case, we have a promotion scheme in a company that has fully mixed all of the individuals. We expect the individuals to have the same policy distribution over the generations. Since the individual's weights influence team policies, in the condition where everybody has the same weighting, we expect a normal distribution of team policies. As weighting becomes more significant, the distribution shifts toward the better policies. As mentioned above, this is because individuals with good policies become more influential in team policy making, while individuals with poor policies become less influential in team policy making. Since individuals with poor policies become less significant and individuals with good policies become more significant, we would expect the average for all the team policies to be somewhere between the average policy and best policy. Thus, the company has evolved to a better state.

Figure 7-7 shows the histogram for policies of all individuals within the company at generation 1 and at generation 100. Policies were assigned to individuals using a uniform distribution. Since no

learning has taken place, no individual policies have changed, thus keeping the same individual policy distribution throughout the simulation. In Figure 7-8, we have the distribution of team policies at generation 1 and at generation 100. At generation 1, promotion has not taken effect yet, so we have a normal shaped distribution. As an individual's status begins to change, individuals with stronger policies have more influence on team policies, due to their weighting value. Individuals with weaker policies have less influence on the outcome of the team policy. Therefore, the distribution for the team policies shifts towards the more optimal policy value, as shown in generation 100 of Figure 7-8. Since there is no learning, we still expect some teams to create poor policies. Figure 7-9 shows the average of team policies over 100 generations. Promotion is turned off for the first 10 generations. From generation 10 to 30, we see an increase of the average of team policies. From there, a steady state value of around 17 programmers is reached. Note the increase in the variance of team policies. Looking at Figure 7-8, we can see that the cause of this variance increase is due to the increased spread of team policies between generation 1 and generation 100. This spread in team policies is due to the initial wide range of individual policies along with the effects of promoting.

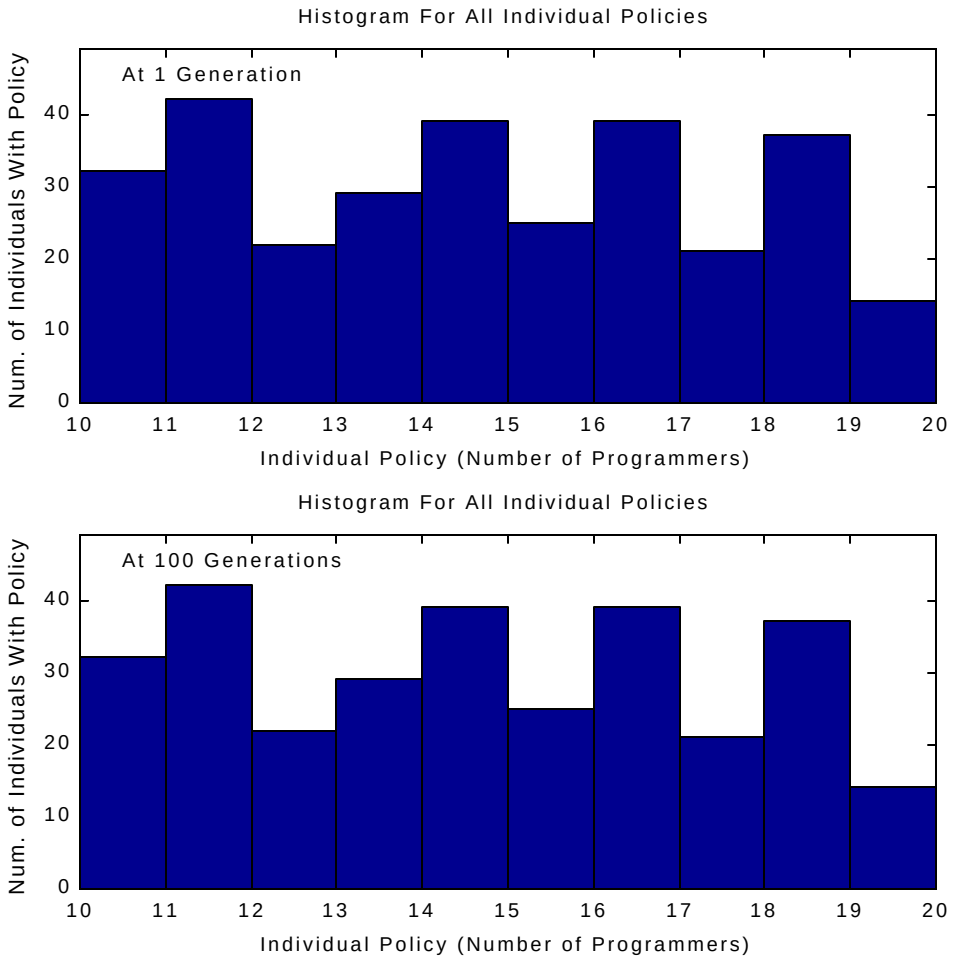

Figure 7-7. Histograms of individual policies in a company with promotion only. Note the identical policy distributions at generation 1 and at generation 100 (s_p_r10p50u_4_11).

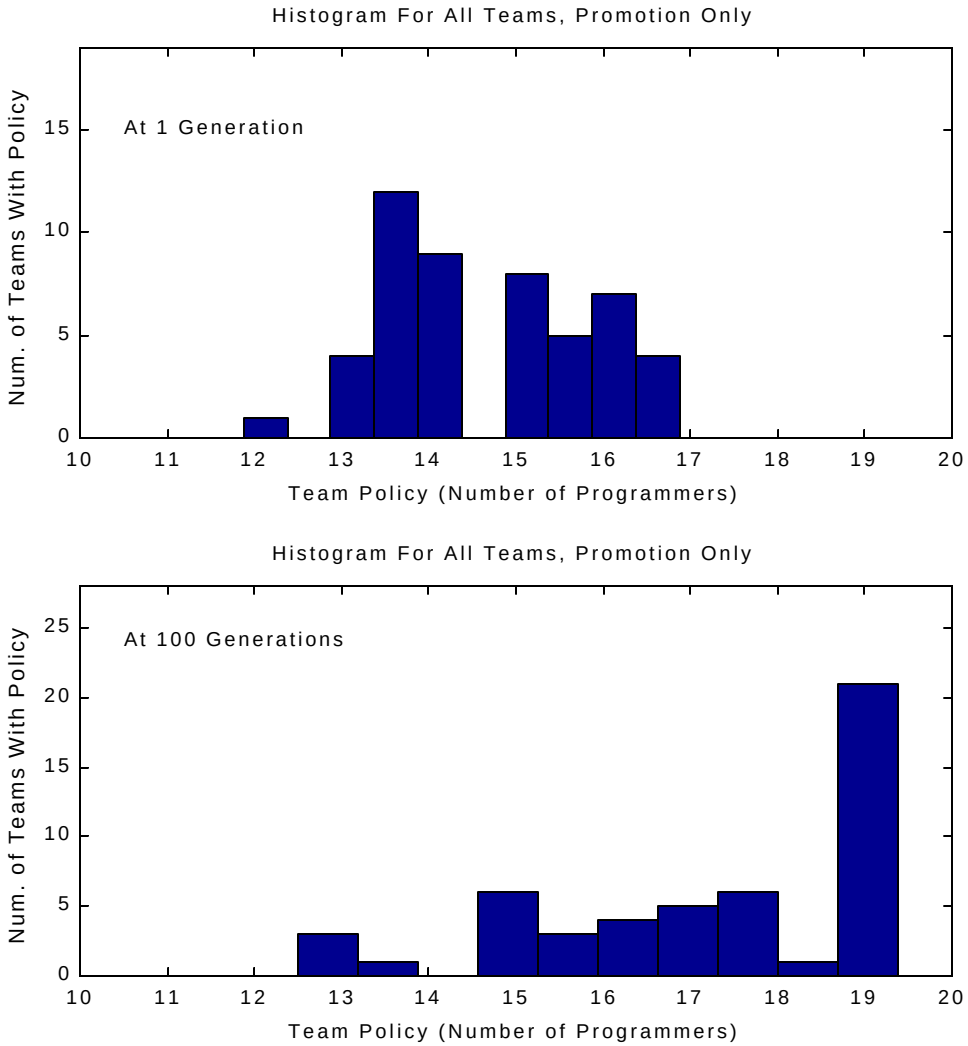

Figure 7-8. Histogram of team policies in a company with promotion. In generation 1, the distribution of team policies follows a normal distribution. By generation 100, weighting to individual policies has skewed the team policy distribution toward better values (s_p_r10p50u_4_11).

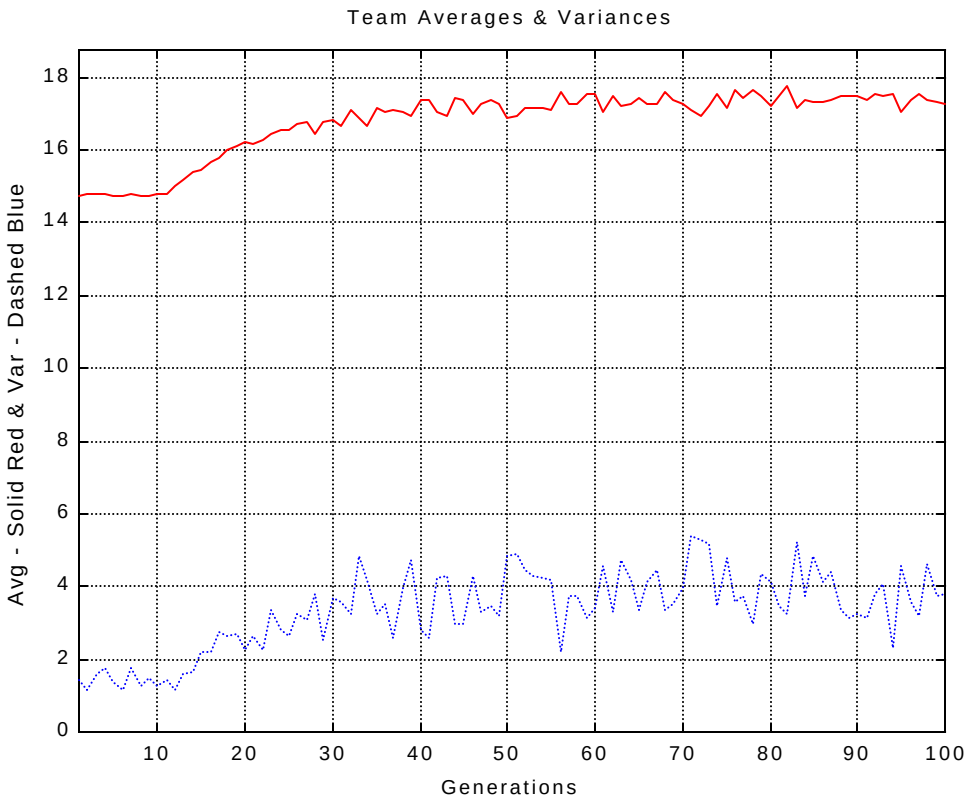

Figure 7-9. Average and variance of company team policies over 100 generations. Promotion is turned on after 10 generations (s_p_r10p50u_4_11).

7.2.2 Mixed Team Company

In the mixed team case, there are different ways to rank teams. One method is to rank teams from each company separately. Another method is to rank all teams from both companies together. Depending upon the goal of the merger, both methods are valid. For example, suppose a defense company that specialized in missiles and radars developed a new cooking method and wanted to enter a new market (this is actually how the microwave oven was introduced). To do this, they acquire an appliance company and setup mixed teams to develop this product. In this case, it makes sense to evaluate each of the company's teams separately. Since the defense market is quite different from the washer and dryer market, these products vastly differ in their requirements, resources, manufacturing needs, development time, functions, and much more. It would be very difficult to determine if a missile project was more or less successful than a washer project. However, suppose that this defense company had also acquired another missile company because they were able to build better heat seeking missiles. Mixed teams would be used to improve the knowledge transfer between companies. In this case, it would make sense for all the project teams from both companies to be evaluated against each other. These projects would most likely have similar development times, resource requirements, manufacturing needs,

etc. Because of this, it is easier to determine if one project team performed better than another. Also, we can look at the single company approach, where individuals from two companies are fully mixed together. It makes more sense to mix two missile companies together than it does mixing a missile company with an appliance company. As a result, we will rank all team from both companies together in the simulations.

In the mixed team case, we would expect promotion to have the same effects on average team policies as in the fully merged company case. As mentioned above, promotion is being performed on the whole company, i.e., all teams are evaluated against each other. Figure 7-10 shows the histogram for policies of all individuals within the company at generation 1 and at generation 100. Policies were assigned to individuals using a uniform distribution. Since no learning has taken place, no individual policies have changed, thus keeping the same individual policy distribution throughout the simulation. In Figure 7-11, we have the distribution of team policies at generation 1 and at generation 100. At generation 1, promotion has not yet taken effect, so we have a normal shaped distribution. As an individual's status begins to change, the individuals with stronger policies will have more influence over team policies, due to their weighting value. Individuals with weaker policies will have less influence on the outcome of the team policy. This causes the distribution of team policies to shift toward the more optimal policy value, as shown in generation 100 of Figure 7-11. Since there is no learning, we still expect some teams to create poor policies. Figure 7-12 (A) shows the average of all team policies over 100 generations. Figure 7-12 (B) and (C) show the average for each legacy company, and Figure 7-12 (D) show the average of the mixed teams only. Promotion is turned off for the first 10 generations. From generation 11 to 30, we see an increase of the average of team policies, where it seems to reach a steady state value of around 17 programmers. The average team policy has improved but has not reached the optimal value. Note the increase in the variance of team policies. This is due to the wider spread of team policies.

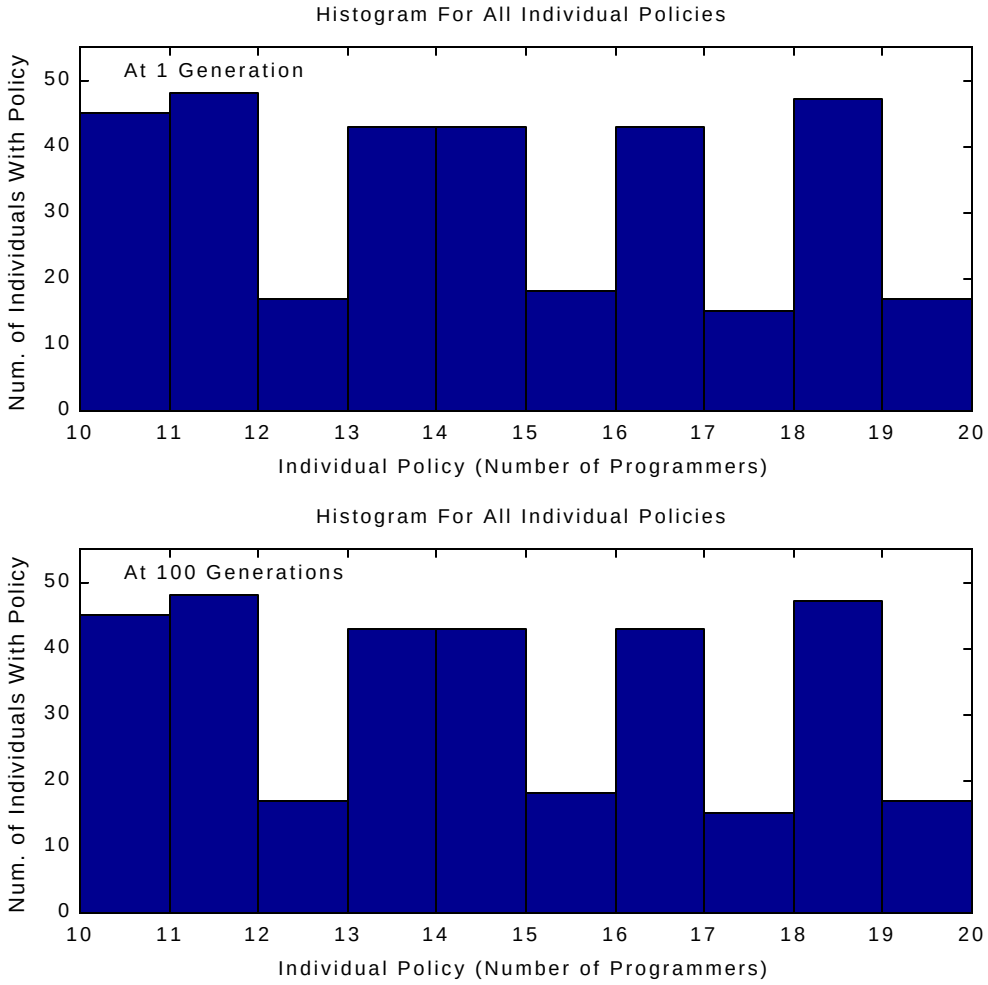

Figure 7-10. Distribution of individual policies in a mixed team company with promotion, at generation 1 and generation 100. Note how the distribution remains the same (d_p_r10m6p25u_4_11).

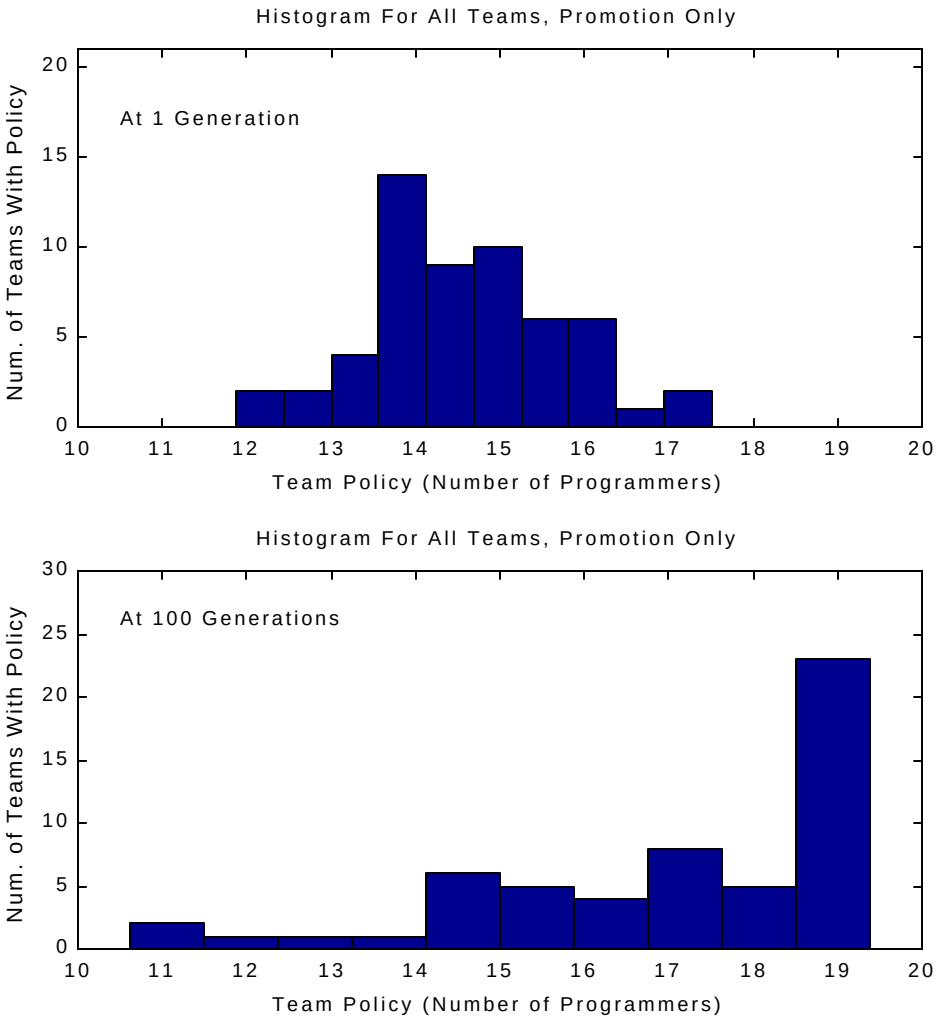

Figure 7-11. Team policies evolve from a normal distribution to a skewed distribution due to the effects of promotion (d_p_r10m6p25u_4_11).

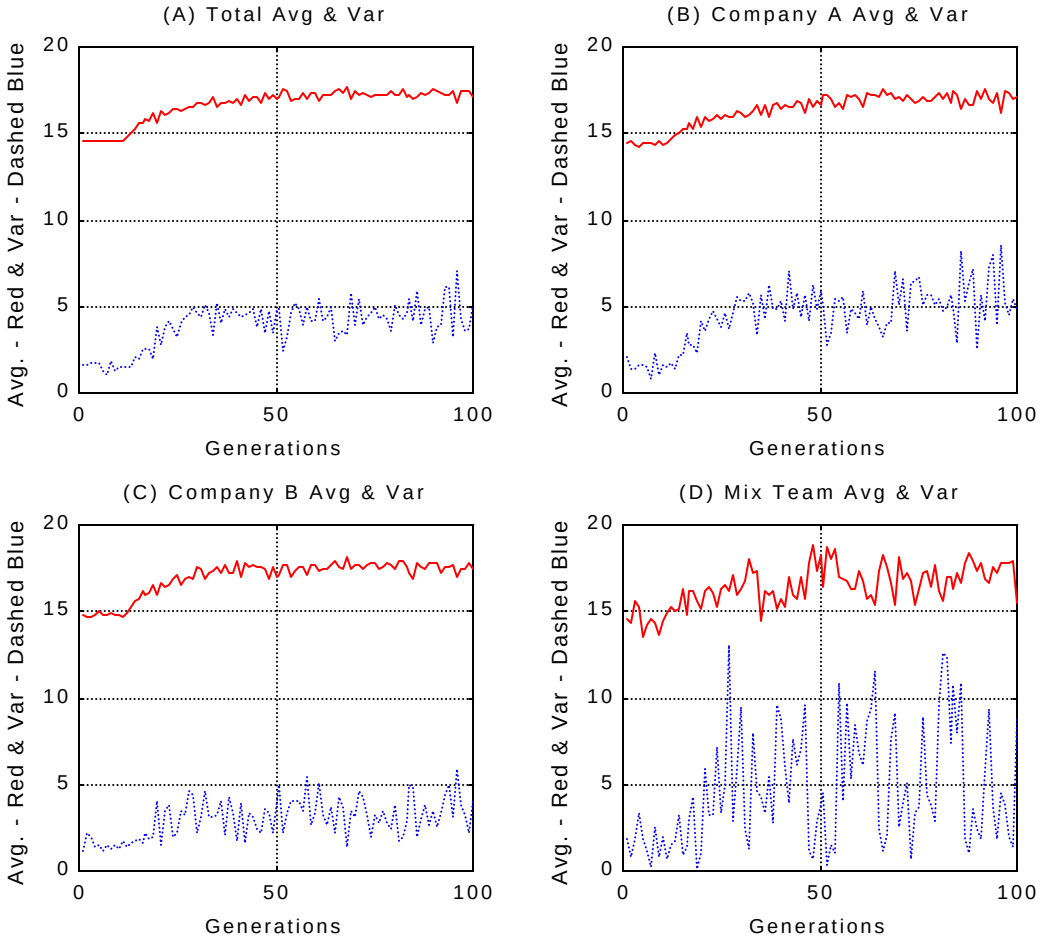

Figure 7-12. Average and variance of team policies over 100 generations. Promotion is turned on after 10 generations. (A) Whole company. (B) Legacy company A. (C) Legacy company B. (D) Mixed teams (d_p_r10m6p25u_4_11).

7.3 Learning Only

To understand the effects of learning on a population, let us assume that there is no promotion within a company. This is also another unrealistic situation; however, it will let us understand the affects of learning under controlled conditions. Since there is no promotion, individuals with good policies will have the same influence on the team policy as individuals with poor policies. This also means that individuals on a team would randomly learn from each other. As a consequence, an individual will be just as likely to learn from someone with a good policy as a poor one. Hence, we would expect the average for all team policies within the company to be around average (around 15 programmers in this case). Depending upon the initial distribution of the individual policies and who learns from whom, this average

can be higher or lower than expected. For the most part, we would expect that polices generated by the teams to be about average, whether we are looking at a mixed team structure or fully merged company.

7.3.1 Fully Merged, Single Company

As mentioned above, we do not expect to see any improvement in the average of the team policies. In the fully merged, single company case, we initialize the individual's policy using a uniform distribution. Figure 7-13 shows the distribution of the policy for individuals at generation 1 and at generation 100. Since learning is occurring, individuals are learning new policies at each generation; thus, the policy distribution at generation 1 is different from the policy distribution at generation 100. Figure 7-14 shows the distribution of team policies at generation 1 and at generation 100. As we can see from Figure 7-14, the average of the team policies is around 15 programmers. Finally, Figure 7-15 and Figure 7-16 shows how the average policy can improve or become worse in two different runs. Due to the randomness, the average of team policies will drift around. If we introduced promotion into the simulation, we would expect that the status of individuals would provide direction for learning.

Taking a closer look at generation 100 in Figure 7-13, we see that the distribution is no longer uniform. Specifically, many individuals in the population seemed to have come to a consensus of around 17 programmers. To understand consensus of a policy better, lets look at the population size and the range of policy values. Within this organization, there are 300 individuals (50 teams of 6 individuals) and 10 valid policy values (11 to 20 programmers). Due to the limited number of policy choices, it is easier for one policy to be more randomly selected by individuals for recombination. This causes the policy to spread throughout the population, hence, causing consensus. An everyday example of this would be trying to get a large group of people to decide on a restaurant to eat at in Boston. Since everybody's preferences are different and the selection of restaurants is large, it would be very difficult for the group to come to a consensus. However, if we limit the group's choice of restaurants to the Kendall Square area, it would be much easier for the group to come to a consensus.

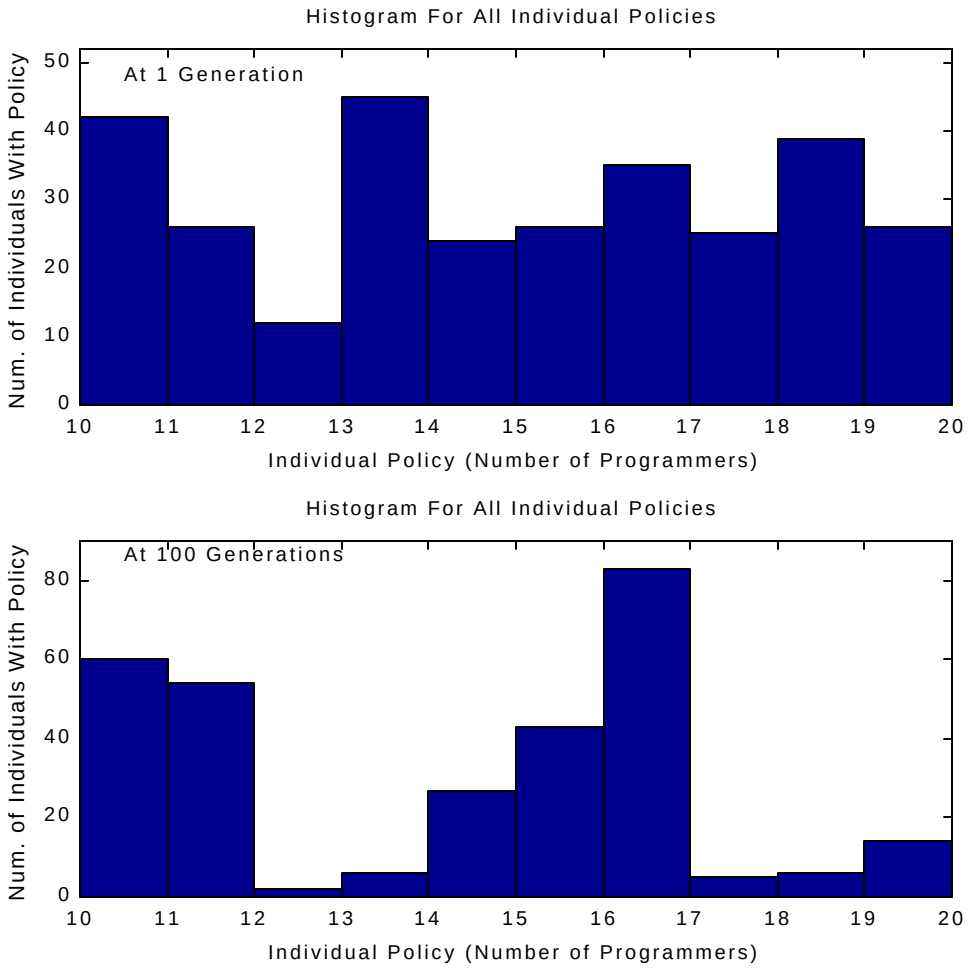

Figure 7-13. Learning causes individuals to change their policies over the generations (s_l_r10p50u_4_11).

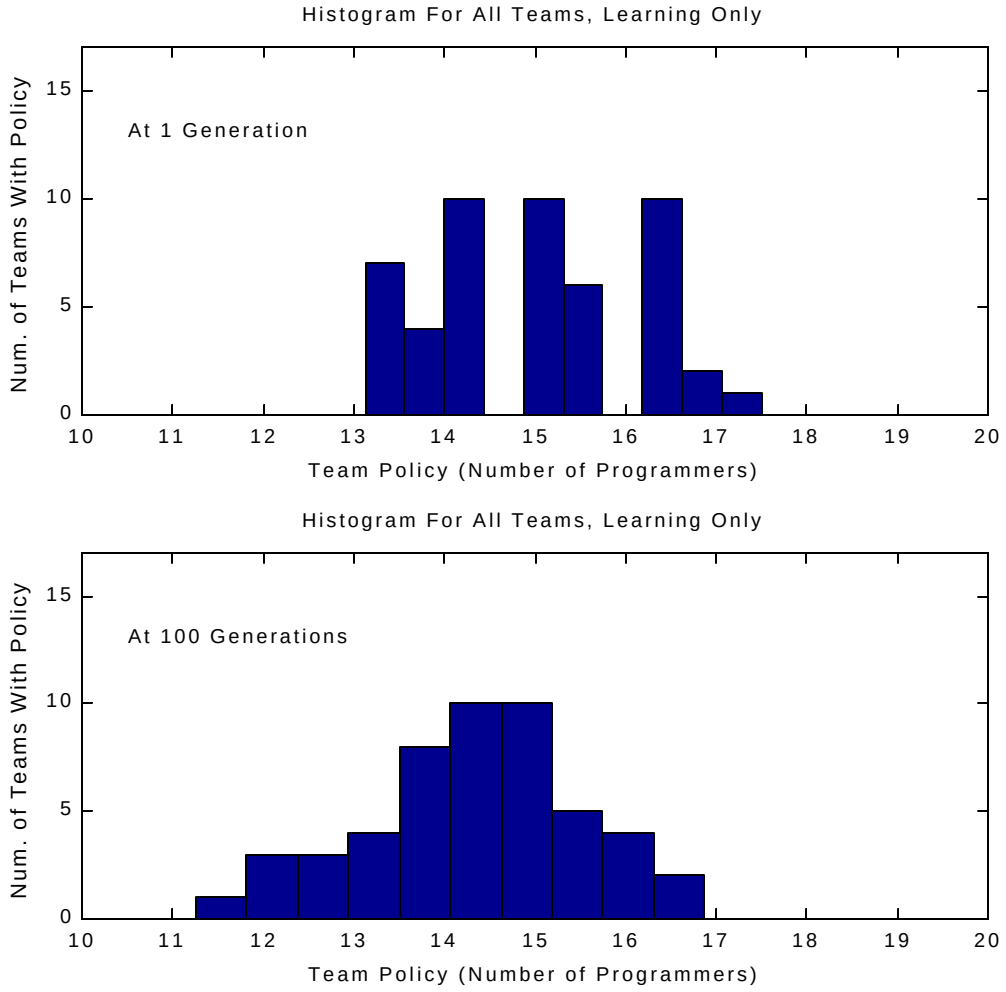

Figure 7-14. Learning causes the team policy distribution to vary (s_l_r10p50u_4_11).

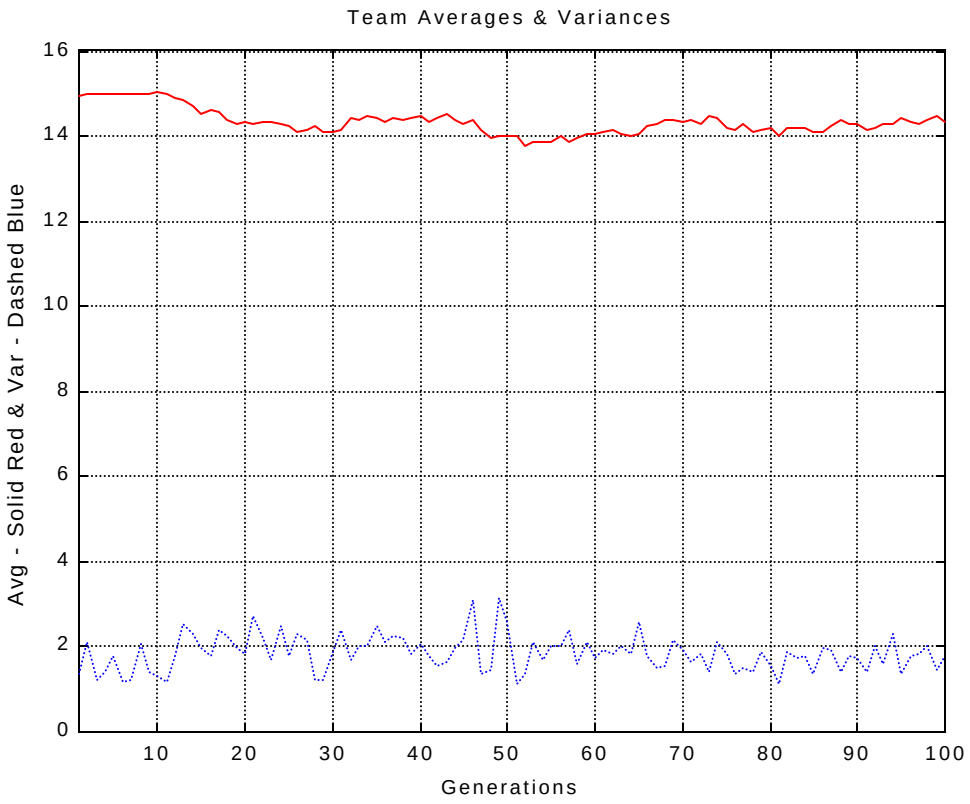

Figure 7-15. Average and variance of team policies over 100 generations. In this simulation, learning is turned on after 10 generations, and the average team policy drifts toward a worse state (s_l_r10p50u_4_11).

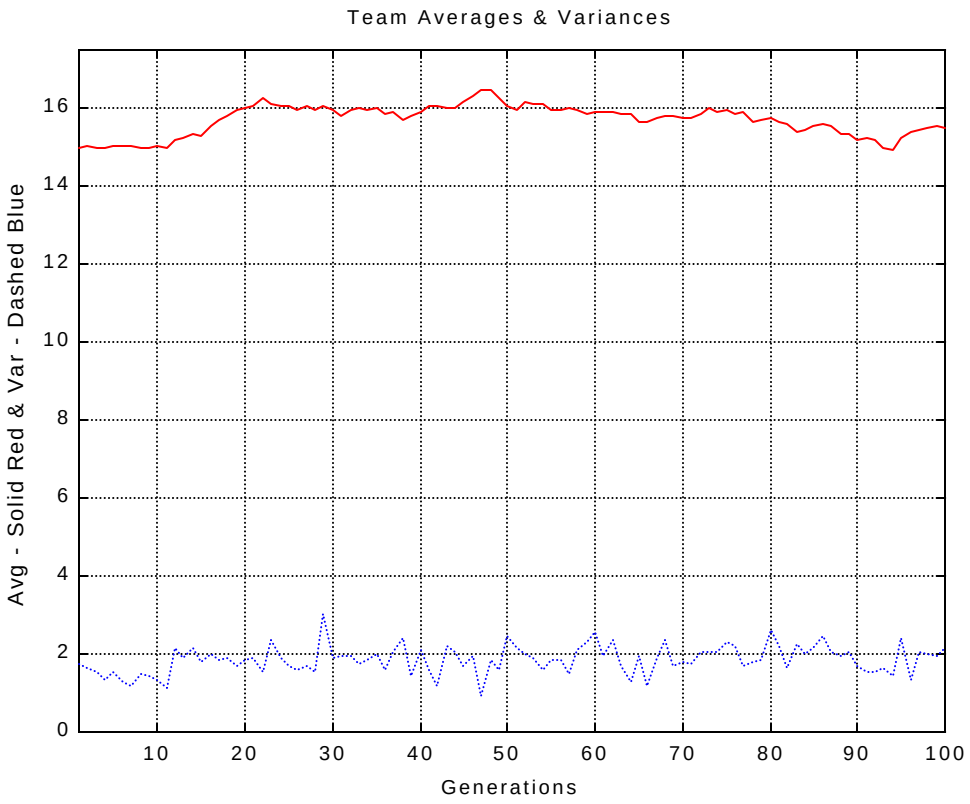

Figure 7-16 Average team policy drifts toward an improved state (s_l_r7p50u_4_11).

7.3.2 Mixed Team Company

Like the fully merged company case, we expect that having mixed teams with learning to react in the same way. Policies for the individuals are initialized randomly, using a uniform distribution. Figure 7-17 shows the distribution of the policy for individuals at generation 1 and at generation 100. Since learning is occurring, individuals are learning new policies at each generation; thus, the policy distribution at generation 1 is different from the policy distribution at generation 100. Again, we see the effects of consensus. Figure 7-18 shows the distribution of team policies at generation 1 and at generation 100. As we can see from Figure 7-18, the average of the team policies is around 15 programmers. Finally, Figure 7-19 (A) shows the average of team policies of the whole company over 100 generations. Due to the randomness of learning, this average drifts around; however, it remains around 15 programmers. In Figure 7-19 (B) and (C), we see the average team policy in both legacy company. Figure 7-19 (D) shows the average team policy within the mixed teams over 100 generations.

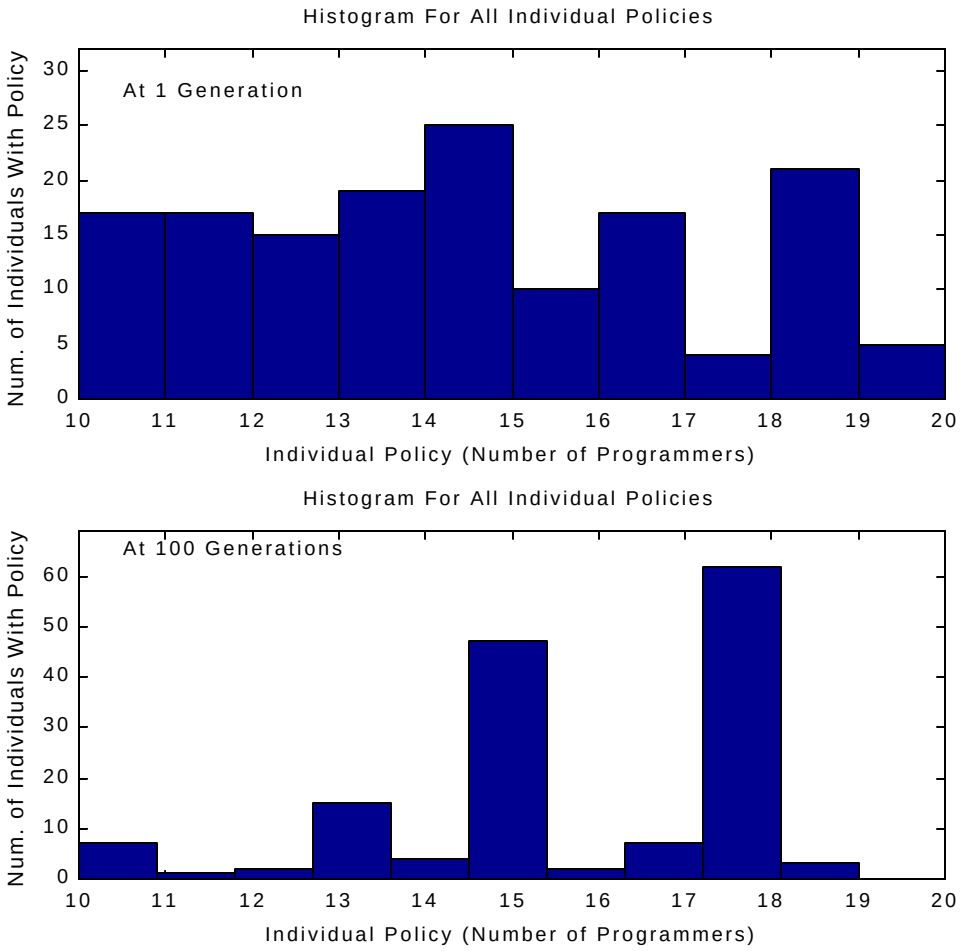

Figure 7-17. Distribution of individual's policies change with learning. Generation 1 and generation 100 is shown here (d_l_r10m6p25u_4_11).

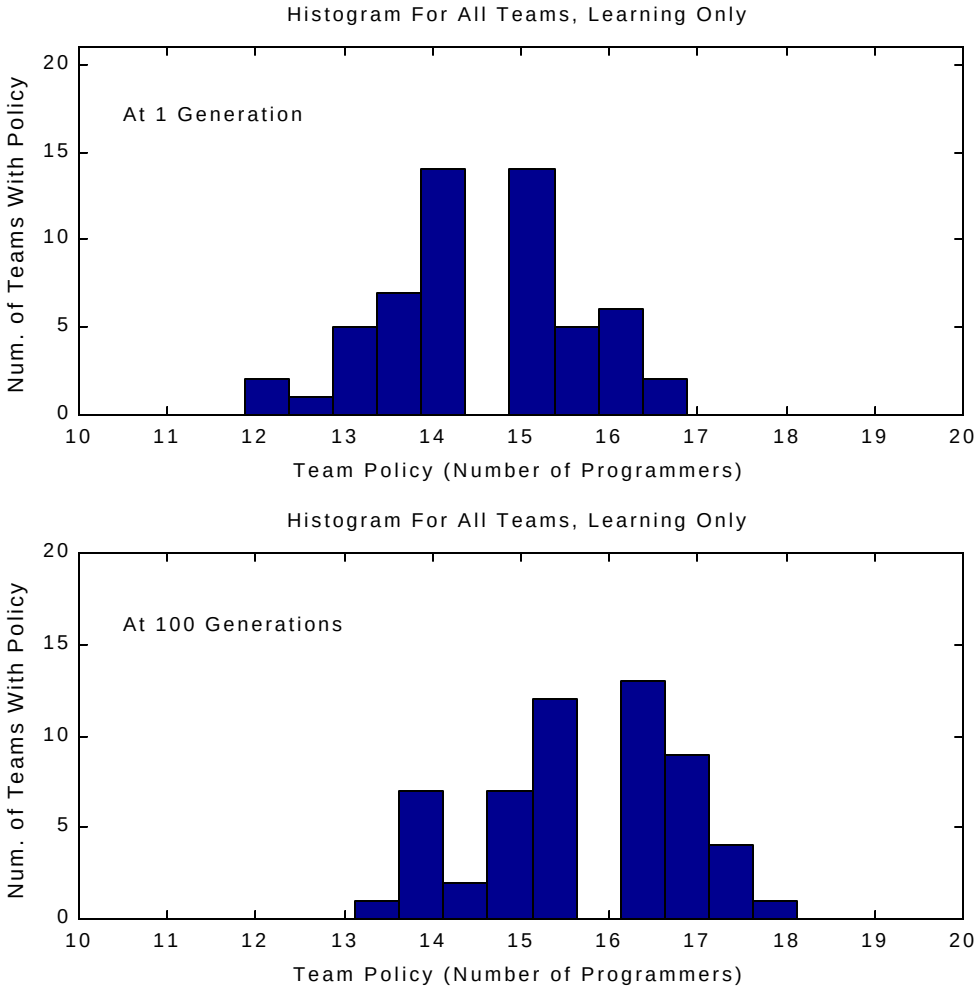

Figure 7-18. Distribution of team policies at generation 1 and generation 100 (d_l_r10m6p25u_4_11).

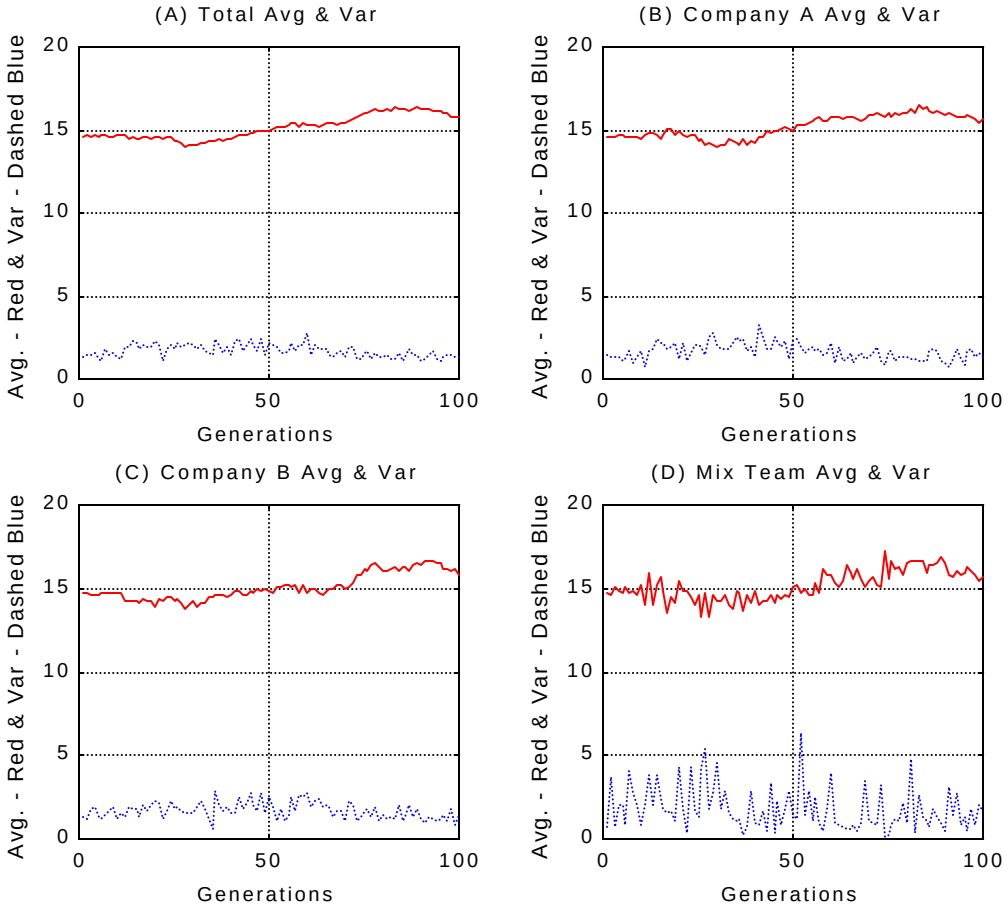

Figure 7-19. In this simulation, learning is turned on after 10 generations. Note how average team policy drifts around. (A) Whole company. (B) Legacy company A. (C) Legacy company B. (D) Mixed teams (d_l_r10m6p25u_4_11).

7.4 Learning and Promotion

Now that we have investigated the effects of promotion and learning alone, let's see what happens when we put them together. In a company where promotion and learning takes place, we would expect to see that the company would evolve to a much better state. Promotions allows for individuals with good policies to gain more status and individuals with poor policies to lose status. Because of this, we would expect to see some improvement within the company. Individuals with good policies will have greater influence over the team policies than individuals with poor policies. If we now introduce learning, individuals are now able to change their policies. An individual with high status on a team will have a greater probability of being selected by another individual for learning. Thus, an individual with a poor

policy might improve their policy from learning from an individual with a good policy. If that team is ranked high, individuals on that team will be promoted and their status will go up. This does two things for when individuals move onto their next project. It increases their influence on the team policy. It also improves the probability that they will be selected by another individual to learn from. Learning (recombination) allows an individual with a poor policy to create a good policy. Furthermore, individuals with good policies that may not be optimal can recombine with other individuals to create an even better policy than before. Because of this, we would expect that the average of team policies to improve over several generations. Eventually, we would expect individual policies to converge to some optimal value because of the learning and promoting.

7.4.1 Fully Merged, Single Company

As mentioned above, we expect to see improvement in team policies and individual policies due to the effects of promotion and learning. In this fully merged, single company case, we initialize the individual's policy using a uniform distribution. Figure 7-20 shows the distribution of the policy for individuals at generation 1 and at generation 100. Since promotion and learning occur, individuals will tend to learn good policies at each generation. We expect that the distribution of individual and team policies to improve. At generation 100, we see that all of the individual policies have converged to 20 programmers. This is the best policy value and we had expected this due to the learning and promotion. Figure 7-21 shows the distribution of team policies at generation 1 and at generation 100. As we can see from Figure 7-21, the average of the team policies at generation 1 is a normal distribution around 15 programmers. By generation 100, the team policies have also converged to the best policy of 20 programmers on the project. Finally, Figure 7-22 shows the average of team policies over 100 generations. Promotion and learning is turned off for the first 10 generations. As expected, the average team policy value is 15 programmers. From generation 11 to 40, promotion is turned on and we see an improvement of team policy values. Under these conditions, the average of the team policy plateaus out at around 17 programmers. However, since no learning occurs, the variance increases due to the wider spread of team policies (recall from the promoting section). From generation 41 to 100, promotion and learning is turned on and we see another rise in the average team policy value. This value approaches 20 programmers. Also, the variance drops back down to zero, since the individuals have recombined to learn the best policy. By having promotion and learning in the simulation, we are able to provide direction and drive for the evolution of policies within a fully merged company.

One other point to note here is the effect of turning promotion on before learning. Though we would expect the same results if promoting and learning were turned on at the same time, there could be several generations of "counterproductive learning" before teams begin to improve. In Figure 7-22, we see that promoting is performed in generations 10 to 40, and that learning is turned on after generation 40. Notice how rapidly we converge when learning is turned on. Under these conditions, an individual's

status has already been determined within the company, due to the pre-promotion. Therefore, individuals with good policies will already have a higher status and will most likely be selected for recombination by others. If we turned on learning at the same time as promotion, there would be no clear initial distinction between individuals with good or bad policies, because everyone starts off with equal status. The question now becomes do we promote first? That is, does it make sense to sort out individuals with good and bad policies before learning is applied? It can be argued that promoting before learning has already been done to a limited extent, since companies have already evolved somewhat before a merger. What is not known is the status of individuals in one company relative to the other. An individual in Company A could be an outstanding performer; however, compared to Company B's standards, this individual would be just average. Because of this ambiguity of status between companies, it would be a good idea to sort out individuals with good and bad policies before learning, and to rank all teams from both companies together.

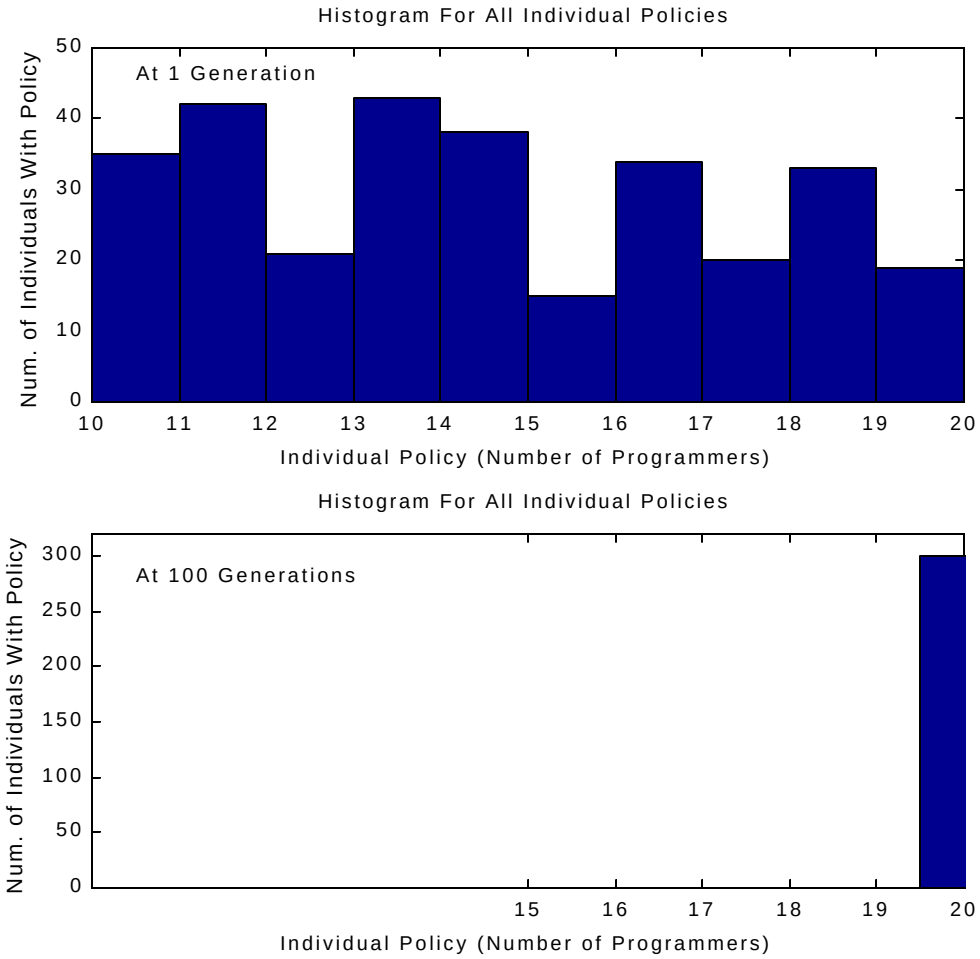

Figure 7-20. With learning and promotion, individual policies improve over the generations (s_lp_r1p50u_4_12).

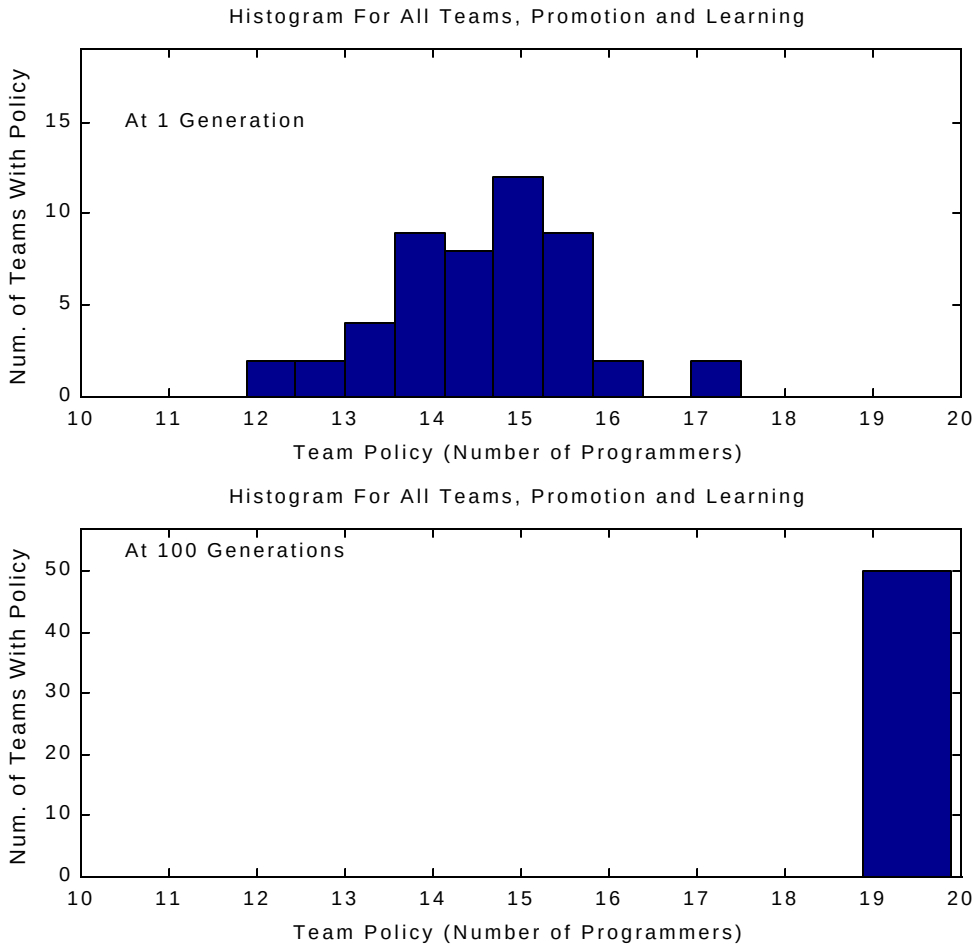

Figure 7-21. Team policies also evolve over the generations with learning and promotion (s_lp_r1p50u_4_12).

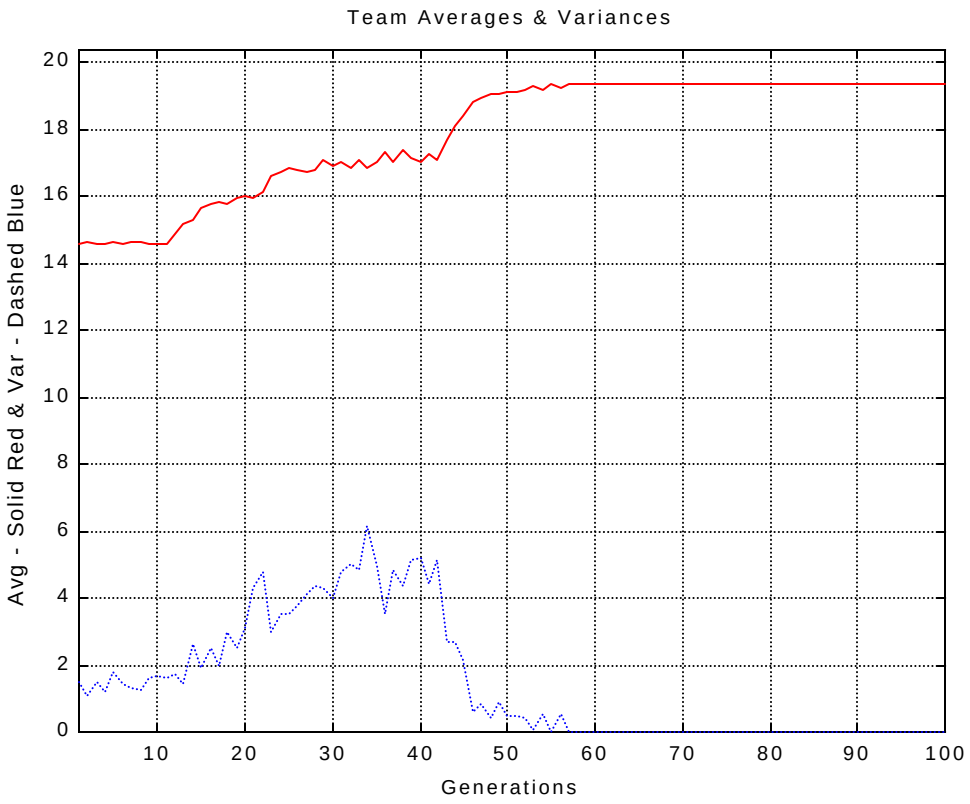

Figure 7-22. Average team policy and variance over 100 generations. Promotion is turned on after 10 generations and learning is turned on after 40 generations (s_lp_r1p50u_4_12).

7.4.2 Mixed Team Company

In the mixed team case, individuals are randomly assigned a policy value using a uniform distribution. Figure 7-23 shows the distribution of the policy for individuals at generation 1 and at generation 100. Since promotion and learning occurs, individuals will tend to learn good policies at each generation; thus, we expect that the distribution of policies to improve. At generation 100, we see that all of the individual policies have converged to 20 programmers. This is the best policy value and we had expected this due to learning and promoting. Figure 7-24 shows the distribution of team policies at generation 1 and at generation 100. As we can see from Figure 7-24, the average of the team policies at generation 1 is a normal distribution around 15 programmers. By generation 100, the team policies have also converged to the best policy of 20 programmers on the project. Finally, Figure 7-25 (A), (B), and (C) shows the average of team policies over 100 generations for all the teams within the company, all the teams in legacy company A, and all the teams in legacy company B respectively. Promotion and learning is turned off for the first 10 generations. As expected, the average team policy value is 15 programmers. From generation 11 to 40, promotion is turned on and we see an improvement of team policy values. Under

these conditions, the average of the team policy plateaus out at around 17 programmers. Since no learning occurs, the variance increases due to the spreading of team policies over a wider range. From generation 41 to 100, promotion and learning is turned on and we see another rise in the average team policy value. This is when individuals with poor policies learn from individuals with good policies. This improves the average team policy value, and over time, this value approaches 20 programmers. Notice that the rate of improvement from learning and promoting in the mixed team case (after generation 40) is slower than in the single company case in Figure 7-22. This is because a smaller number of individuals are mixing in the mixed team structure as compared to the single company structure. Also, the variance drops back down to zero, since all of the individuals have learned the same policy. As we can see, the mixed team case produces the same results as the fully merged company case.

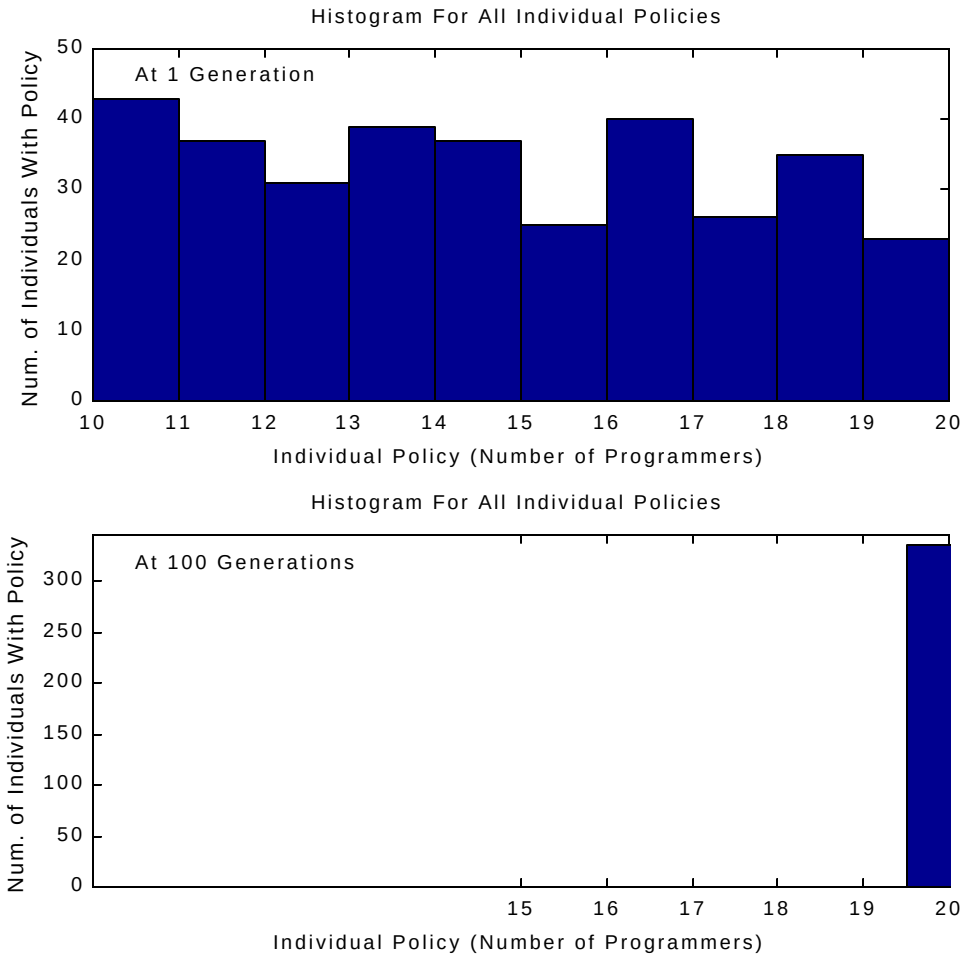

Figure 7-23. With promotion and learning, individual policies evolve over the generations (d_lp_r1m6p25u_4_12).

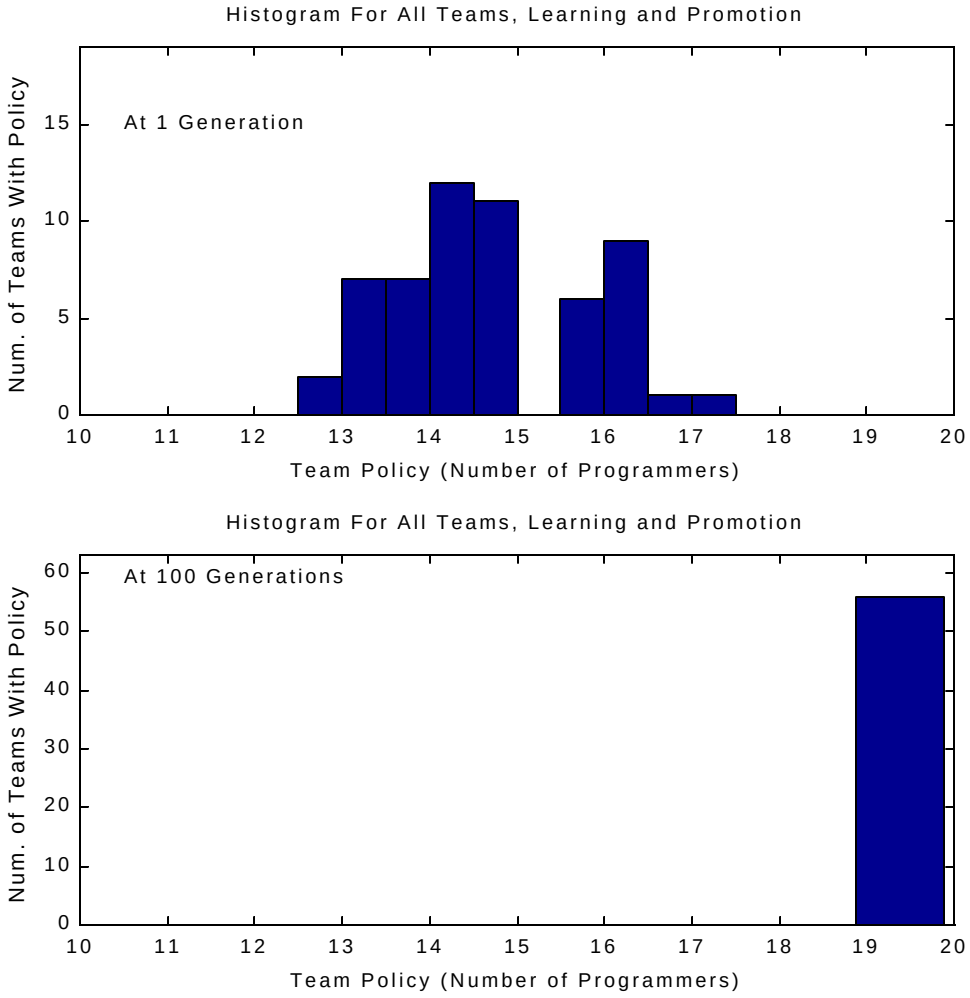

Figure 7-24. Team policies evolve from a normal distribution to a single policy due to learning and promotion (d_lp_r1m6p25u_4_12).

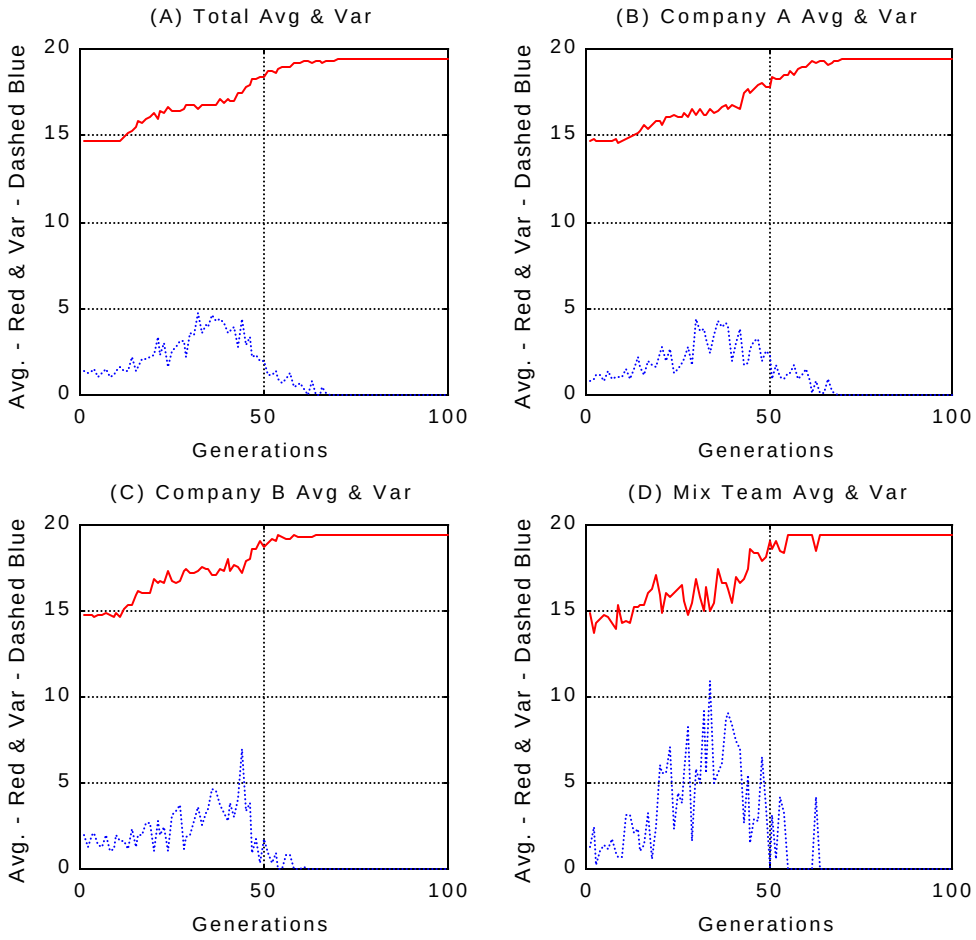

Figure 7-25. Evolution of the average team policy in a mixed team structure over 100 generations. Promotion is turned on after 10 generations and learning is turned on after 40 generations. (A) Whole company. (B) Legacy company A. (C) Legacy company B. (D) Mixed teams (d_lp_r1m6p25u_4_12).

7.5 Mixed Team Policy With Good and Bad Policy

Now that we understand the effects of learning and promotion in the fully mixed company structure and the mixed team structure, lets see what happens when we merge a poor policy company with a good policy company in the mixed team structure.

Suppose Company A is a company with poor policies and Company B is a company with good policies. In Company A, individuals have a policy value of around 12 programmers on a project. Individuals in Company B have a policy value of 19 programmers on a project. This individual policy distribution is shown in Figure 7-26 at generation 1. Policies are clustered either around 12 or 19 programmers. If we look at the team policy distribution at generation 1, we would expect that team policies in Company A to be around 12 programmers per team policies in Company B to be around 19 programmers per team. Since individuals in Company A and Company B are equally represented on mixed teams and everybody has equal weighting, we would expect the mixed teams to produce policies around 15. Figure 7-27, generation 1 shows the initial distribution of team policies.

From what has been learned so far, we would expect that the company with poor policies to evolve to a better state. The key to making this happen is to allow the effect of promoting and learning on the mixed teams. Since the teams in Company B already have good policies, learning will not have a great effect there. Company A will not be able to evolve also, since individuals in that company will only learn poor policies, which they already know. Since promoting on the whole company has been implemented first, the status of the individuals in Company B improves, while the status of the individuals in Company A worsens. Initially, we expect the policy of mixed teams to be somewhere between the best policy and the worst policy, since mixed teams are equally represented by both companies. Eventually, the individuals from Company B that are sent to the mixed teams will have a higher status than the individuals sent by Company A. This shift in status causes the policies created on the mixed teams to be much closer to the better policies created by the teams in Company B. We expect to see an improvement in the overall company performance, due to the effects of promoting. Once learning occurs with promoting, good policies on the mixed teams will be transferred effectively and the organization as a whole will evolve even further. Because individuals from Company B on the mixed teams have higher status than individuals from Company A, the individuals from Company A will tend to learn from individuals from Company B. Since the mixed teams perform better than Company A's teams, individuals from Company A are able to improve their policies through learning. Individuals return back to Company A with improved policies and higher status. Their improved status will cause other individuals on their new team to select them for learning. Furthermore, their improved status will also cause their policy to be weighted heavier in the creation of the team policy. Thus, these teams perform better, and individuals in Company A are able to learn better policies. After several generations, we expect that the individuals in Company A to learn the good policies of Company B. If an individual from Company B learns a poor policy from and individual in Company A while on a mixed team, we don't expect that to be a problem.

When that individual returns to Company B, they will relearn the better policies from other members of their company.

Figure 7-28 shows the average team policy over 100 generations. For the first 10 generations, there is no learning and no promotion. As we would expect, an average of all the team policies yields an average policy value of about 15. Promotion is turned on after 10 generations; however, we do not see a large overall affect to the company. Since the mixed teams accounts for only a small percentage of the total teams, we do not see a drastic increase in the total company performance. Learning is turned on after generation 40. This has a dramatic effect on the average team policies in Company A, as shown in Figure 7-28 (B). After a few generations, policies in Company A improve and eventually become identical to policies in Company B. Figure 7-26 and Figure 7-27 show that individual policies and team policies have all conformed to the best policy by generation 100.

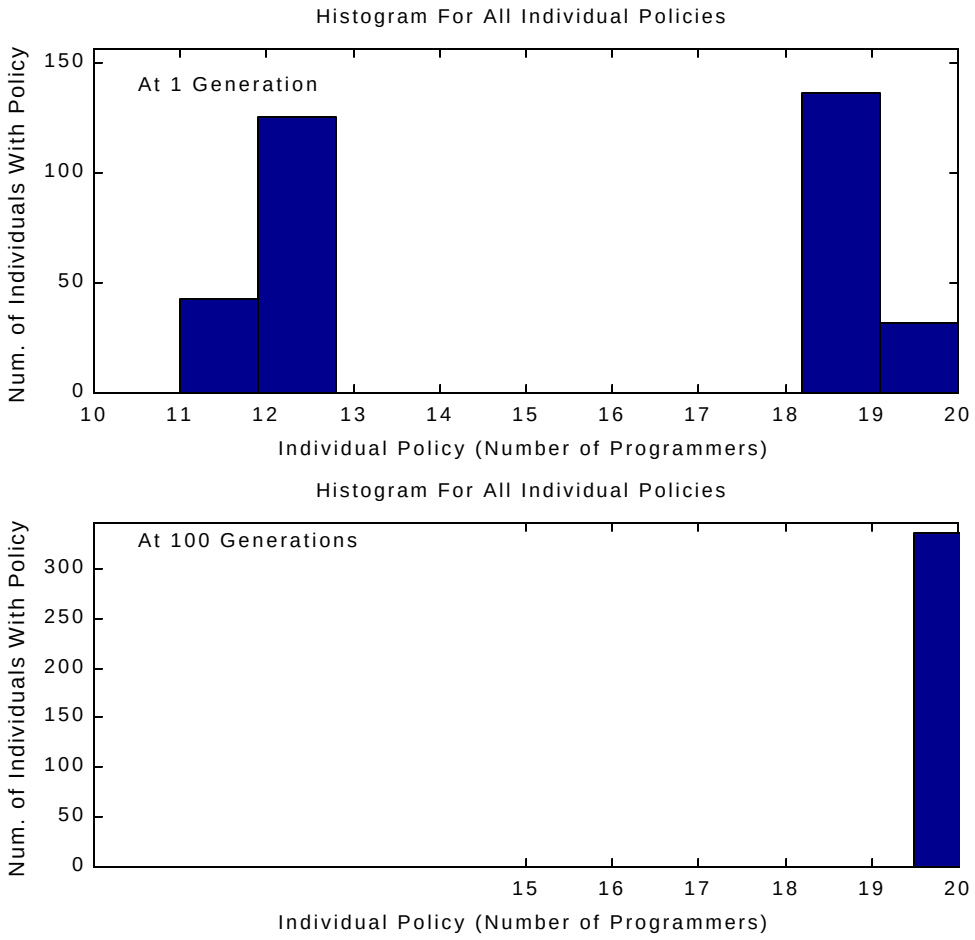

Figure 7-26. Looking at one company with a good policy and another with a poor policy at generation 1 and generation 100. Individual policy distribution (d_bias_pol).

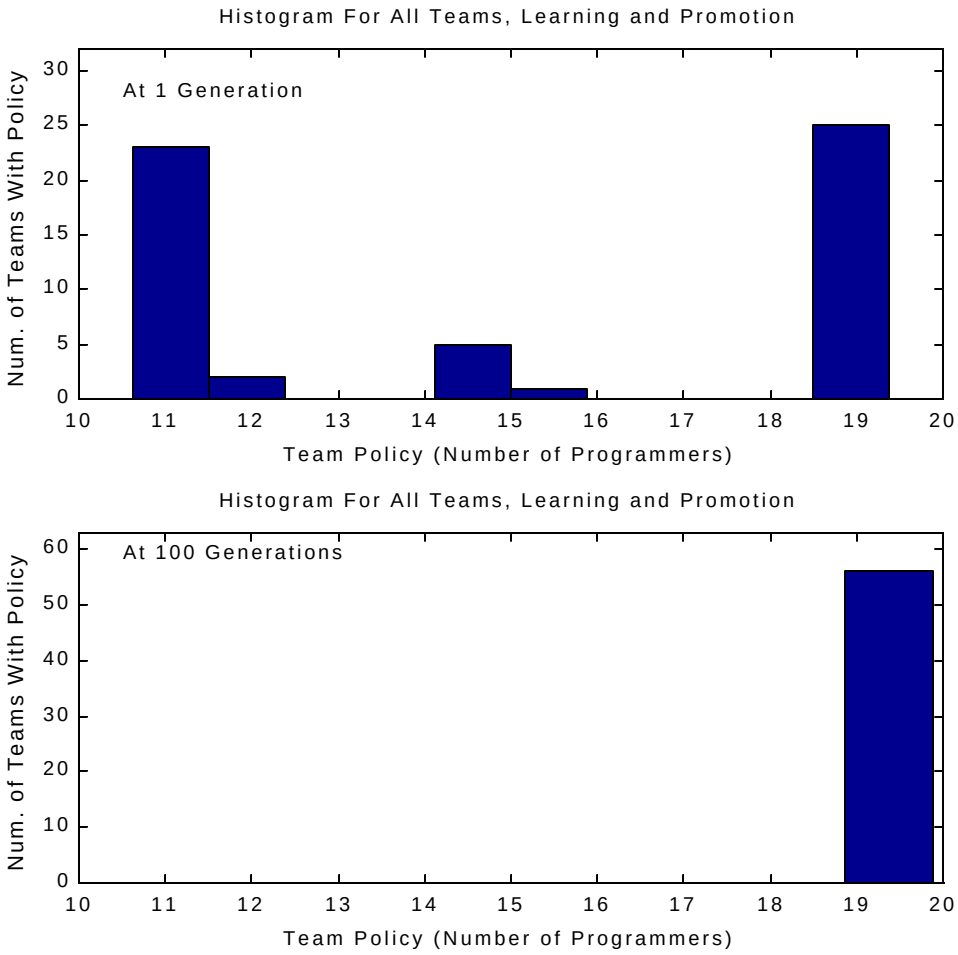

Figure 7-27. Team policy distribution at generation 1 and generation 100 (d_bias_pol).

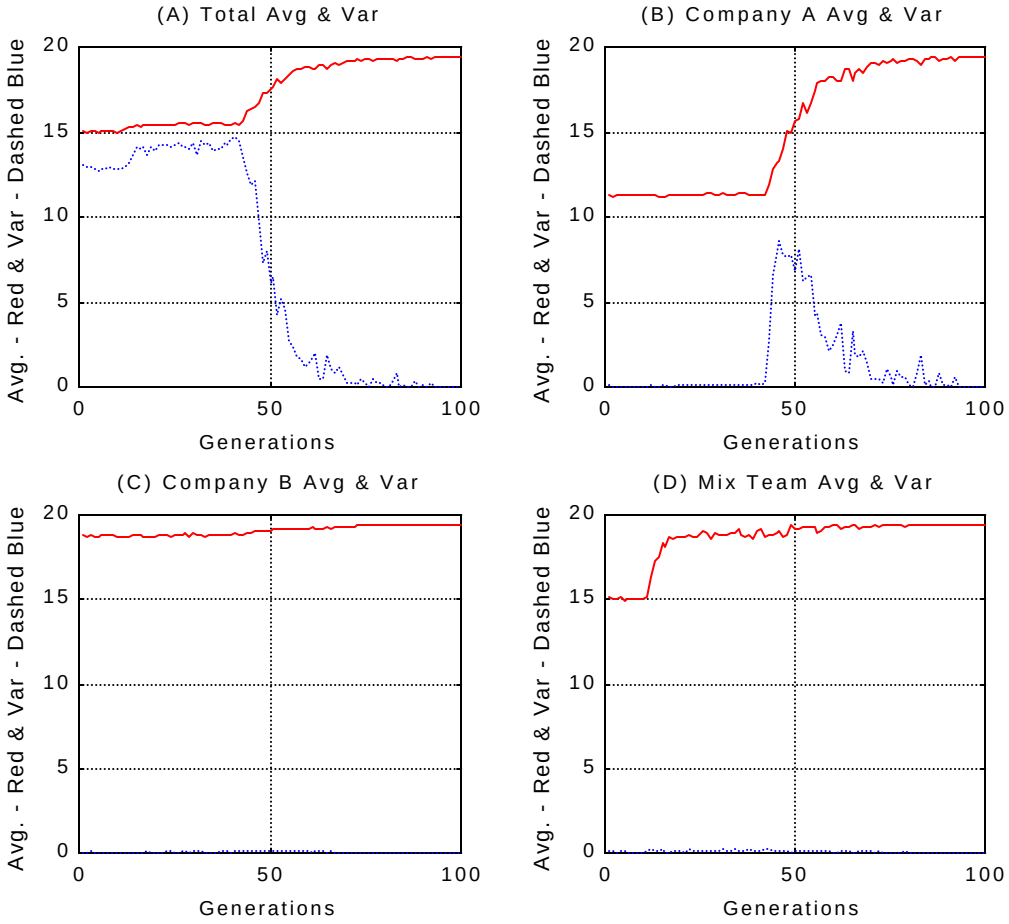

Figure 7-28. Evolution of the average team policy in a mixed team structure over 100 generations. Promotion is turned on after 10 generations and learning is turned on after 40 generations. (A) Whole company. (B) Legacy company A. (C) Legacy company B. (D) Mixed Teams (d_bias_pol).

8 Conclusion

As stated earlier, we believe that an organizational evolutionary approach to improve the transfer of knowledge between merging companies will result in a more successful merger, as a result, creating more value for the company and its shareholders. One way of implementing an evolutionary approach would be to fully mix individuals from both companies together. However, the use of a small number of mixed teams should be just as effective as fully mixing the individuals from two companies but at a slower rate.

To gain a better understanding of how knowledge is transferred between companies, simulations were performed to gain insight on how learning and promoting affects the transfer of policies in a mixed team structure. As we have found, the results from the mixed team structure indicate that it is as effective as fully mixing two companies together. Here is a summary of the results:

1. No Learning and No promoting – This is the baseline case. As expected, the company does not evolve at all. No learning takes place and everybody has the same status.

2. Promotion only – Here we look at the effects of status and promotions. Though status changes, policies do not. This will affect an individual's weight or influence on a team's policy. Teams that perform above average will be promoted, while teams that perform below average are demoted. Hence, individuals with good policies will have better status. When these individuals are assigned to the next project, they will be more influential on the team policy. This influence shifts the team policy closer to their policy. Individuals with poor policies lose status, thus they will be less influential on the team policy. With promotion only, the company evolves some but won't reach the optimal value. The optimal value is never reached, since this would require individuals to learn better policies. Though individuals do not change policies, good policies become more influential in team decisions than poor policies.

3. Learning only – Here we look at the effects of individuals randomly learning. This is not good, since the overall evolution of the company can evolve to a better state, a worse state, or remain in the same. Individuals change policies but with no direction of who to learn from. Depending upon the conditions, we may see consensus by individuals within the population.

4. Learning and promoting – Here we have status influencing how individuals learn new policies. Individuals with good policies will eventually gain higher status through promotion. When mixed onto the next project, individuals with good policies will have more influence over the team policy. This creates a better team policy. Also, individuals with poor policies will tend to learn from individuals with higher status, thus, improving their policy. Individuals with good policies can also

learn from other individuals, which as a result could create an even better policy. Eventually, the whole organization evolves in a positive direction, more so than just promoting alone. We also expect a faster convergence in this case due to two factors. First, we perform ranking on the merged company as a whole. When we do this, an individual's status is known in both companies, instead of their own company only. If we had ranked individuals against their respective company only, there is an initial ambiguity of not knowing which company performs better. Thus, there is the possibility that an individual could be highly ranked one company but average in the other, which could slow down evolution. Ranking the company as a whole eliminates this initial ambiguity. Second, promoting is turned on before learning. This allows individuals with good and poor policies to be identified before learning takes place. When learning does occur, individuals will have a better idea of who to learn from; as a result, speeding up convergence.

5. Good Policy/Poor Policy – In this experiment, we initialize the individuals at one company to have good policies, while the individuals at the other company to have poor policies. As we expected, the learning that occurred on the mixed teams allowed for the company with poor policies to evolve. When individuals with poor policies worked on a mixed team, they were able to learn from individuals with good policies. When these individuals returned to their companies, their higher status influenced other individuals to learn from them. Eventually, the company with poor policies evolved to the same level as the company with good policies.

From these experiments, we see that promoting provides direction for individuals to learn from and learning provides a way for these individuals to improve their policies. We can use the metaphor of a sailboat to describe the effects of learning and promoting. Say one's organization is a sailboat, and the goal of the sailboat is to reach a certain landmark. The sail on the boat is analogous to learning, while the rudder is analogous to promoting. If you could just control the rudder, the most you could do is point your boat toward the landmark. The sailboat is unable to reach the landmark without a force to move it. With just a sail, the boat can move around, but there is no method to guide the boat toward the landmark. With a sail and rudder, the sailboat is able to capture the wind and navigate toward the landmark.

From the analysis of the simulations, we were able to make suggestions on how to implement the mixed team structure for merging companies.

1. The mixed teams provide a conduit for passing information between organizations; thus, we would like the least amount of resistance in this mixed team structure. Individuals on the mixed team should be willing to learn from others as well as teach others.

2. If it makes sense, rank all projects and individuals together, not with respect to their own company only. This will help to identify individuals with good or bad policies for both companies.
3. Once individuals with good policies have been identified from both companies (from step 2), encourage others to learn from them. This is analogous to promoting first and then learning.
4. Individuals that are good performers and are highly respected within their own company should be used on the mixed teams. Chances are, these individuals will have something to teach other individuals on the mixed teams. The real benefit is when they return to their own companies. Since they are good performers and are well regarded by their peers, more individuals will likely try to learn from them or imitate them.
5. Mix individuals frequently on the mixed teams. This allows for more information to be diffused between companies.
6. Try to keep the mixed team project lengths short in nature; however don't make the projects trivial. Shorter projects will help speed up the mixing of individuals. Yet, the project needs to be of some significance. We want individuals to transfer hard to learn knowledge; however, this may not happen when teams work on simple projects. There may be another benefit for having short projects. Since some individuals do not wish to relocate, they may be willing to work at another site for a short period of time. One suggestion is to have a few mixed teams at each location. Another possibility is if the project is long, have the individuals switch sites in the middle of a project.

There are several benefits for using this mixed team structure for merging companies.

1. This mixed team structure is not as costly to implement as fully mixing two companies together. The merged company would only need to support the few individuals on the mixed teams that are changing locations.
2. Individuals are able to create a network of people to learn from, which would span over both companies. Though this was not investigated in this study, we believe that individuals that use this network will improve the transfer of knowledge and speed up the evolutionary process.
3. Companies are very dynamic and could change goals after a merger or when changes in the market occur. Policies that individuals had before these changes may not be as optimal after the changes. This evolutionary approach allows for individuals to evolve their policies when the goals of the company have changed.

If we look beyond the scope of mergers and acquisitions, it may be possible to apply the idea of using mixed teams to improve the transfer of knowledge in other areas. This concept may be a valid way to transfer information between organizational boundaries. For example, many large engineering firms have both an R&D group as well as a design engineering group. The R&D group develops new

technologies, while the design group develops products based on these technologies. Having mixed teams of R&D individuals with design individuals to work on R&D or design projects should improve how knowledge is transferred. Not only is it important to transfer R&D knowledge to design, but vice versa.

Analysis from the simulation has suggested that the mixed team approach is an effective method for improving knowledge transfer between two merging companies. Suggestions on how to implement the mixed team approach were also given. To further understand how evolution affects knowledge transfer within merging organizations, organizational evolutionists should further study various promotional policies, learning behavior, network of individuals that people learn from, and insights from simulation and how well they apply to the real world.

9 References

"After the Deal," *The Economist*, Jan. 7, 1999

Barker, Robert. "Cisco May Not Be Bouncing Back Soon," *Business Week*, Apr. 16, 2001, pp. 138.

Booth, Wayne C., Gregory G. Colomb, and Joseph M. Williams. The Craft of Research. Chicago: University of Chicago Press, 1995.

Colvin, Geoffrey. "The Year of the Mega Merger," *Fortune*, Jan. 11, 1999, pp. 62-64.

Goldberg, David E. Genetic Algorithms in Search, Optimization & Machine Learning. Massachusetts: Addison-Wesley, 1989.

Hines, James H. "Five Rules for Evolutionary Management." System Thinker, 1998.

Hines, James H. and House, Jody L. "Harnessing Evolution for Organizational Management," in InterJournal, International Conference on Complex Systems. 1998. Nashua, New Hampshire: NECSI.

Hines, James H. and House, Jody L. "The Source of Poor Policy: Controlling Learning Drift and Premature Consensus in Human Organizations," System Dynamics Review, Vol. 17, No. 1, Spring 2001.

House, Jody Lee, Alexander Kain, and James Hines. "Evolutionary Algorithm: Metaphor for Learning." In Genetic and Evolutionary Computation Conference. 2000. Las Vegas, NV: Morgan Kaufmann.

Kelly, John, et al. "Unlocking Shareholder Value: the Keys to Success," KPMG White Paper, 1999, pp. 7.

Leon-Garcia, Alberto. Probability and Random Processes for Electrical Engineering. Massachusetts: Addison-Wesley, 1994.

Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge, MA: The MIT Press, 1999.

Schein, Edgar H. The Corporate Culture Survival Guide. San Francisco: Jossey-Bass Publishers, 1999.

Spitzer, Don, et al. "The New Art of the Deal: How Leading Organizations Realize Value From Transactions," KPMG White Paper, 1999, pp. 3.

Walpole, Ronald E, and Raymond H. Myers. Probability and Statistics for Engineers and Scientists. New York: Macmillan Publishing Company, 1989.

10 Appendix

10.1 Further Readings in Organizational Evolution

1. Hines, J. and J. House. Harnessing Evolution for Organizational Management. in OACES 98. 1997. England.

2. Hines, J. and J. House. Harnessing Evolution for Organizational Management. in InterJournal, International Conference on Complex Systems. 1998. Nashua, New Hampshire: NECSI.

3. Hines, J. and J. House. Team promotion to foster policy evolution. in International Conference on Complex Systems. 1998. Nashua, New Hampshire: original submission to NECSI in 98.

4. Hines, J. and J. House, Policy Evolution within an Organization: Research Proposal, 1998, NSF IOC 98.

5. Hines, J. and J. House. Where policies come from. in International System Dynamics Coneference. 1998. New Zealand.

6. Hines, J. and J. House, Baskin-Robbins vs. Darwin. Harvard Business Review (hopefully), 1999.

7. Hines, J.H. and J.L. House. Policy Evolution within an Organization. in 2000 Design and Manufacturing Conference. 1999. Vancouver, B.C.

8. Hines, J.H. and J.L. House. Management fads & poor policies: A year of organization evolution research. in 2001 NSF Design and Manufacturing Research Conference. 2001. Tampa, FL.

9. House, J., GA coupled to continuous time feedback model, . 1999.

10. House, J., Organizationl Evolution Simulator: Matlab, . 1999.

10.2 Matlab Code

This section contains the main sections of the Matlab code.

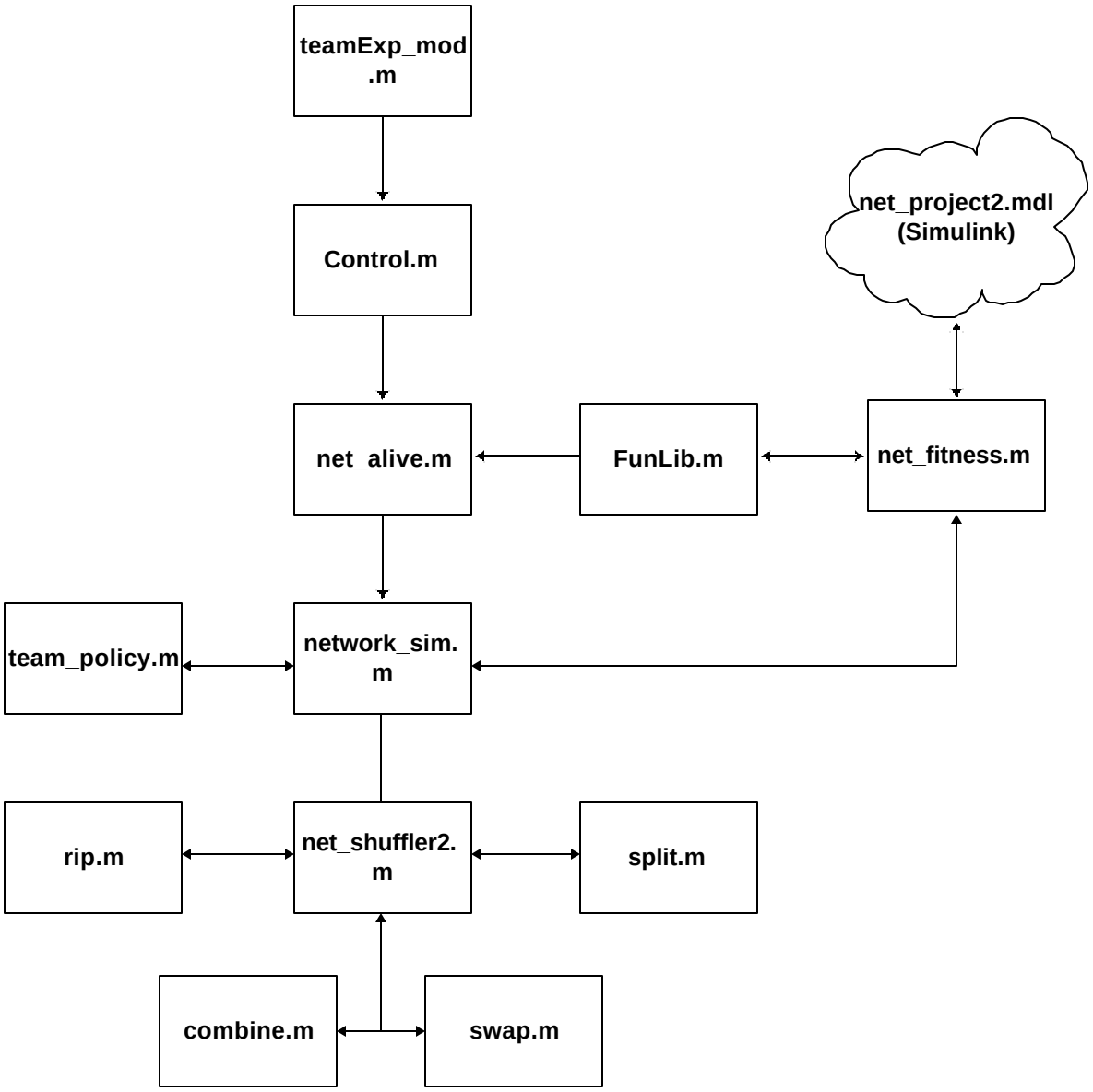

Figure 10-1. Matlab code modules and their relationship.

10.2.1 TeamExp_mod.m

```
function E=teamExp_mod()


% load function library
funlib


% Format: fun, mode, teams, gen, pop, pc, pm, init_switch

% Sub-Team Format:  funct, mode, num subteams, subteam size, teams in company
A/B,
%     generations, network size, p-xover, pmutation, pnetwork, init_switch

E={ ...
      % control

   % Single company simulation
   %  {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...
   %   {projectMdl_st,'st', 6, 6, 50, 50, 5, 1, 0, .5, 1} ...


   % Merging company simulation
%      {projectMdl_st,'st', 3, 4, 5, 2, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...
%      {projectMdl_st,'st', 6, 6, 25, 100, 5, 1, 0, .5, 1} ...


};
```

10.2.2 Control.m

```
% control center

% load experiment
%E=gecco2000; % Format: fun, mode, teams, pop, pc, pm
%E=GECCOtest;
%E=teamAssignExp;


clear;
pack;

% * * * * * * * * * * * * * * * * * * * *
% cdat = 1 - collect data
% cdat = 0 - don't collect data
cdat = 0;
% * * * * * * * * * * * * * * * * * * * *


   % **********   Sets the Single or Merging company mode.   **********
   % Mode         Desc.
   % ----         -----
   %  1           Single company mode.
   %
   %  2           Merging compnay mode.
   company = 2;
   % ********************************************


%E=teamExp;
E=teamExp_mod;

% Gives the number of cells (rows) in E.  Each cell contains multiple
attributes.
len_E = length(E);

% Control Variables
%slength = 1;  % Made this a variable, which is set here.

%action='create'; % create, stat, collect
ver='2';
% perform actions
for cntra = 1:len_E
   %   j=char(112+i);
%  j=char(cntra);
%  switch action
%  case 'create'
%     for seed=1:slength  % Call alive_mod - Sends all of the control
parameters

%        switch lower(E{cntra}{2})
%        case ('st')
%           disp('This is the networked version')
```

```
           disp(lower(E{cntra}{2}))

%           n =
net_alive(E{i}{1},E{i}{2},E{i}{3},E{i}{4},E{i}{5},E{i}{6},E{i}{7},E{i}{8},E{i
}{9},...
%              E{i}{10}, E{i}{11}, seed);

            % Forcing into the random state
            seed = -1;

net_alive;

if cdat == 1
            ravg(cntra).lavg = mean(avg_teamp(1:10));
            ravg(cntra).havg = mean(avg_teamp(25:experiment.generation_max));
            ravg(cntra).mixsize = E{cntra}{3};
            ravg(cntra).puresize = E{cntra}{5};
            ravg(cntra).percent = abs((ravg(cntra).lavg-
ravg(cntra).havg)/ravg(cntra).lavg)*100;
            switch company
            case 1
               save (['s_l_r' num2str(cntra) 'm' num2str(E{cntra}{3}) 'p'
num2str(E{cntra}{5}) 'u_4_11']);

            case 2
                  save (['d_lp_r' num2str(cntra) 'm' num2str(E{cntra}{3}) 'p'
num2str(E{cntra}{5}) 'u_4_12']);
            otherwise
                  disp('Error in writing selection')
            end


            %close(gcf)
            %close(gcf)
            %close(gcf)
            disp('End of Run: ')
            cntra
end

            % save(['nv' ver '_e' num2str(i)  '_s' num2str(seed)],'e');
%        otherwise
%          disp('Standard Case')
          %
e=alive(E{i}{1},E{i}{2},E{i}{3},E{i}{4},E{i}{5},E{i}{6},E{i}{7},E{i}{8},seed)
;
          % save(['v' ver '_e' num2str(i)  '_s' num2str(seed)],'e');
%        end
   end

%  case 'stat'
%     for seed=1:slength
%        i
%      load(['v' ver '_e' num2str(i) '_s' num2str(seed)]);
%      e=stat(e);
%      e=stat2(e);
%      save(['v' ver '_e' num2str(i) '_s' num2str(seed)],'e');
%    end
```

```
%  case 'collect'
    % load desired stats variables into memory for easy plotting
%    for seed=1:slength
%       load(['v' ver '_e' num2str(i) '_s' num2str(seed)]);
%      F(i,seed)=e.stat;
%    end

%  end
%end



if cdat == 1

disp('* * * Simulation Complete * * *');

 save apr12_d_lp ravg;
end

%clear len_E, slength;
%pack;
```

10.2.3 Net_alive.m

```
%function experiment=net_alive(fun, mode, sub_teams, team_size,
corp_teama,...
%   gen, net_size, pc, pm, pn, init, seed)

% This function includes the network size for each individual.

% fitness function def



%  These parameters comes from the file funlib.m
experiment.domain=E{1}{1}.d; % function defined over this domain
experiment.epsilon=E{1}{1}.e; % simulation accuracy
experiment.fitness=E{1}{1}.f; % fitness


% experiment mode
experiment.mode=E{1}{2}; % st - Network and sub-teams
experiment.sub_teams=E{1}{3}; % number of sub-teams
experiment.sub_team_size=E{1}{4}; % size of the teams
experiment.corp_teama = E{1}{5}; % Number of teams in company A
experiment.corp_teamb = E{1}{5}; % Number of teams in company B.  Equal to A
for now.
experiment.teammode='random'; % for tm mode: random/best/equal

% other
experiment.init=E{1}{11};
experiment.generation_max=E{1}{6}; % number of generations to simulate
experiment.net_size = E{1}{7};  % sets an individual's network size

% disp('**************** Network size *********************')
% disp(experiment.net_size)

experiment.p_crossover=E{1}{8}; % probability of crossover (or learning)
experiment.p_mutation=E{1}{9}; % probability of mutation
experiment.p_network=E{1}{10}; % probability of learning from a network.

% experiment label
experiment.label='ALIVE Genetic Simulator';
experiment.seed=seed; % -1 for "truly random", nonnegative number is seed for
"deterministic random"


% Generating fitness function
F=['function [y, track]=fitness(x, generation)\n' ...
    '%% This file is created automatically, manual edits will be lost\n' ...
    'track=[];\n' ...
    experiment.fitness ';\n' ...
    %'if y<0, error(''Negative fitness not allowed''), end' ... % guard
against negative fitness
];
fid=fopen('fitness.m','w');
fprintf(fid,F,'%s');
```

```
fclose(fid);
clear fitness


% Checking

%This checks to see if the members per team is even.
if rem(experiment.sub_team_size,2)
   error('Number of individuals per team must be even')
end


% Echo
fprintf('Simulation: %s\n\n',experiment.label');
fprintf('Mode: %s\n',experiment.mode');
fprintf('Number of generations to simulate: %i\n',experiment.generation_max);
fprintf('Total number of individuals in simulation: %i\n',...

2*((experiment.sub_teams/2+experiment.corp_teama)*experiment.sub_team_size));
fprintf('Number of individuals per company: %i\n',...
   (experiment.sub_teams/2+experiment.corp_teama)*experiment.sub_team_size);
fprintf('Number of subteams: %i\n',experiment.sub_teams);
fprintf('Number of individuals per subteam: %i\n',experiment.sub_team_size);
fprintf('Number of teams per company: %i\n',experiment.corp_teama);
fprintf('Team mode: %s\n',experiment.teammode);
fprintf('Network size per individual: %i\n',experiment.net_size);
fprintf('Probability of crossover: %f\n',experiment.p_crossover);
fprintf('Probability of using network: %f\n',experiment.p_network);
fprintf('Probability of mutation: %f\n',experiment.p_mutation);
fprintf('Domain: \n'); disp(experiment.domain);
fprintf('Epsilons: \n'); disp(experiment.epsilon');


% Forcing into the random state
% experiment.seed = -1;

% Randomness
if experiment.seed<0
  experiment.seed=sum(100*clock); % "truly random"
end
rand('state',experiment.seed);



% do it

switch company
case 1
   disp('Performing a Single Company Simulation')
   network_sim1;
case 2
   disp('Performing a Merged Company Simulation')
   network_sim;
otherwise
   error('>>> This is not a valid company simulation case <<<')
end
```

10.2.4 FunLib.m

```
% funlib - a place to store often used fitness functions
% in "def" structure: domain, epsilon, function

disp('Loading function librabry')

peak_norm=struct('d',[-3 3; -3 3],'e',[.1 .1],'f','y =
abs(peaks(x(1),x(2)))');
peak_same=struct('d',[-10 10; -10 10],'e',[.5 .5],'f','y=x(2);x=x(1);y=exp(-
(x-6).^2-(y-6).^2)+exp(-(x-6).^2-(y-0).^2)+exp(-(x-6).^2-(y+6).^2)+exp(-
(x+6).^2-(y-6).^2)+exp(-(x+6).^2-(y-0).^2)+exp(-(x+6).^2-(y+6).^2)');


DeJong(1)=struct('d',[-5.12 5.12; -5.12 5.12],'e',[.1 .1],'f','y=x(2);x=x(1);
y =50-(x.^2+y.^2)');
DeJong(2)=struct('d',repmat([-2.048 2.048],2,1),'e',repmat(.1,2,1),'f','y =
100*(x(1)^2-x(2))^2+(1-x(1))^2');
DeJong(3)=struct('d',repmat([-5.12 5.12],5,1),'e',repmat(.1,5,1),'f','y =
sum(floor(abs(x)))');
DeJong(4)=struct('d',repmat([-1.28 1.28],30,1),'e',repmat(.1,30,1),'f','y=0;
for i=1:30, y=y+i*x(i)^4+randn(1); end');
DeJong(5)=struct('d',repmat([-65.536
65.536],2,1),'e',repmat(.001,2,1),'f','y=0; incomplete');

ProjectMdl=struct('d',[0.25 10; 0.25 10],'e',[1
1],'f','y=projectSim(x(1),x(2))');
projectMdl2=struct('d',[0 15],'e',[1],'f','y=projectSim2(x(1))');
ProjectMdl3=struct('d',[0 15 0 15],'e',[1
1],'f','y=projectSim3(x(1),generation)');

% ********************************************************************
% This is the fitness function that I'm using right now - R.R.
projectMdl_st=struct('d',[10 20],'e',[1],'f','y=projectSim2(x(1))');
% ********************************************************************

MrktMdl=struct('d',[0.1 1; 100 5000],'e',[0.1
100],'f','y=mrktSim(x(1),x(2))');
wfSalesMdl=struct('d',[0.1 1; 0.1 1],'e',[0.1 0.1],'f','y=wfSim(x(1),x(2))');
```

10.2.5 Net_fitness.m

```
function [finalTime]= net_fitness(programmers)

%This function determines the final time for the project model
%simulation in simulink. 7/26/99
%In the genetic algorithm -- the final time is used as the measure
%of fitness
simTime=80;  % Is this how long it's suppose to run for - rro?

lb = '[';
rb = ']';
prgms = [lb num2str(programmers) rb];

set_param('net_project2/Programmers','Value', prgms);

%set_param('net_project2/Programmers','Value', num2str(programmers));



%run project model.
[T,X,Y] = sim('net_project2',simTime);  % Run the simulation for simTime and
collect state - rro
done= (FinalTime);


finalTimeMatrix=[T,done];

[y, indx] = max(done);

for j = 1:length(indx)
   u(j,:) = [T(indx(j)) simTime-T(indx(j))];
end

% The better the performance the larger the number
finalTime = round(u(:,2));

%%find out when the amount left of code to write falls to zero
%%codeToWrite=X(:,1);
%i=1;
%while i<(length(T))
%   if finalTimeMatrix(i,2)==1
%      finalTime=finalTimeMatrix(i,1);
%      i=length(T)+10;
%   else
%      i=i+10;
%      finalTime=simTime;
%   end
%end
%finalTime=simTime-finalTime;
%finalTime;
```

10.2.6 Network_sim.m – Core of Mixed Team Company Simulator

```
% Raymond Ro, 2001
% Multiple company simulator


   disp('*************** Merged Company Simulator ******************')


   % ************************************************

   % **********   Sets the promotion mode.   **********
   % Mode         Desc.
   % ----         -----
   %  1           Teams are evaluated against other teams in their respective
companies.
   %              Mixed teams (subteams) are evaluated against each other
only.
   %
   %  2           Teams from popa, popb, and the subteams are all evaluated
together.
   %
   %  3           No promotional value is used (no weighting); however, teams
are sorted
   %              within their companies only.  This is only effects how the
data is kept.
   %
   %  4           No promotional value is used (no weighting); however, all
teams are sorted
   %              together, regardless of company.  This is only effects how
the data is kept.
   %
   %  5           Weighting is turned on after a specified number of
generations.  The variable
   %              step is used to specify the number of generations w/o
weighting.  Mode 1 is
   %              used for the type of weighting.
   %
   %  6           Weighting is turned on after a specified number of
generations.  The variable
   %              step is used to specify the number of generations w/o
weighting.  Mode 2 is
   %              used for the type of weighting.
   %


   prom_mode = 6;
   step = 10;

   % ************************************************



   % **********   Sets the Learning mode.   **********
   % Mode         Desc.
```

```
   % ----          -----
   %  1            No learning occurs.
   %
   %  2            Learning in all teams occur.
   %
   %  3            Only learning in mixed teams occur.
   %
   %  4            Learning in ALL teams occur after a certain time.
   %               Variable stepl is used to indicate when learning is turned
on.
   %
   %  5            Learning in MIXED teams occur after a certain time.
   %               Variable stepl is used to indicate when learning is turned
on.
   %
   %  6            Learning in all teams occur along with using a network.
   %
   %  7            Learning in all teams occur along with using a network.
This will be
   %               turned on after a specified time.

   learning = 4;
   stepl = 40;
   % ************************************************

   % ************************************************

   % **********   Sets the Histogram movie mode.   **********
   % Mode         Desc.
   % ----         -----
   %  0           No movie.
   %
   %  1           Movieswitch is set to 1 and a movie is recorded.

   movieswitch = 1;
   % ************************************************



% ********** Dimensional Resolution **********

% This section calculates the number of bits needed for each gene in the
chromosome.
locus=zeros(size(experiment.domain,1),3);
for dim=1:size(locus,1) % 1 to length of locus (rows)
   locus(dim,3)=ceil(log2(abs(experiment.domain(dim,1)-
experiment.domain(dim,2))/experiment.epsilon(dim)));
end

% Now we start to build the String loci definition
%  (Starting bit | Stopping bit | Segment length) for each gene
locus(1,1)=1;
locus(1,2)=locus(1,3);
for dim=2:size(locus,1)
   locus(dim,1)=locus(dim-1,2)+1;
   locus(dim,2)=locus(dim,1)+locus(dim,3)-1;
end
```

```
fprintf('String loci (left, right, and lengths) per dimension:\n');
disp(locus);

timer=cputime; % start


% Initialize populations

generation = 1;

chrom_len = locus(end,2);
comp_size = experiment.corp_teama*experiment.sub_team_size;




   % **********   Sets the initial population policy distribution.
**********
   % Mode         Desc.
   % ----         -----
   %  1           Creates policies based on a uniform distribution over the
whole
   %              dynamic range.
   %
   %  2           Creates polices based on two uniform distributions.
Company 'a' will
   %              have a mean of u1 and 'b' will have a mean of u2. Margin
specifies
   %              how wide the distribution should spread around the means.


   distribution = 2;
   base = 2;
   drange = base^chrom_len;
   u1 = 2;
   u2 = drange-2;
   margin = 1;



% Index (Generations)

% This allocates memory ahead of time for the various population attributes
for generation=1:experiment.generation_max+1
   popa(generation).chromosome = zeros(comp_size,chrom_len);
   popa(generation).team_id = zeros(comp_size,1);
   popa(generation).id = zeros(comp_size,1);
   popa(generation).rank = zeros(comp_size,1);
   popa(generation).network = zeros(comp_size,experiment.net_size);
   % which pop a individual in the network is from  1 - other population
   % Descibes the company an individual is from
   popa(generation).net_team = zeros(comp_size,experiment.net_size);
   popa(generation).team_policy = zeros(experiment.corp_teama,chrom_len);
   popa(generation).team_ranking = zeros(experiment.corp_teama,1);
   % For Debugging purposes.
   popa(generation).xover = zeros(comp_size,1);
```

```
   popb(generation).chromosome = zeros(comp_size,chrom_len);
   popb(generation).team_id = zeros(comp_size,1);
   popb(generation).id = zeros(comp_size,1);
   popb(generation).rank = zeros(comp_size,1);
   popb(generation).network = zeros(comp_size,experiment.net_size);
   % which pop a individual in the network is from.  1 - other population
   popb(generation).net_team = zeros(comp_size,experiment.net_size);
   popb(generation).team_policy = zeros(experiment.corp_teama,chrom_len);
   popb(generation).team_ranking = zeros(experiment.corp_teamb,1);
   % For Debugging purposes.
   popb(generation).xover = zeros(comp_size,1);
end



%  Create an initial value for each individual
%popa(1).chromosome = ceil(rand(comp_size,locus(end,2))*2)-1;
%popb(1).chromosome = ceil(rand(comp_size,locus(end,2))*2)-1;

popa(1).rank = ones(comp_size,1);
popb(1).rank = ones(comp_size,1);

%  Assign ID's to all individuals
popa(1).id(1:comp_size) = 1:comp_size;
popb(1).id(1:comp_size) = 1:comp_size;


%  Assign each member to a team

%  This is just picking a batch of individuals and then assigning them to a
team
for team = 1:experiment.corp_teama
   popa(1).team_id(experiment.sub_team_size*team-
experiment.sub_team_size+1:team*experiment.sub_team_size)=team*...
      ones(experiment.sub_team_size,1);
   popb(1).team_id(experiment.sub_team_size*team-
experiment.sub_team_size+1:team*experiment.sub_team_size)=team*...
      ones(experiment.sub_team_size,1);
end



% Build sub-teams now

% Number of individuals in the subteams
sub_size = experiment.sub_teams*experiment.sub_team_size;



% This allocates memory ahead of time for the various population attributes
for generation=1:experiment.generation_max+1
   subteam(generation).chromosome = zeros(sub_size,chrom_len);
   subteam(generation).team_id = zeros(sub_size,1);
   subteam(generation).id = zeros(sub_size,1);
   subteam(generation).rank = zeros(sub_size,1);
```

```
   subteam(generation).network = zeros(sub_size,experiment.net_size);
   % which pop a individual in the network is from.  1 - other population
   subteam(generation).net_team = zeros(sub_size,experiment.net_size);
   subteam(generation).corp_id = zeros(sub_size,1);
   subteam(generation).team_policy = zeros(experiment.sub_teams,chrom_len);
   subteam(generation).team_ranking = zeros(experiment.sub_teams,1);
   % For Debugging purposes.
   subteam(generation).xover = zeros(sub_size,1);
end

%  Create an initial value for each individual
%subteam(1).chromosome = ceil(rand(sub_size,locus(end,2))*2)-1;

% Initialize everybody's rank to 1.
subteam(1).rank = ones(sub_size,1);


%  Assign ID's to all individuals
cntr = 1;
for i = 1:2:sub_size
   subteam(1).id(i) = cntr + comp_size;
   subteam(1).id(i+1) = cntr + comp_size;
   cntr = cntr + 1;
end
clear cntr;

%  This is just picking a batch of individuals and then assigning them to a
subteam
for team = 1:experiment.sub_teams

   subteam(1).team_id(experiment.sub_team_size*team-
experiment.sub_team_size+1:team*experiment.sub_team_size)=team*...
      ones(experiment.sub_team_size,1);

end


for team = 1:sub_size

   % Take team and divide by 2 and assign by vectors
   % Identifying which company an individual came from.
   if mod(team,2)
      subteam(1).corp_id(team) = 'a';
   else
      subteam(1).corp_id(team) = 'b';
   end

end


%  Create an initial value for each individual

switch distribution
case 1
   disp('Using a uniform distribution to create policies')
   popa(1).chromosome = ceil(rand(comp_size,locus(end,2))*2)-1;
   popb(1).chromosome = ceil(rand(comp_size,locus(end,2))*2)-1;
```

```
   subteam(1).chromosome = ceil(rand(sub_size,locus(end,2))*2)-1;
case 2
   disp('Using a uniform distribution to create policies around 2 means')
   [popa(1).chromosome, popb(1).chromosome, subteam(1).chromosome] = ...
      bias_policy(popa(1).chromosome, popb(1).chromosome,
subteam(1).chromosome,...
      u1, u2, drange, margin);
otherwise
   error('>>> This is not a valid initialization case <<<')
end



% ************************ This is the main loop ************************
% experiment.generation_max = 3;

% Creates a handle for the figure

switch movieswitch
   case 0

   case 1
      movie_fig = figure;
   otherwise
      error('>>> This is not a valid movie option <<<')

end

for gen = 1:experiment.generation_max  % Need to expand this for the total
number of generations


% ******************* Create team policies *******************
% This computes the team policy for the subteams (mixed teams)

for i = 1:experiment.sub_team_size:sub_size

subteam(gen).team_policy((i+experiment.sub_team_size-
1)/experiment.sub_team_size,:) = ...
   team_policy(subteam(gen).chromosome(i:i+experiment.sub_team_size-
1,1:chrom_len),...
   subteam(gen).rank(i:i+experiment.sub_team_size-1))-48;
end


%disp('Return from Sub-routine')
%subteam(1).team_policy

% This computes the team policy for both corporations
for i = 1:experiment.sub_team_size:comp_size

popa(gen).team_policy((i+experiment.sub_team_size-
1)/experiment.sub_team_size,:) = ...
   team_policy(popa(gen).chromosome(i:i+experiment.sub_team_size-
1,1:chrom_len),...
   popa(gen).rank(i:i+experiment.sub_team_size-1))-48;
```

```
popb(gen).team_policy((i+experiment.sub_team_size-
1)/experiment.sub_team_size,:) = ...
   team_policy(popb(gen).chromosome(i:i+experiment.sub_team_size-
1,1:chrom_len),...
   popb(gen).rank(i:i+experiment.sub_team_size-1))-48;
end


% ****************** Evaluate Team Fitnesses ******************

% Convert policies to actual numbers
% Contains team policies in this order: popa, popb, & subteam
%policy_vec = [bin2dec(char(popa(1).team_policy+48));...
%      bin2dec(char(popb(1).team_policy+48));...
%     bin2dec(char(subteam(1).team_policy+48))]';
%policy_vec = (policy_vec/2^(chrom_len));
%policy_vec = policy_vec*diff(experiment.domain) + experiment.domain(1);


policy_veca = [bin2dec(char(popa(gen).team_policy+48))]';
policy_vecb = [bin2dec(char(popb(gen).team_policy+48))]';
policy_vecs = [bin2dec(char(subteam(gen).team_policy+48))]';

policy_veca = (policy_veca/base^(chrom_len));
policy_vecb = (policy_vecb/base^(chrom_len));
policy_vecs = (policy_vecs/base^(chrom_len));

policy_veca = policy_veca*diff(experiment.domain) + experiment.domain(1);
policy_vecb = policy_vecb*diff(experiment.domain) + experiment.domain(1);
policy_vecs = policy_vecs*diff(experiment.domain) + experiment.domain(1);

% This is to capture the histogram
tpolicy = [policy_veca policy_vecb policy_vecs];
 % subplot(211)

 tpol(gen,:) = tpolicy;
 avg_teamp(gen) = mean(tpolicy);
 var_teamp(gen) = var(tpolicy);

 avga(gen) = mean(policy_veca);
 vara(gen) = var(policy_veca);

 avgb(gen) = mean(policy_vecb);
 varb(gen) = var(policy_vecb);

 avgs(gen) = mean(policy_vecs);
 vars(gen) = var(policy_vecs);


 switch movieswitch
 case 0

 case 1

   % Creating a movie of the histogram
   [n(gen,:), bincen(gen,:)] = hist(tpolicy);
```

```
   %subplot(221)
   hist(tpolicy);
   title('Histogram For All Teams')
   axis([experiment.domain(1) experiment.domain(2) 0 max(n(gen,:))+2])
   xlabel('Team Policy (Number of Programmers)')
   ylabel('Number of Teams With Policy')
   grid;

   figure(movie_fig);
   m(gen) = getframe;

   %subplot(222)
   %hist(policy_veca);
   %title('Corp A Teams')
   %grid;

   %ma(gen) = getframe;

   %subplot(223)
   %hist(policy_vecb);
   %title('Corp B Teams')
   %grid;

   %mb(gen) = getframe;

   %subplot(224)
   %hist(policy_vecs);
   %title('Mixed Teams')
   %grid;

   %ms(gen) = getframe;

   % subplot(212)
   % hist(policy_vec)

   otherwise
      error('>>> This is not a valid movie option <<<')

end



% This returns the fitness of each team policy
fit_a = net_fitness(policy_veca);
fit_b = net_fitness(policy_vecb);
fit_s = net_fitness(policy_vecs);

% *** This is used to turn on ranking ***
%if gen <= 20
%  % Use net_promotion2 for non-promotion
%  [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);
%  [prom_valb, popb(gen).team_ranking] = net_promotion2(fit_b);
%  [prom_vals, subteam(gen).team_ranking] = net_promotion2(fit_s);
%else
%  % Use net_promotion to allow for promotion of individuals
%  [prom_vala, popa(gen).team_ranking] = net_promotion(fit_a);
%  [prom_valb, popb(gen).team_ranking] = net_promotion(fit_b);
```

```
%  [prom_vals, subteam(gen).team_ranking] = net_promotion(fit_s);
%end

% *** Promotion is calculated here ***

switch prom_mode
   case 1
      disp('Case 1 - Separate team evaluation between corporations');
      [prom_vala, popa(gen).team_ranking] = net_promotion(fit_a);
      [prom_valb, popb(gen).team_ranking] = net_promotion(fit_b);
      [prom_vals, subteam(gen).team_ranking] = net_promotion(fit_s);

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;

   case 2
      disp('Case 2 - All teams are evaluated together');
      alength = length(fit_a);
      blength = length(fit_b);
      slength = length(fit_s);
      fit_all = [fit_a; fit_b; fit_s];
      [prom_val, team_rankings] = net_promotion(fit_all);

      prom_vala = prom_val(1:alength);
      prom_valb = prom_val(alength+1:alength+blength);
      prom_vals = prom_val(alength+blength+1:alength+blength+slength);

      popa(gen).team_ranking = team_rankings(1:alength);
      popb(gen).team_ranking = team_rankings(alength+1:blength);
      subteam(gen).team_ranking = team_rankings(...
         alength+blength+1:alength+blength+slength);

      %team_rankings
      %prom_vala'
      %prom_valb'
      %prom_vals'
      %prom_val'
      %pause;

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;

   case 3
      disp('Case 3 - Separate team evaluation between corporations but no
promotion');
      [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);
      [prom_valb, popb(gen).team_ranking] = net_promotion2(fit_b);
      [prom_vals, subteam(gen).team_ranking] = net_promotion2(fit_s);

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;
```

```
   case 4

      disp('Case 4 - All teams evaluated together but no promotion');
      alength = length(fit_a);
      blength = length(fit_b);
      slength = length(fit_s);
      fit_all = [fit_a; fit_b; fit_s];
      [prom_val, team_rankings] = net_promotion2(fit_all);

      prom_vala = prom_val(1:alength);
      prom_valb = prom_val(alength+1:alength+blength);
      prom_vals = prom_val(alength+blength+1:alength+blength+slength);

      popa(gen).team_ranking = team_rankings(1:alength);
      popb(gen).team_ranking = team_rankings(alength+1:blength);
      subteam(gen).team_ranking = team_rankings(...
         alength+blength+1:alength+blength+slength);

      %team_rankings
      %prom_vala'
      %prom_valb'
      %prom_vals'
      %prom_val'
      %pause;

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;

   case 5

      % *** This is used to turn on ranking.  Companies are ranked
separately. ***

      disp('Case 5 - Promotion is turned on with a delay.  Companies are
ranked separately');
      if gen <= step
         % Use net_promotion2 for non-promotion
         [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);
         [prom_valb, popb(gen).team_ranking] = net_promotion2(fit_b);
         [prom_vals, subteam(gen).team_ranking] = net_promotion2(fit_s);
      else
         disp('Promotion is enabled')
         % Use net_promotion to allow for promotion of individuals
         [prom_vala, popa(gen).team_ranking] = net_promotion(fit_a);
         [prom_valb, popb(gen).team_ranking] = net_promotion(fit_b);
         [prom_vals, subteam(gen).team_ranking] = net_promotion(fit_s);
      end

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;

   case 6
```

```
      % *** This is used to turn on ranking.  Companies are ranked together.
***

      disp('Case 6 - Promotion is turned on with a delay.  Companies are
ranked together');
      if gen <= step

         % Use net_promotion2 for non-promotion
         alength = length(fit_a);
         blength = length(fit_b);
         slength = length(fit_s);

         fit_all = [fit_a; fit_b; fit_s];

         [prom_val, team_rankings] = net_promotion2(fit_all);

         prom_vala = prom_val(1:alength);
         prom_valb = prom_val(alength+1:alength+blength);
         prom_vals = prom_val(alength+blength+1:alength+blength+slength);
         popa(gen).team_ranking = team_rankings(1:alength);

         popb(gen).team_ranking = team_rankings(alength+1:blength);
         subteam(gen).team_ranking = team_rankings(...
            alength+blength+1:alength+blength+slength);

      else
         disp('Promotion is enabled')
         % Use net_promotion to allow for promotion of individuals
         alength = length(fit_a);
         blength = length(fit_b);
         slength = length(fit_s);

         fit_all = [fit_a; fit_b; fit_s];

         [prom_val, team_rankings] = net_promotion(fit_all);

         prom_vala = prom_val(1:alength);
         prom_valb = prom_val(alength+1:alength+blength);
         prom_vals = prom_val(alength+blength+1:alength+blength+slength);
         popa(gen).team_ranking = team_rankings(1:alength);

         popb(gen).team_ranking = team_rankings(alength+1:blength);
         subteam(gen).team_ranking = team_rankings(...
            alength+blength+1:alength+blength+slength);
      end

      % These variables are for monitoring purposes
      m_vala(:,gen) = prom_vala;
      m_valb(:,gen) = prom_valb;
      m_vals(:,gen) = prom_vals;

   otherwise
      error('>>> This is not a valid promotion case <<<')

   end
```

```
for i = 1:experiment.sub_team_size:comp_size
   % We are taking the promotion values for the teams
   % and then multiplying them to the individual's rank on the teams.
   % Stores the current generation rank in variable arank, because the new
ranking is
   % computed in this current generation and overwrites the old information.
The
   % new ranking is passed over to the net_shuffler routine and is moved with
the
   % individual to the new team.  After the shuffle, the old ranking
information
   % is placed back into the current generation.

   arank(i:(i+experiment.sub_team_size-1),1) =
popa(gen).rank(i:(i+experiment.sub_team_size-1));
   popa(gen).rank(i:(i+experiment.sub_team_size-1)) = ...
      popa(gen).rank(i:(i+experiment.sub_team_size-1))*...
      prom_vala((i+experiment.sub_team_size-1)/experiment.sub_team_size);

   brank(i:(i+experiment.sub_team_size-1),1) =
popb(gen).rank(i:(i+experiment.sub_team_size-1));
   popb(gen).rank(i:(i+experiment.sub_team_size-1)) = ...
      popb(gen).rank(i:(i+experiment.sub_team_size-1))*...
      prom_valb((i+experiment.sub_team_size-1)/experiment.sub_team_size);

end

%uk = 4;
%monitor(gen,:) = [prom_vala(uk) popa(gen).rank(uk*experiment.sub_team_size)
popa(gen+1).rank(uk*experiment.sub_team_size)];

for i = 1:experiment.sub_team_size:sub_size
   % We are taking the promotion values for the teams
   % and then multiplying them to the individual's rank on the teams.
   % Stores the current generation rank in variable arank, because the new
ranking is
   % computed in this current generation and overwrites the old information.
The
   % new ranking is passed over to the net_shuffler routine and is moved with
the
   % individual to the new team.  After the shuffle, the old ranking
information
   % is placed back into the current generation.

   srank(i:(i+experiment.sub_team_size-1),1) =
subteam(gen).rank(i:(i+experiment.sub_team_size-1));
   subteam(gen).rank(i:(i+experiment.sub_team_size-1)) = ...
      subteam(gen).rank(i:(i+experiment.sub_team_size-1))*...
      prom_vals((i+experiment.sub_team_size-1)/experiment.sub_team_size);
end


% This section randomly assigns individuals to new teams, using the routine
% net_shuffler2.  Afterwards,
```

```
[popa(gen+1),popb(gen+1),subteam(gen+1)] =
net_shuffler2(popa(gen),popb(gen),subteam(gen));


for i = 1:experiment.sub_team_size:sub_size
   subteam(gen).rank(i:(i+experiment.sub_team_size-1)) = ...
      srank(i:(i+experiment.sub_team_size-1),1);
end

for i = 1:experiment.sub_team_size:comp_size
   popa(gen).rank(i:(i+experiment.sub_team_size-1)) =
arank(i:(i+experiment.sub_team_size-1),1);
   popb(gen).rank(i:(i+experiment.sub_team_size-1)) =
brank(i:(i+experiment.sub_team_size-1),1);
end


% ********** Learning Procedure **********

switch learning
case 1

    disp('Case 1 - No Learning')

case 2

   disp('Case 2 - Learning in all teams')
   % Learning in Popa
   [popa(gen+1)] = net_learning1(popa(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
      experiment.p_mutation, experiment.p_network);
   % Learning in Popb
   [popb(gen+1)] = net_learning1(popb(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
      experiment.p_mutation, experiment.p_network);
   % Learning in Subteams (Mixed teams)
   [subteam(gen+1)] = net_learning1(subteam(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
      experiment.p_mutation, experiment.p_network);


case 3
   % Mixed team learning only
   disp('Case 3 - Learning in mixed teams only')
   [subteam(gen+1)] = net_learning1(subteam(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
      experiment.p_mutation, experiment.p_network);

case 4
   disp('Case 4 - Learning in all teams after a delay')

   if gen > stepl
      disp('Learning Enabled')
      % Learning in Popa
      [popa(gen+1)] = net_learning1(popa(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
         experiment.p_mutation, experiment.p_network);
```

```
      % Learning in Popb
      [popb(gen+1)] = net_learning1(popb(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
         experiment.p_mutation, experiment.p_network);
      % Learning in Subteams (Mixed teams)
      [subteam(gen+1)] =
net_learning1(subteam(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
         experiment.p_mutation, experiment.p_network);
   end

case 5

   disp('Case 5 - Learning in mixed teams after a delay')
   if gen > stepl
      disp('Learning Enabled')
      % Mixed team learning only
      [subteam(gen+1)] =
net_learning1(subteam(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
         experiment.p_mutation, experiment.p_network);
   end

case 6
   disp('Case 6 - Learning and using the network')
   [popa(gen+1), popb(gen+1), subteam(gen+1)] = net_learning(...
      popa(gen+1), popb(gen+1), subteam(gen+1), experiment.sub_team_size,...
      experiment.p_crossover, experiment.p_mutation, experiment.p_network,1);
   [popb(gen+1), popa(gen+1), subteam(gen+1)] = net_learning(...
      popb(gen+1), popa(gen+1), subteam(gen+1), experiment.sub_team_size,...
      experiment.p_crossover, experiment.p_mutation, experiment.p_network,0);

case 7
   disp('Case 7 - Learning and using the network after a delay')

   if (gen > stepl)
      disp('Learning and Network is Enabled')
   [popa(gen+1), popb(gen+1), subteam(gen+1)] = net_learning(...
      popa(gen+1), popb(gen+1), subteam(gen+1), experiment.sub_team_size,...
      experiment.p_crossover, experiment.p_mutation, experiment.p_network,1);
   [popb(gen+1), popa(gen+1), subteam(gen+1)] = net_learning(...
      popb(gen+1), popa(gen+1), subteam(gen+1), experiment.sub_team_size,...
      experiment.p_crossover, experiment.p_mutation, experiment.p_network,0);
   end

otherwise
      error('>>> This is not a valid promotion case <<<')

end


disp('This is generation: ')
gen

end % End of generation for loop
```

```
avg_fig = figure;
figure(avg_fig);

subplot(221)
plot(1:experiment.generation_max,avg_teamp,'r-
',1:experiment.generation_max,var_teamp,'b:')
xlabel('Generations')
ylabel('Average - Red & Variance - Blue')
title('Population Avg & Var over Gens')
grid;

subplot(222)
plot(1:experiment.generation_max,avga,'r-
',1:experiment.generation_max,vara,'b:')
xlabel('Generations')
ylabel('Average - Red & Variance - Blue')
title('Population A Avg & Var over Gens')
grid;

subplot(223)
plot(1:experiment.generation_max,avgb,'r-
',1:experiment.generation_max,varb,'b:')
xlabel('Generations')
ylabel('Average - Red & Variance - Blue')
title('Population B Avg & Var over Gens')
grid;

subplot(224)
plot(1:experiment.generation_max,avgs,'r-
',1:experiment.generation_max,vars,'b:')
xlabel('Generations')
ylabel('Average - Red & Variance - Blue')
title('Population Mix Avg & Var over Gens')
grid;


disp('Simulation Time')
ftimer=cputime-timer % Finished
```

10.2.7 Network1_sim.m – Core of Fully Merged Company Simulator

```
%  Raymond Ro, 2001
%  Single company simulator

   disp('*************** Single Company Simulator ******************')



   % ************************************************
   % **********   Sets the promotion mode.   **********
   % Mode         Desc.
   % ----         -----
   %  1           No promotional value is used (no weighting).
   %
   %  2           Promotion is used (weighting).
   %
   %  3           Promotion (weighting) is turned on after a specified number
of generations.
   %              The variable step is used to specify the number of
generations w/o weighting.
   %
   %  4           Demotion (weighting) is turned on after a specified number
of generations.
   %              The variable step is used to specify the number of
generations w/o weighting.

   prom_mode = 3;
   step = 10;
   % ************************************************



   % **********   Sets the Learning mode.   **********
   % Mode         Desc.
   % ----         -----
   %  1           No learning occurs.
   %
   %  2           Learning in all teams occur.
   %
   %  3           Learning in all teams occur after a certain time.
   %              Variable stepl is used to indicate when learning is turned
on.

   learning = 3;
   stepl = 40;
   % ************************************************

   % **********   Sets the Histogram movie mode.   **********
   % Mode         Desc.
   % ----         -----
   %  0           No movie.
   %
   %  1           Movieswitch is set to 1 and a movie is recorded.
```

```
   movieswitch = 1;
   % ************************************************


% ********** Dimensional Resolution **********

% This section calculates the number of bits needed for each gene in the
chromosome.
locus=zeros(size(experiment.domain,1),3);
for dim=1:size(locus,1) % 1 to length of locus (rows)
   locus(dim,3)=ceil(log2(abs(experiment.domain(dim,1)-
experiment.domain(dim,2))/experiment.epsilon(dim)));
end

% Now we start to build the String loci definition
%  (Starting bit | Stopping bit | Segment length) for each gene
locus(1,1)=1;
locus(1,2)=locus(1,3);
for dim=2:size(locus,1)
   locus(dim,1)=locus(dim-1,2)+1;
   locus(dim,2)=locus(dim,1)+locus(dim,3)-1;
end
fprintf('String loci (left, right, and lengths) per dimension:\n');
disp(locus);

timer=cputime; % start


% Initialize populations

generation = 1;

chrom_len = locus(end,2);
comp_size = experiment.corp_teama*experiment.sub_team_size;




   % **********   Sets the initial population policy distribution.
**********
   % Mode         Desc.
   % ----         -----
   %  1           Creates policies based on a uniform distribution over the
whole
   %              dynamic range.
   %
   %  2           Creates polices based on a uniform distribution.  The
company will
   %              have a mean of u1. Margin specifies how wide the
distribution
   %              should spread around the mean.


   distribution = 1;
   base = 2;
   drange = base^chrom_len;
```

```
   u1 = 2;
   margin = 1;



   % Index (Generations)

% This allocates memory ahead of time for the various population attributes

for generation=1:experiment.generation_max+1
   popa(generation).chromosome = zeros(comp_size,chrom_len);
   popa(generation).team_id = zeros(comp_size,1);
   popa(generation).id = zeros(comp_size,1);
   popa(generation).rank = zeros(comp_size,1);
   popa(generation).network = zeros(comp_size,experiment.net_size);
   % which pop a individual in the network is from  1 - other population
   % Descibes the company an individual is from
   popa(generation).net_team = zeros(comp_size,experiment.net_size);
   popa(generation).team_policy = zeros(experiment.corp_teama,chrom_len);
   popa(generation).team_ranking = zeros(experiment.corp_teama,1);
   % For Debugging purposes.
   popa(generation).xover = zeros(comp_size,1);
end




%  Create an initial value for each individual
popa(1).rank = ones(comp_size,1);

%  Assign ID's to all individuals
popa(1).id(1:comp_size) = 1:comp_size;


%  Assign each member to a team

%  This is just picking a batch of individuals and then assigning them to a
team
for team = 1:experiment.corp_teama
   popa(1).team_id(experiment.sub_team_size*team-
experiment.sub_team_size+1:team*experiment.sub_team_size)=team*...
      ones(experiment.sub_team_size,1);
end


%  Create an initial value for each individual

switch distribution
case 1
   disp('Using a uniform distribution to create policies')
   popa(1).chromosome = ceil(rand(comp_size,locus(end,2))*2)-1;
case 2
   disp('Using a uniform distribution to create a biased policy around a
mean')
   [popa(1).chromosome] = bias_policy_one(popa(1).chromosome, u1, drange,
margin);
otherwise
```

```
   error('>>> This is not a valid initialization case <<<')
end



% ************************ This is the main loop ************************

% Creates a handle for the figure


switch movieswitch
   case 0

   case 1
      movie_fig = figure;
   otherwise
      error('>>> This is not a valid movie option <<<')

end


for gen = 1:experiment.generation_max  % Need to expand this for the total
number of generations


% ******************* Create team policies *******************
% This computes the team policy for the corporation
for i = 1:experiment.sub_team_size:comp_size

popa(gen).team_policy((i+experiment.sub_team_size-
1)/experiment.sub_team_size,:) = ...
   team_policy(popa(gen).chromosome(i:i+experiment.sub_team_size-
1,1:chrom_len),...
   popa(gen).rank(i:i+experiment.sub_team_size-1))-48;

end


% ******************* Evaluate Team Fitnesses *******************

policy_veca = [bin2dec(char(popa(gen).team_policy+48))]';
policy_veca = (policy_veca/base^(chrom_len));
policy_veca = policy_veca*diff(experiment.domain) + experiment.domain(1);

% This is to capture the histogram
tpolicy = [policy_veca];
 % subplot(211)

 tpol(gen,:) = tpolicy;
 avg_teamp(gen) = mean(tpolicy);
 var_teamp(gen) = var(tpolicy);



switch movieswitch
   case 0
```

```
   case 1
      % Creating a movie of the histogram
      [n(gen,:), bincen(gen,:)] = hist(tpolicy);
      hist(tpolicy);
      axis([experiment.domain(1) experiment.domain(2) 0 max(n(gen,:))+2])
      title('Histogram For All Teams')
      xlabel('Team Policy (Number of Programmers)')
      ylabel('Number of Teams With Policy')
      grid;

      figure(movie_fig);
      m(gen) = getframe;
   otherwise
      error('>>> This is not a valid movie option <<<')

end

% This returns the fitness of each team policy
fit_a = net_fitness(policy_veca);


% *** Promotion is calculated here ***

switch prom_mode
   case 1
      disp('Case 1 - No promotion used');
      [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);


      % This variable are for monitoring purposes
      m_vala(:,gen) = prom_vala;

   case 2
      disp('Case 2 - Promotion used');

      [prom_vala, popa(gen).team_ranking] = net_promotion(fit_a);


      % This variable are for monitoring purposes
      m_vala(:,gen) = prom_vala;


   case 3

      % *** This is used to turn on ranking. ***

      disp('Case 3 - Promotion is turned on with a delay.');
      if gen <= step
         % Use net_promotion2 for non-promotion
         [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);
      else
         disp('Promotion enabled')
         % Use net_promotion to allow for promotion of individuals
         [prom_vala, popa(gen).team_ranking] = net_promotion(fit_a);
      end

      % These variables are for monitoring purposes
```

```
        m_vala(:,gen) = prom_vala;

   case 4

      % *** This is used to turn on ranking. ***

      disp('Case 4 - Demotion is turned on with a delay.');
      if gen <= step
         % Use net_promotion2 for non-promotion
         [prom_vala, popa(gen).team_ranking] = net_promotion2(fit_a);
      else
         disp('Demotion enabled')
         % Use net_promotion to allow for promotion of individuals
         [prom_vala, popa(gen).team_ranking] = net_demotion(fit_a);
      end

      % These variables are for monitoring purposes
       m_vala(:,gen) = prom_vala;

   otherwise
      error('>>> This is not a valid promotion case <<<')

   end


for i = 1:experiment.sub_team_size:comp_size
   % We are taking the promotion values for the teams
   % and then multiplying them to the individual's rank on the teams.
   % Stores the current generation rank in variable arank, because the new
ranking is
   % computed in this current generation and overwrites the old information.
The
   % new ranking is passed over to the net_shuffler routine and is moved with
the
   % individual to the new team.  After the shuffle, the old ranking
information
   % is placed back into the current generation.

   arank(i:(i+experiment.sub_team_size-1),1) =
popa(gen).rank(i:(i+experiment.sub_team_size-1));
   popa(gen).rank(i:(i+experiment.sub_team_size-1)) = ...
      popa(gen).rank(i:(i+experiment.sub_team_size-1))*...
      prom_vala((i+experiment.sub_team_size-1)/experiment.sub_team_size);

end


[popa(gen+1)] = net_shuffler(popa(gen));

% Copies proper ranking back to the current generation
for i = 1:experiment.sub_team_size:comp_size
   popa(gen).rank(i:(i+experiment.sub_team_size-1)) =
arank(i:(i+experiment.sub_team_size-1),1);
end


% ********** Learning Procedure **********
```

```
switch learning
case 1

    disp('Case 1 - No Learning')

case 2

   disp('Case 2 - Learning in all teams')
   % Learning in Popa
   [popa(gen+1)] = net_learning1(popa(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
      experiment.p_mutation, experiment.p_network);

case 3
   disp('Case 3 - Learning in all teams after a delay')

   if gen > stepl
      disp('Learning Enabled')
      % Learning in Popa
      [popa(gen+1)] = net_learning1(popa(gen+1),experiment.sub_team_size,
experiment.p_crossover,...
         experiment.p_mutation, experiment.p_network);
   end

otherwise
      error('>>> This is not a valid promotion case <<<')

end


disp('This is generation: ')
gen

end % End of generation for loop


avg_fig = figure;
figure(avg_fig);

plot(1:experiment.generation_max,avg_teamp,'r-
',1:experiment.generation_max,var_teamp,'b:')
xlabel('Generations')
ylabel('Average - Red & Variance - Blue')
title('Population Avg & Var over Gens')
axis([1 experiment.generation_max 0 max(avg_teamp)+2])
grid;

disp('Simulation Time')
ftimer=cputime-timer % Finished
```

10.2.8 Team_policy.m

```
function [chromosome] = team_policy(team,trank)
% This creates the net team policy, given a team.

%temp = bin2dec(char(48+team))
%trank(1:length(trank)) = 1:length(trank)
%wsump = sum((bin2dec(char(48+team)).*trank))
%sump = sum(trank)

team_pol = round(sum((bin2dec(char(48+team)).*trank))/sum(trank));
chromosome = dec2bin(team_pol,length(team(1,:)));
```

10.2.9 Rip.m

```
function [a, b] = rip(subt)
% rip.m subroutine
% This subroutine is used to break-up the mixed subteam group
% into the popa and popb group.  These subgroups are then returned
% to be incorporated with the populations for the A and B corporations.

%disp('*** This is the length in the rip subroutine ***')

len = length(subt.chromosome(:,1));

%disp('*** Subteam ID info ***')
%subt.corp_id

cnta = 1;
cntb = 1;
for i = 1:len

   if subt.corp_id(i) == 97
      a.chromosome(cnta,:) = subt.chromosome(i,:);
      a.id(cnta) = subt.id(i);
      a.rank(cnta) = subt.rank(i);
      a.network(cnta,:) = subt.network(i,:);
      % Debug purposes
      a.net_team(cnta,:) = subt.net_team(i,:);
      a.xover(cnta,:) = subt.xover(i,:);
      cnta = cnta + 1;
   else
      b.chromosome(cntb,:) = subt.chromosome(i,:);
      b.id(cntb) = subt.id(i);
      b.rank(cntb) = subt.rank(i);
      b.network(cntb,:) = subt.network(i,:);
      % Debug purposes
      b.net_team(cntb,:) = subt.net_team(i,:);
      b.xover(cntb,:) = subt.xover(i,:);
      cntb = cntb + 1;
   end

end

% Debug stuff
%disp('*** Size of pop A ***')
%size(a.chromosome(:,1))
%disp('*** Size of pop B ***')
%size(b.chromosome(:,1))
```

10.2.10 Net_shuffler2.m

```
function [popa, popb, subt] = net_shuffler2(popa, popb, subt)
% Only the current generation is passed over.


lenpa = length(popa.chromosome(:,1));
lenpb = length(popb.chromosome(:,1));

lens = length(subt.chromosome(:,1));

% This rips the combined subteam apart
[suba, subb] = rip(subt);
sublen = length(suba.chromosome(:,1));

% Now we combine the subteam members with the rest of the population
[tpopa] = combine(popa, suba);
[tpopb] = combine(popb, subb);


range = length(tpopa.chromosome(:,1));

% Now we swap all of the individuals
for i = 1:range
   rndposa = ceil(range*rand(1,1));
   rndposb = ceil(range*rand(1,1));

   tpopa = swap(tpopa, i, rndposa);
   tpopb = swap(tpopb, i, rndposb);
end

% We split the new subteam members from the population
[tmpa,suba] = split(tpopa, lenpa, sublen);
[tmpb,subb] = split(tpopb, lenpb, sublen);

   % Now we combine the subteam members
   temp = sew(suba,subb);

   % Add attributes that have changed back to the populations
   subt.chromosome = temp.chromosome;
   subt.id = temp.id;
   subt.rank = temp.rank;
   subt.network = temp.network;
   subt.net_team = temp.net_team;
   subt.xover = temp.xover;

   popa.chromosome = tmpa.chromosome;
   popa.id = tmpa.id;
   popa.rank = tmpa.rank;
   popa.network = tmpa.network;
   popa.net_team = tmpa.net_team;
   popa.xover = tmpa.xover;

   popb.chromosome = tmpb.chromosome;
   popb.id = tmpb.id;
```

```
   popb.rank = tmpb.rank;
   popb.network = tmpb.network;
   popb.net_team = tmpb.net_team;
   popb.xover = tmpb.xover;

   % Add in sew routine and then recombine routine, or put recombine into sew
and pass over
   % the subteam variable.


popa;
popb;
subt;
```

10.2.11 Combine.m

```
function [tpop] = combine(pop, sub)

lenp = length(pop.chromosome(:,1));
lens = length(sub.chromosome(:,1));
tpop = pop;

for i = 1:lens
   tpop.chromosome(i+lenp,:) = sub.chromosome(i,:);
   tpop.id(i+lenp) = sub.id(i);
   tpop.rank(i+lenp) = sub.rank(i);
   tpop.network(i+lenp,:) = sub.network(i,:);
   % Debug purposes
   tpop.net_team(i+lenp,:) = sub.net_team(i,:);
   tpop.xover(i+lenp,:) = sub.xover(i,:);
end
```

10.2.12 Swap.m

```
function [pop] = swap(pop, p1, p2)
%  Will swap 2 individuals given their location within the population

%size(pop.chromosome)
%p1
%p2
%pause

temp.chromosome(1,:) = pop.chromosome(p2,:);
temp.id = pop.id(p2);
temp.rank = pop.rank(p2);
temp.network(1,:) = pop.network(p2,:);
% Debug terms
temp.net_team(1,:) = pop.net_team(p2,:);
temp.xover(1,:) = pop.xover(p2,:);

pop.chromosome(p2,:) = pop.chromosome(p1,:);
pop.id(p2) = pop.id(p1);
pop.rank(p2) = pop.rank(p1);
pop.network(p2,:) = pop.network(p1,:);
% Debug
pop.net_team(p2,:) = pop.net_team(p1,:);
pop.xover(p2,:) = pop.xover(p1,:);

pop.chromosome(p1,:) = temp.chromosome(1,:);
pop.id(p1) = temp.id;
pop.rank(p1) = temp.rank;
pop.network(p1,:) = temp.network(1,:);
% Debug
pop.net_team(p1,:) = temp.net_team(1,:);
pop.xover(p1,:) = temp.xover(1,:);
```

10.2.13 Split.m

```
function [pop, sub] = split(tpop, lenp, lens)
% This splits tpop into the sub pop and pop components

% disp('**************** SPLIT FUNCTION *****************')


pop.chromosome(1:lenp,:) = tpop.chromosome(1:lenp,:);
pop.id(1:lenp,1) = tpop.id(1:lenp);
pop.rank(1:lenp,1) = tpop.rank(1:lenp);
pop.network(1:lenp,:) = tpop.network(1:lenp,:);
% Debug purposes
pop.net_team(1:lenp,:) = tpop.network(1:lenp,:);
pop.xover(1:lenp,:) = tpop.xover(1:lenp,:);
%size(tpop.chromosome)
%lenp+1
%lenp
%lens


sub.chromosome(1:lens,:) = tpop.chromosome(lenp+1:lenp+lens,:);
sub.id(1:lens,1) = tpop.id(lenp+1:lenp+lens);
sub.rank(1:lens,1) = tpop.rank(lenp+1:lenp+lens);
sub.network(1:lens,:) = tpop.network(lenp+1:lenp+lens,:);
% Debug purposes
sub.net_team(1:lens,:) = tpop.net_team(lenp+1:lenp+lens,:);
sub.xover(1:lens,:) = tpop.xover(lenp+1:lenp+lens,:);
```