APPLICATION OF GENETIC ALGORITHMS TO ANALYSIS AND POLICY DESIGN IN SYSTEM DYNAMICS

by

Ceyhun Eksin

B.S., Electric Engineering, Istanbul Technical University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2009

ACKNOWLEDGEMENTS

I am deeply grateful to Professor Yaman Barlas, my thesis supervisor, for being a great example of an enthusiastic scientist. I especially would like to thank him for his invaluable guidance, and for teaching me the scientist's approach to life.

I would like to thank Assoc. Prof. Necati Aras and Prof. Cem Say for taking part in my thesis jury and providing valuable feedback.

I would like to thank Ali Osman Konuray and Genco Fas for their company, during and after intense academic moments. I would also like to thank members of SESDYN Research Group for their support and friendship, and all the bright people in the department for contributing to my academic development.

I would like to thank Defne Över for helping me go through the worst and enjoy the good times.

I would like to express my deepest gratitude to my mother. I would not be able to get to where I am right now without her support and care. Lastly, I would like to thank my father for being a true guide in life.

This thesis was supported by TUBITAK research grant for M. S. Students.

ABSTRACT

APPLICATION OF GENETIC ALGORITHMS TO ANALYSIS AND POLICY DESIGN IN SYSTEM DYNAMICS

System Dynamics is a methodology to build valid models of systems in order to come up with policies to improve their dynamic behaviors. Search algorithms are increasingly being used in model building, validation, sensitivity analysis, and policy design phases of System Dynamics. This increase in application of search algorithms brings out the necessity to evaluate the efficacy of these algorithms with respect to the expectations of System Dynamics. This necessity forms the primary objective of this thesis.

The thesis focuses on Genetic Algorithms (GA), which is a well-known and frequently used search algorithm. The main objective is to explore Genetic Algorithms' adequacy in applications to System Dynamics methodology. We create eight variants of GA from the vast amount of options that the GA approach provides. The study focuses on six different dynamic models, three of them being from the area of control theory and the other three from socio-economic System Dynamics literature. For each model, we determine an appropriate objective function based on some desired dynamic behavior and set of analysis parameters. The performance of eight GA configurations for all of the models indicates that Genetic Algorithms obtain good solutions, i.e. behaviors that match the desired behaviors. Specifically, explorative type of Genetic Algorithms is more suitable for System Dynamics methodology: as the search space grows, they are more likely to find good solutions fast. Additionally, we emphasize to the fact that the objective functions and results provided by algorithms should not be regarded as absolute or conclusive truths; rather they must be critically analyzed, objective function and parameters of the algorithm iteratively reformulated as necessary. We demonstrate this fact in some of the models by further analyzing the systems based on initial results provided by the algorithm.

ÖZET

SİSTEM DİNAMİGİ MODELLERİNİN ANALİZİ VE STRATEJİ TASARIMINDA GENETİK ALGORİTMA UYGULAMALARI

Sistem Dinamiği yöntemi, sistemlerin analizi ve strateji tasarımı için geçerli modeller kurulması ve kullanılmasıdır. Arama algoritmaları Sistem Dinamiğinin model kurma, geçerleme, hassasiyet ve senaryo analizi, ve strateji tasarımı aşamalarında artarak kullanılmaktadır. Arama algoritmalarının uygulamasındaki bu artış, bu algoritmaların Sistem Dinamiğinin beklentilerine göre verimliliğini değerlendirmeyi zorunlu kılmıştır. Bu zorunluluğa yanıtlar bulmak bu çalışmanın ana hedefini oluşturmaktadır.

Bu tez bilinen ve sıkça kullanılan bir arama algoritması olan Genetik Algoritmalar (GA) üzerine yoğunlaşmaktadır. Ana amaç, Genetik Algoritmalarının Sistem Dinamiği uygulamalarındaki yeterliliğini araştırmaktır. Genetik algoritmanın sunduğu geniş seçenekler içinden sekiz farklı algoritma yaratılmıştır. Çalışma, üç tanesi kontrol teorisi alanından, diğer üçü de sosyo-ekonomik Sistem Dinamiği alanından olmak üzere altı farklı model üzerinde odaklanmıştır. Her model için istenilen dinamik davranışa uygun amaç fonksiyonu ve analiz parametreleri belirlenmiştir. Sekiz farklı Genetik Algoritmanın bu modellerdeki performansı, Genetik Algoritmaların iyi sonuçlar, diğer bir deyişle istenilene uygun dinamik davranışlar elde ettiklerini göstermektedir. Özellikle, daha ucu açık/araştırmacı tipli Genetik Algoritmaların, arama alanı büyüdükçe iyi sonuçlara daha hızlı bir şekilde ulaşma yatkınlıkları olduğundan, Sistem Dinamiği yöntemine daha uygun oldukları görülmüştür. Ek olarak, bu çalışma, amaç fonksiyonu ve arama sonuçlarının kesin ve mutlak doğrular olarak kabul edilmesi yerine, sonuçların dikkatli bir şekilde analiz edilip, gerektiğinde amaç fonksiyonları ve algoritma parametrelerinin yeniden belirlenmesinin yararlarını ortaya çıkarmıştır. Birkaç modelde algoritmanın ilk sonuçlarına göre daha kapsamlı analiz yapılarak, bu tekrar gözden geçirme biçimi ve yararları gösterilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

$\mathbf{B}_{mxn}$	Best individuals' (solutions) matrix
Bac	Backlog
Bac^{*}	Desired Backlog
Bac^{MAX}	Maximum Backlog
Bac^{S}	Settled Backlog
b_{ij}	Best individual of replication i with j^{th} GA configuration
b_i^{min}	Best individual (solution) from replication i
b_i^{max}	Worst individual (solution) from replication i
b_j^{min}	Best individual (solution) from j^{th} GA configuration
b_j^{max}	Worst individual (solution) from j^{th} GA configuration
b_{ij}^{min*}	Best individual (solution) among all runs
b_{ij}^{max*}	Worst individual (solution) among all runs
C	Capacity
D	Displacement
D^{*}	Desired Displacement
D^{S}	Settled Displacement
e_{ss}	Steady state error
$\mathbf{F}_{mxn}$	Fitness value matrix
F_d	Fan force
FM	Fan-Motor
f_{ij}	Fitness value of replication i with j^{th} GA configuration
f_j^{min}	Best fitness value from j^{th} GA configuration
f_j^{max}	Worst fitness value from j^{th} GA configuration
$\bar{f}_j$	Mean fitness value from j^{th} GA configuration
f_{ij}^{min*}	Best fitness value among all runs
f_{ij}^{max*}	Worst fitness value among all runs
GA	Genetic Algorithm
I	Current
K_P	Proportional control constant

K_I	Integration control constant
K_D	Derivative control constant
L	Inductance
MBS	Magnetic Ball Suspension system
NormCompl	Normal Completion
NormBacklog	Normal Backlog
P	Proportion
PI	Proportion-Integration
PID	Proportion-Integration-Derivative
Pts	Penalty constant for settling time
Pess	Penalty constant for steady state
Po	Penalty constant for overshoot
S	Stock Level
S^*	Desired Stock Level
S^{MAX}	Maximum Stock Level
S^s	Settled Stock Level
SA	Stock Adjustment
SAA	Simulated Annealing Algorithms
Sal	Salesmen
Sal^*	Desired Salesmen
Sal^{MAX}	Maximum Salesmen
Sal^S	Settled Salesmen
sat	Stock Adjustment Time
SD	System Dynamics
SLA	Supply Line Adjustment
smthtime	Smoothing time
T_{AD}	Acquisition Delay
T_{SA}	Stock Adjustment Time
t_s	Settling time
O	Overshoot
W_M	Motor speed
W_S	Weight of Stock Adjustment
W_{SL}	Weight of Supply Line Adjustment

V	Voltage
V_F	Final Voltage
V^*	Desired Voltage
V^{MAX}	Maximum Voltage
V^S	Settled Voltage

1. INTRODUCTION

System Dynamics is a methodology for examining dynamic systems with complex feedback mechanisms. The aim of System Dynamics (SD) modeling is to obtain a valid representation of a real system and a deeper understanding of the system behavior via simulation. The ultimate aim is to come up with policies that improve the dynamic performance of the system with respect to the stated goals.

SD has made use of search algorithms such as Hill Climbing algorithms, Random Search, Simulated Annealing (SA), Genetic Algorithms (GAs) in model building, validation and policy design. Previous studies show that most of the time search algorithms provide better solutions than the alternative methods such as the traditional approach. Although the efficiency of these algorithms are tested and compared in other contexts, very few studies extensively test whether search algorithms provide "good" solutions with high percentage and efficiently in SD model applications. Coyle (1996) discusses ways to overcome the weaknesses of hill climbing algorithms. Miller (1998) compares the final population (objective function) values provided by random search, SAA and GA on the World 3 model. However, a "good" solution in SD should not be entirely based on the value of the objective function; it is mostly based on the judgment of the analyst.

GAs are used in various studies of optimization of parameter sets in SD models. Certainly, there is a need for a more extensive study on the efficiency and effectiveness of such search methods on SD models for synthetic policy design. One of the aims of this thesis is to evaluate the adequacy of GA in coming up with satisfactory policy parameters efficiently and effectively on various dynamic models with complex objective functions.

The second aim of this study has developed while searching for candidate models from a wide range of dynamic models to test the GA performance. Hence, the supplementary goal of this study is to analyze the generic physical system models used in control theory using stock-flow modeling interface, in the aim of providing a step towards a stronger bridge between the control theory and SD fields. The study involves a fan

motor model which includes a linear DC- motor, a nonlinear RLC circuit, and a nonlinear magnetic ball suspension system (MBS) which are generic systems used in control theory. The fan-motor and MBS systems are constructed and appropriate Proportion-Integration (PI) and Proportion-Integration-Derivative (PID) controllers are modeled. These generic controllers can also be used to develop policies for SD models. Besides these models, the study also applies the parametric search by GA to other generic SD models such as predator-prey, market growth and supply chain models.

An obvious output of simulation-based optimization is to obtain a deeper understanding of the structure of the model during developing the objective function and afterwards from the search results. This study shows that parameter optimization can also help analysts clarify their objectives or gain insight about the system while developing and improving the objective function. In light of this insight, we analyze some of these models further. Hence, we show that parameter search according to a performance index can yield essential information about the dynamics of a system.

In this research, we work on six dynamic models. Each model is extensively explained including mathematical analysis when possible. For each model, an objective function is defined and GA performance is analyzed. To evaluate the GA performance, different variations of GA are tested on each model with the aim of coming up with significant pointers for GA implementation. Some of the models are further analyzed for policy design.

2. LITERATURE REVIEW

Policy design or improvement of the system behavior is the utmost goal of a SD study. Although optimization tools provide possible efficiency and improvements during policy design, model building, validation, and sensitivity analysis, SD has been struggling to make effective use of these tools. With growing quantitative data and trend towards the aspiration for quantified results, abundant use of optimization tools in SD methodology is inevitable (Graham et al. 2003). There has been promising work, but researchers have approached the application of optimization to SD with caution due to the following reasons:

1) Model losing practical value because of oversimplification to be able use mathematical tools of optimization (Forrester 1961, Section I.2)
2) Fear of the designer resorting solely to the maximization or minimization of the objective function and missing other essential knowledge on the system that SD model provides
3) The structure of managerial models are so far away from optimal that great improvements can be obtained simply by analyzing feedback mechanisms and making intuitive structural changes

Hence, analysts have adopted what is called the traditional approach to policy design, which is usually an informal process where the model builder or some other expert applies parametric or structural changes to the system on a trial-and-error basis (Coyle 1977). Model builder's or expert's intuitive ability to come up with acceptable and good enough policy alternatives bound the traditional approach. Thus, Porter (1969) and others realized that a more analytic and guided approach that is not based solely on trial-and-error should be developed.

Formal approaches to policy design started almost three decades ago. Policy design of SD models has made use of two main optimization tools: control theory or optimal control, and search algorithms. There has been a major interest to apply control theory or optimal control theory methods to policy design on the first few years of the field's journal

"System Dynamics Review". SD literature has used concepts from control theory because both try to understand and control or provide policies that manage the systems according to certain goals. Benchmark studies on the application of modal control theory methods to SD models are by Mohapatra and Sharma (1985), and by Özveren and Sterman (1989). Macedo (1989) proposes another method combining optimal control and modal control. These were criticized for being not user friendly and hard to implement due to their need for rigorous mathematics.

Seemingly very similar, the differences between the types of problems that the control theory and SD fields are interested in have caused them to depart in separate ways. SD concentrates on managerial models where the model building process is very important. SD tries to create a critical thinking environment based on feedback perceptions on the model. Unlike control theory, models cannot be regarded as black box or simply a set of dynamic equations that creates output(s), given input(s). This different look on models by SD gives a flexible approach for policy design. Since model parameter values or structure can be altered, it gives SD a more diverse ability to deal with undesired behavior. To this date, there have been very few studies that have tried to come up with generic structural changes that make use of this flexibility of SD. Yasarcan (2003) and Barlas (2006) propose a structure called *virtual supply line* to deal with the undesired effects of delays, which can be applied, to any dynamic system with delays. Wolstenholme (1986) also proposes generic control structures that are applicable to a wide range of systems. In control theory, the systems that are tried to be controlled, generally cannot be changed, hence the control structure is added on to the system in a closed loop structure. However, the option to treat a system as black box also allows the application of other tools that are not always right or easy to implement on SD systems such as neural networks. One of the goals of this thesis is to form some new ties between these two fields by working on models and control structures closer to classical control theory, on SD interface.

Keloharju has leaded the other wing of application of optimization to SD methodology (1977) in which he provided a package (DYSMOD). In this package, he uses heuristic methods to optimize parameter values according to a certain objective function. The study by Keloharju is based on simulation but as argued by Keloharju and supported by Coyle (1996), it should not be seen as "optimization through repeated simulation, but as

simulation via repeated optimization". Coyle (1985) combined control theory rules of thumb with Keloharju's method. Other applications of similar methods to SD models, which make use of simulation based algorithmic search, have followed. Wolstenholme and Al-Alusi (1987) make use of DYSMOD package to optimize the strategy of an army in a defense model. Dangerfield and Roberts (1999) use hill-climbing search algorithm to fit the model to AIDS data to obtain the distribution of incubation period. Graham and Ariza (2003) present a real-world consulting application of the policy design by parameter optimization using simulated annealing.

Among search algorithms that are used for policy design by simulation-based optimization, there are a number of genetic algorithm applications. GA searches the solution space in multiple and random directions. GA is observed to be an effective algorithm for highly nonlinear solution spaces since it is not typically trapped in local optima. Miller (1998) uses hill-climbing algorithm and GA to search for parameters that World 3 model is sensitive to. Grossmann (2002) applies a genetic algorithm to Information Society Integrated Model by first stating several objective functions and then determining the set of policy parameters. Then, he interprets the results of the generated policies. One interesting approach is that he looks for a worst policy with the hope of discovering more about the system. Chen and Jeng (2004) do another application of GA to policy design on SD models where the main idea depends on the transformation of a SD model into a specially designed neural network. Then the policy design is viewed as a learning problem, and a genetic algorithm is applied on this neural network model. McSharry (2004) applies parameter search with GA to a malaria-control model. He further analyzes the robustness of the policy provided by GA. One other interesting approach to use GA as a tool for optimizing the behavior of dynamic model is provided by Yucel and Barlas (2007). A pattern recognition algorithm is used to measure the performance of different policies where an objective function is developed for performance measure based on the desired behavior *pattern* of the system. Duggan (2008) uses GA to map Pareto optimal front of a two-agent beer game where minimization of the costs incurred of both agents are tried to be minimized.

Simulation-based optimization or use of search algorithms requires determining a performance measure to guide the search. Generally, simulation-based optimization in SD

has been performed via maximizing or minimizing the final value of a function such as maximizing the final market share, by searching a single parameter or a set of parameters in the system (Wolstenholme and Al-Alusi 1987; Miller 1998; McSharry 2004; Grossmann 2002; Duggan 2008). Others use deviation from a desired path or data set as a performance measure (Dangerfield and Roberts 1999; Graham and Ariza 2003; Chen and Jeng 2004). However, in many cases it is not meaningful to look at the final value of a function. Neither, can we express our goals explicitly with a desired path, since we do not have a specific desired path. We often have some target factors or desired features or desired behavior. Hence, we need to formulate additional "equations to penalize failure to meet the target factors" (Coyle 1996 p.247). The wide range of SD models makes it hard to guide a policy optimizer in developing an objective function. Thus, simulation-based optimization needs the skills and knowledge of the analyst in developing an objective function and judging whether the output of an optimization meets the goals. The analyst should not give too much importance to the objective function and should be able to abandon an objective function and create another one based on doubts on the results of optimization (Coyle 1996 p. 294). This is another positive aspect of simulation-based optimization since it forces the analyst to state her/his goals clearly and to learn more about the system while trying to develop a "good" objective function.

3. GENETIC ALGORITHMS AND RESEARCH METHODOLOGY

In the first part of this section, GA is explained to provide some background on some of the elements used in the research. The second part explains the analysis proposed to test the adequacy of a GA in coming up with policy parameters.

3.1. Elements of Genetic Algorithms (GA)

Genetic Algorithms, almost universally abbreviated as GA, were first introduced by John Holland in 1975. GAs operate on ideas of biological evolution process. In fact, Holland followed the biological terminology closely and named each individual undergoing reproduction as chromosomes. Each basic unit of a chromosome is named a gene. A chromosome can be thought of as a string or a vector where each of its components is a symbol from a possible set of values. The set of vectors collectively make up the population. The main idea lying behind the GA is the reproduction of a population. Because of the reproduction, a new population is formed. Although many variations are possible and available, the main parts of reproduction, analogous to biological reproduction, are selection, recombination (i.e. crossover) and mutation. A common version of the algorithm is given on Figure 3.1.

```
choose initial population
evaluate each individual's fitness
repeat
Scale every individual according to their fitness
        select individuals based on their scaled fitness values
        mate pairs at random
                 apply crossover operator with a certain probability
                 apply mutation operator with a certain probability
         evaluate each individual's fitness
until terminating condition (e.g. until at least one individual has
          the desired fitness or enough generations have passed)
```

Figure 3.1. A genetic algorithm pseudo-code

As the pseudo-code implies, each of these steps could be done with different operators, functions, or values. GA have many elements which provide high flexibility in coming up with an appropriate algorithm for the search space. This flexibility makes GA

significantly more complicated than neighborhood search methods. While this opportunity for variation in implementation is one of the reasons of popularity of the algorithm, it is frustrating both for the practitioner and for the theorist (Reeves and Rowe 2003). Next, we describe the elements of GA and different strategies that one might use at the implementation of the algorithm.

We give the elements to provide an understanding of the algorithm and introduce some of the options that are used in this research. The list is not exhaustive but it gives a sound understanding on the vast amount of options that the algorithm provides and the user has to decide on.

3.1.1. Definition of the String of a Chromosome

The string of searching parameters for the optimization problem should be defined first. These parameters are genes in a chromosome, which can be binary, coded or real (decimal) coded.

[1.2 3.6 4.1]	Real coded chromosome with three genes
[010101100011]	Binary coded chromosome with two variables, six bits each

Figure 3.2. Example of real and binary coded genes

3.1.2. Initial Population

N chromosomes should be randomly generated before using a GA operation. These chromosomes are called the initial population. While using a GA one of the first tasks before implementing the algorithm is to decide on the size of the population. It seems that too small population size would decrease the chances of effectively exploring the search space, while too large a population would increase substantially the time to obtain good solutions. There are some studies that relate the population size to vector length (Reeves and Rowe 2003). Yet empirical results show and suggest that population size of about 50 is adequate in many cases (Reeves and Rowe 2003).

3.1.3. Termination Criteria

Since GAs are stochastic search methods, a termination condition is needed unlike simple local search methods that stop when there is no improvement on the objective value. Some of the stopping conditions are maximum number of iterations, time limit, or maximum number of iterations without an improvement on the fitness value.

3.1.4. Scaling

Before selection, the individuals should be scaled according to their fitness function. The simplest way is *proportional* scaling. This strategy may contain some disadvantages as raw fitness values are used. For example, the values 1 and 2 are much more distinguished than 101 and 102. Hence, when the raw scores are not in good range, proportional scaling may cause problems such as giving a "bad" individual almost the same probability to be selected as a "good" individual. This problem can be avoided by linear transformation. Another way of scaling is to use *ranking*. In this way, the selection probability of individuals is based on their rank based on their fitness value rather than their raw fitness values. Besides these two most popular ways of scaling, there are other scaling options. *Top* is one of them, which is assigning the top n of the fittest individuals the same probability ($1/n$) to be selected, and the rest of the individuals zero. The scaling element gives a measure of fitness for members of the population. Scaling elements have their weaknesses and strengths based on the individuals of the population, the search space and the values provided by the objective function.

3.1.5. Selection

Selection is an essential part of the natural evolution process and hence is an essential part of GA. We may account scaling as a sub-element of selection. Scaling provides a measure for the members of the population from which they are selected. The most commonly known selection element is *roulette-wheel* selection. In roulette-wheel method, the candidate individuals are given a slice of the wheel proportional to their assigned probabilities obtained by scaling. Then the wheel is spun N times where N is the number of individuals or parents needed for reproduction. Thus, the fittest individuals

have a higher probability of being chosen. However, bad solutions have also a small chance of survival. More computation time effective way to select individuals from the wheel is to create one random number and from that point continue in increments of equal size (1/*N*). Another alternative for selection is *tournament* in which τ chromosomes are selected randomly and the fittest is chosen among the τ. This procedure is again repeated *N* times. There can also be variations of this tournament where the best string does not always win but wins with a chance of $p<1$. These are called *soft* tournaments whereas the former type is called *strict* tournament (Reeves and Rowe 2003).

3.1.6. Crossover

Once the candidate parents are selected, they can be recombined in certain ways to form new offspring. In this section, we give some of the types of crossover and leave the discussion of when to use crossover and/or mutation after explaining them.

Given the importance of recombination pointed by influential authors of GA literature like Holland and Goldberg, there has been an extensive study to come up with many different ways of crossing over chromosomes. Here, we explain only few of the very common ones.

Single Point crossover is one of the most basic one. The position of the crossover is determined randomly. Then the determined length of the strings is exchanged between the mating pair.

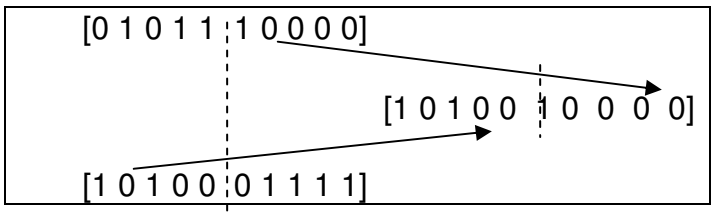

Figure 3.3. Example of a Single Point Crossover

Variations of this crossover exist. *Scattered* crossover where a vector the same size of the chromosome is created containing zeros and ones. The new child obtains the genes of the first parent whenever there is a one on the vector and gets the genes of the second

parent whenever there is a zero on the vector. *Two point* crossover is another one where two poitions for crossover is selected as *a* and *b*. Genes from 1 to *a* are from parent 1, from *a*+1 to *b* are from parent 2, and from *b*+1 to rest of the genes are from parent 1 again.

In *Heuristic* crossover, a child lying on the line containing parent 1 and parent 2 is created.

Child= Parent 1 + *r* (Parent 2 – Parent 1) Fitness(Parent 2) > Fitness(Parent 1) where *r*>1

The child lies farther away from the worse parent (which is Parent 1 for a maximization problem) since *r*>1. When *r*<1, the crossover is called *arithmetic*. In both cases, *r* can be a random number or previously fixed number satisfying the given conditions above. Many different variations of this type of crossover are possible.

Hybridization of these crossover operators is also possible. A set of crossover operations could be used each time with a certain probability.

3.1.7. Mutation

Once the decision of when to apply mutation is made, the mutation operator is simply randomly changing the value of a gene. For real valued genes, this change could be done by selecting a number uniformly from the range of possible values. One other possible way is to use a distribution like Gaussian with mean zero and a given variance where the number generated by this distribution is added to the value of the gene. It may be adjusted such that the variance decreases as the algorithm iterates. In our analysis for Gaussian mutation, the variance is chosen to be 1 and shrinks linearly to zero with each generation.

3.1.8. Crossover and Mutation Condition

It is totally the choice of implementation to apply crossover *and* mutation or crossover *or* mutation. A strategy would be to apply crossover with certain probability and

try to apply mutation with a certain probability to the selected strings. In this case, both of them could be applied to obtain an offspring or neither of them could be used where a selected individual passes to the next generation without being altered. A strategy to apply crossover or mutation to obtain the next generation could be chosen where at least and at most, one of the two is applied. This means a certain part of the population is created by using crossover and the rest is mutated (Reeves and Rowe 2003). It is also possible to make variations of these possible strategies.

3.1.9. Elitism

Because of reproduction of population, the next generation might or might not contain the best individual of the previous generation. In terms of optimization, losing the best individual does not make sense. Hence, De Jong (1975) introduces the idea of elites. A number of elites or the best one can be allowed to pass on to the next population. Elitism of single individual is also known as the survival of the fittest.

3.2. Implementation of the Genetic Algorithms

So far, in this chapter, the general structures of GA and possible elements that can be used are discussed. It is clear that GA allows vast amount of variation in implementation. In general, an algorithm with population based selection, recombination and mutation is called a GA (Reeves and Rowe 2003). The discussion of which set of strategies gives us the best is based on the optimization problem. However, some basic suggestions can be made. One such suggestion is that GAs are stochastic search algorithms and to obtain a general idea about the performance of a GA several replications should be done. It can also be suggested that the diversity of the population should be kept and premature convergence should be avoided. The empirical evidence also suggest that a population size of 50 is generally enough to efficiently scan the search space (Reeves and Rowe 2003).

3.3. Research methodology

Application of search methods to SD methodology during model building, validation and policy analysis has been enjoying a growing interest. For optimization,

clear statement of objectives is necessary. For each model to be experimented on, we develop an objective function or a fitness function. Then, the adequacy of GA in SD models is to be tested extensively. GA can have many variations and these variations can play a significant role in the performance of the algorithm. Hence, to test the performance of GA, these variations should also be analyzed. Some of the elements of the GA are kept the same in all of the variations. The chromosomes are real-coded. Population size is chosen to be 50. Initial population is created by using a uniform random number generator. The runs are limited to 100 generations.

In application of GA, we make use of *"Genetic Algorithm and Direct Search Toolbox"* of MATLAB®. Before we make use of the toolbox, we have also coded the GA with the following configuration to confirm the results provided by the toolbox:

- Individuals are scaled proportionally.
- Roulette-wheel selection is used to select parents.
- Number of elites is equal to one.
- Beside the elite individual that is carried on to the next generation, 0.95 of the next generation is created by crossover.
- 0.05 of the next population has the chance to be created by uniform mutation.
- Crossover type is heuristic crossover with ratio r=1.2.
- In uniform mutation, each gene has a 0.05 probability of being mutated.

In Figure 3.4, we give a detailed pseudo-code of the GA with configurations as mentioned above.

```
      initialization
      population size
      number of chromosomes
      number of elites
      crossover fraction
      percentage of individuals created by crossover(crossover kids)
      calculate crossover kids= crossover fraction*(population size-number of
elites)
      calculate mutation kids= population size-(crossover kids + number of elites)
      calculate number of parents nP= 2*crossover kids + mutation kids

      initial population creation

      create a vector of random numbers with size (population size, number of
chromosomes)
      initialpopulation= lowerbound + (upperbound-lowerbound)* random(uniform)

      repeat
      evaluate score for all individuals
      store the best individual

      Scale every individual proportional to their score
         if minimization lower scores should have higher expectation
         score(individual)= 2*meanscore-score(individual)
         expectation(individual)=number of parents*score(individual)/sum(scores)

      place individuals on the roulette-wheel based on their expectation
      spin the roulette wheel nP times to obtain the parents
      mate pairs at random

      apply heuristic crossover operator to obtain crossover kids
         kid= parent2 + ratio * (parent1 - parent2) given parent1 is better

      apply uniform mutation operator to the remaining parents
         repeat
                  if random number> mutation fraction
                  chromosome kid= chromosome parent
                  if random number< mutation fraction
                  chromosome kid= lowerbound + (upperbound-lowerbound)*
         random(uniform)
         until number of chromosomes are reached

      new population=crossover kids + mutation kids + best individual

      until terminating condition (number of generations are passed)
```

Figure 3.4. A detailed genetic algorithm pseudo-code

In order to test the performance of varying GA algorithms, we propose a base GA algorithm and make variations on this algorithm one by one. Base GA code has the same configuration as the GA in Figure 3.4. The elements to be tested are scaling and selection operators, crossover operator and ratio, mutation type, and number of elites. We come up with seven additional configurations by changing the elements of the base GA as given in Table 3.1. We are not going to display the GA code of these other configurations any further in the thesis and use the toolbox of MATLAB. Each of these configurations is

replicated several times with different random seeds. After the runs, the effects of these options are discussed by comparing each configuration's mean fitness value and worst solution obtained. An additional analysis is done in chapter five by evaluating the performance when the numbers of genes or variables in the chromosome are increased.

Table 3.1. Specification of base GA and other GA configurations

	Base GA	GA1	GA2	GA3	GA4	GA5	GA6	GA7
Scaling	Prop.	Rank	Top 20	Prop.	Prop.	Prop.	Prop.	Prop.
Selection	Roulette-Wheel	Roulette-Wheel	Roulette-Wheel	Tournament 4	Roulette-Wheel	Roulette-Wheel	Roulette-Wheel	Roulette-Wheel
Elites	1	1	1	1	5	1	1	1
Crossover Type	Heuristic r=1.2	Heuristic r=1.2	Heuristic r=1.2	Heuristic r=1.2	Heuristic r=1.2	Single Point	Heuristic r=1.2	Heuristic r=1.2
Crossover Ratio	0.95	0.95	0.95	0.95	0.95	0.95	0.7	0.95
Mutation Type	uniform 0.05	uniform 0.05	uniform 0.05	uniform 0.05	uniform 0.05	uniform 0.05	uniform 0.05	Gaussian 1 1

The random numbers are generated using MATLAB's random number generator. Throughout the thesis, we distinguish runs, results and points with replication numbers or configuration numbers. When we say best individual from GA-*j*, or j^{th} GA, we mean the best individual obtained from that configuration among j^{th} GA's replications. Let us say $i=1,\ldots,m$ and $j=1,\ldots,n$ and let f_{ij} denote the fitness value for replication i of j^{th} GA configuration. We define the best fitness value ($f_j^{\min}$) from j^{th} GA's replication as follows:

$$f_j^{\min} = \min_{i=1,\ldots,m} f_{ij} \tag{3.1}$$

The best individual (i.e. solution) associated with that best fitness ($f_j^{\min}$) from that configuration is $b_j^{\min}$.

In short, we can think of a fitness matrix $\mathbf{F}_{m\text{x}n}$ with elements f_{ij} where each column corresponds to a GA configuration j and each row corresponds to replication i. The best fitness value of the j^{th} GA is the minimum fitness value in column j of matrix **F**. Then we obtain the associated best individual $b_j^{\min}$ from the best individuals' matrix, $\mathbf{B}_{m\text{x}n}$. Each individual b_{ij} in matrix **B** is a (1 x k) vector where k is the number of design (policy) parameters in the model.

Similarly, we compute corresponding worst fitness values ($f_j^{\max}$) by taking the maximum value of fitness values in that column of matrix $\mathbf{F}_{m\text{x}n}$:

$$f_j^{\max} = \max_{i=1,\ldots,m} f_{ij} \tag{3.2}$$

Their associated best individual ($b_j^{\max}$) is again from the solutions' matrix $\mathbf{B}_{m\text{x}n}$.

We also use the mean (average) fitness $\bar{f}_j$ of j^{th} GA configuration. We define it as follows:

$$\bar{f}_j = \frac{\sum_{i=1}^{m} f_{ij}}{m} \tag{3.3}$$

Notice that we do not define an average best individual (i.e. solution), because in this case there is no one to one correspondence from **F** to **B**. This is clear since the average of elements (b_{ij}) of **B** is not defined since these elements correspond to (1 x k) vectors.

In our notation, when we refer to best, or worst fitness value or individual of a GA configuration (column), we keep j in the notation. For instance, we denote the best fitness of 2nd GA configuration as $f_{j=2}^{\min}$ or best individual of 5th GA configuration as $b_{j=5}^{\min}$.

Finally, we denote the best individual (i.e. fittest individual) among all runs as $b_{ij}^{\min^*}$ with its fitness value ($f_{ij}^{\min^*}$) as defined:

$$f_{ij}^{\min^*} = \min_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}} f_{ij} = \min_{j=1,\ldots,n} f_j^{\min} \tag{3.4}$$

Similarly, we define the worst individual among all runs as $b_{ij}^{\max^*}$ with its fitness value ($f_{ij}^{\max^*}$) as defined:

$$f_{ij}^{\max *} = \max_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}} f_{ij} = \max_{j=1,\ldots,n} f_j^{\max} \tag{3.5}$$

Throughout the thesis, the number of GA configurations is eight, meaning n=8 or j=0,1,…,7. These eight configurations are Base GA, GA1, GA2, GA3, GA4, GA5, GA6, and GA7 as given in Table 3.1. Notice 0 stands for Base GA in j. Generally, we make use of ranking of $f_j^{\min}$, $f_j^{\max}$, and $\bar{f}_j$ to compare solutions of GA configurations.

4. APPLICATION TO SELECTED MODELS

In this chapter, we work on six models. The first three are nonlinear electric circuit, a DC motor system and a magnetic ball suspension system. These are physical dynamic systems, which lie in the interest area of standard control theory. The last three are generic SD models: Simple Predator-Prey Model, Supply Chain Model, and Market Growth Model.

4.1. Nonlinear Electric Circuit

4.1.1. Model

The equations of the model are as shown below where *I* is the current and *V* is the voltage of the RLC circuit connected series (Clark, 2004).

$$\dot{I} = \frac{V}{L} - \frac{-I + I^3}{L} \tag{4.1}$$

$$\dot{V} = \frac{-I}{C} \tag{4.2}$$

where

$$R = -I + I^3 \tag{4.3}$$

Generally, RLC circuits are linear as '*R*' is a constant number. This circuit is nonlinear as the resistance '*R*' changes with cube of the current.

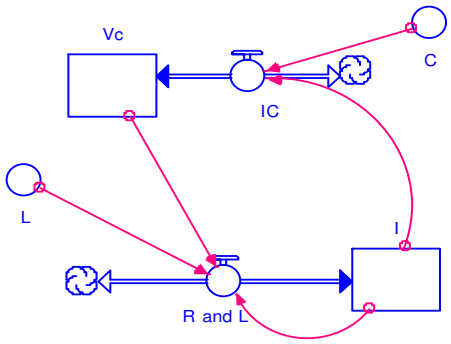

Figure 4.1. Stock-Flow diagram for the electric circuit model

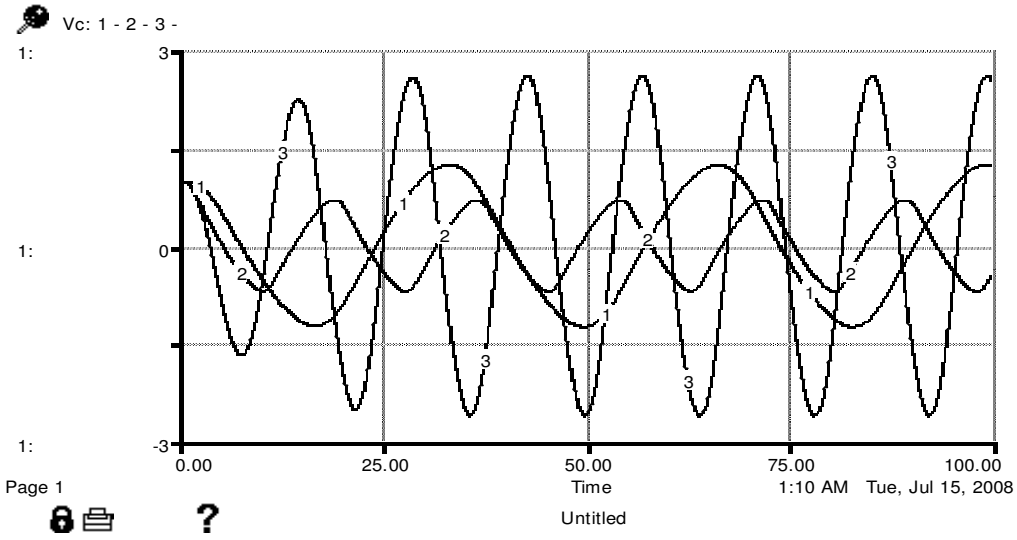

Figure 4.2. Behavior of the voltage with 1: (C, L) = (5, 5), 2: (C, L) = (5, 1), 3: (C, L) = (1, 5). Initially I=0, V=1

The search parameters are capacity (C) and inductance (L) values. Figure 4.2 demonstrates that the system shows constant oscillatory behavior after a transitory behavior independent of L and C values. Figure 4.2 also hints that C and L can be used to adjust the period and amplitude of the oscillation.

4.1.2. Fitness Function for the Electric Circuit Model

The desired behavior is chosen as a cosine wave with a certain period and amplitude for the voltage (V). The fitness function first calculates the average period, and

the amplitude. To obtain average period, first the times where oscillation reaches maximum values throughout the simulation time are marked (t_1, t_2,…,t_n). Then the spotted times are subtracted consecutively starting from t_n to obtain periods. The initial period where the system starts from one at time zero to t_1 is not calculated to omit transient behavior.

$$\begin{aligned} period_{n-1} &= t_n - t_{n-1} \\ period_{n-2} &= t_{n-1} - t_{n-2} \\ &\vdots \\ period_1 &= t_2 - t_1 \end{aligned} \tag{4.4}$$

Finally, the average period is calculated from these *n*-1 periods. To calculate the amplitude, max and min values of the voltage is subtracted.

$$amplitude = V_{MAX} - V_{MIN} \tag{4.5}$$

The calculated period and amplitude values are compared with the target period and amplitude of the cosine. Penalty for the period and amplitude difference is equal which means both goals are given equal importance. The desired cosine function is cos(π/10) which has a period of 20 and oscillates between -1 and 1. This fitness function could easily be applied to any model which has potential to show constant oscillation where the desired behavior is oscillatory.

$$period\_error = |average\_period - desired\_period| \tag{4.6}$$

$$amplitude\_error = |amplitude - desired\_amplitude| \tag{4.7}$$

$$fitness = C_{PERIOD} \times period\_error + C_{AMPLITUDE} \times amplitude\_error \tag{4.8}$$

4.1.3. Genetic Algorithm Results

The parameter boundaries are given to be between 0.001 and 5 for both capacity C and inductance L. We have 20 replications of each GA (i.e. *i*=1,…,20). Best (f_j^{min}), worst (f_j^{max}) and mean ($\bar{f}_j$) fitness values obtained from these 20 replications by the different GA

configurations are given in Table 4.1. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 2) vector composed of (L, C)

Table 4.1. Mean fitness value ($\bar{f}_j$), best fitness value (f_j^{min}) and associated best individual (b_j^{min}), and worst fitness value (f_j^{max}) and associated worst individual (b_j^{max}) of eight different GA configurations in 20 replications

	Mean fitness ($\bar{f}_j$)	Best fitness (f_j^{min}) (b_j^{min})	Worst fitness (f_j^{max}) (b_j^{max})
Base GA	0.00309	0.00116 (2.1833, 3.8522)	0.00999 (2.1366, 3.8593)
1. GA	0.03545	0.00116 (2.1833, 3.8522)	0.45993 (2.9549, 3.0991)
2. GA	0.12428	0.00157 (2.1832, 3.8519)	0.89962 (3.6197, 2.6369)
3. GA	0.12300	0.00138 (2.1832, 3.8529)	0.92817 (3.6986, 2.5831)
4. GA	0.04285	0.00116 (2.1832, 3.8522)	0.80814 (3.4569, 2.6625)
5. GA	0.31790	0.00871 (2.1796, 3.8614)	0.79869 (3.4758, 2.7123)
6. GA	0.00151	0.00117 (2.1832, 3.8522)	0.00444 (2.1816, 3.8756)
7. GA	0.00153	0.00118 (2.1832, 3.8522)	0.00307 (2.1821, 3.8617)

The fittest individual is b_{ij}^{min*}= (2.1833, 3.8522) with fitness f_{ij}^{min*}=0.00116. The L and C value that gives the worst fitness value, f_{ij}^{max*} =0.92817, among all runs is b_{ij}^{max*} = (3.6986, 2.5831). Figure 4.3 shows the behavior of these best and worst individuals.

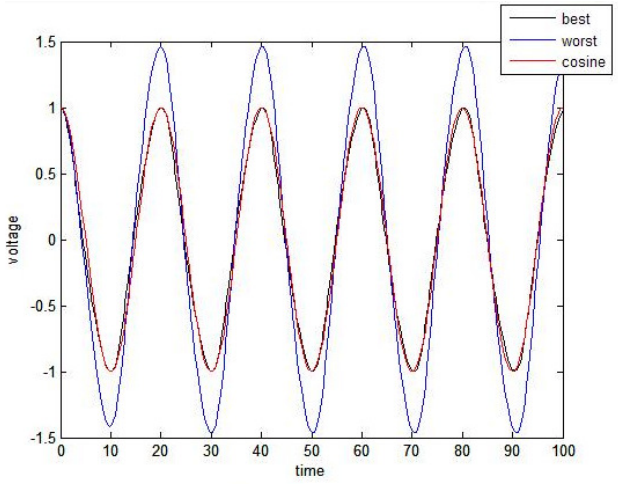

Figure 4.3. Behaviors with best (b_{ij}^{min*}) and worst (b_{ij}^{max*}) parameter sets among all runs

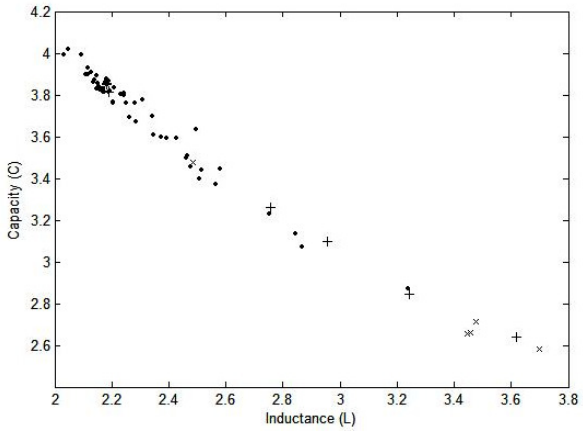

Figure 4.4. Plot of solutions provided by 160 GA runs with eight different configurations. ('+' and '×' indicate the results of replications yielding outlier solutions, replications 5 and 16, respectively)

When we look at Figure 4.3, we observe that the behavior of b_{ij}^{min*} is almost the same as the desired cosine wave, whereas the worst solution behavior has the same period with a period error less than 10^{-5}. Penalty for the amplitude error, which is 0.92817, makes almost the entire penalty (fitness value). Figure 4.4 shows that most of the time GA solutions lay around the b_{ij}^{min*} obtained. Note that b_{ij}^{max*} is also the farthest point away from the region where good solutions lie.

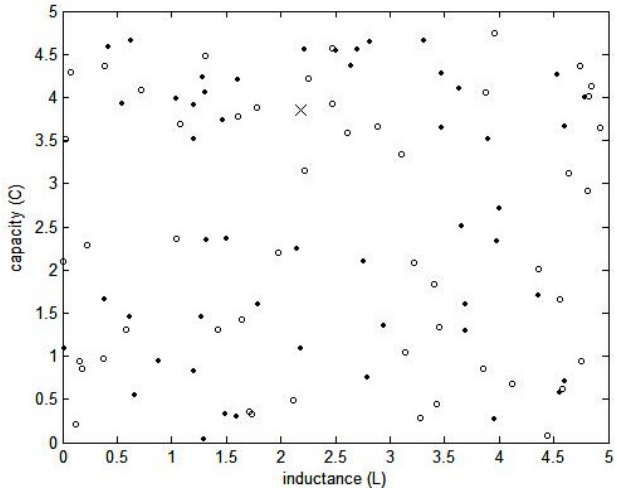

Figure 4.5. Initial population locations in replications 16 (●) and 11 (○). Best solution marked with '×'.

Figure 4.4 also indicates that the initial population is important for some of the GA configurations. Four of the five worst solutions in terms of fitness values are obtained by replication 16, each one with different configuration, which are 2nd, 3rd, 4th, and 5th configuration. The worst fitness values (f_j^{max}) of 3rd, 4th, and 5th GA configurations given on Table 4.1 are from replication 16. The second worst solution of 2nd configuration ((L, C) = (3.4472, 2.6577)) is also from replication 16 which has a fitness of 0.81173. Figure 4.5 reveals that, unlike replication 11, initial population created by replication 16 does not contain any points close to the region where best solutions lie. Hence, it takes longer for some of the GA configurations to obtain good solutions. Similar to replication 16, initial population created by replication 5 does not contain any points lying close to the final GA solutions provided in Figure 4.4 (see Figure 4.6). The worst solution (f_j^{max}) found by the 2nd configuration given on Table 4.1 is from replication 5. In addition, the second worst solution of 3rd configuration ((L, C) = (3.2414, 2.8448)) is also from replication 5 which has a fitness of 0.67353.

Based on these observations, we obtain two results. If we have prior knowledge about where good solutions may lie, we should use it in determining the initial population. Secondly, 2nd, 3rd and 5th configurations with their dependent performance on the initial set of individuals show that they are not reliable to find good solutions and likely to get stuck at local optima.

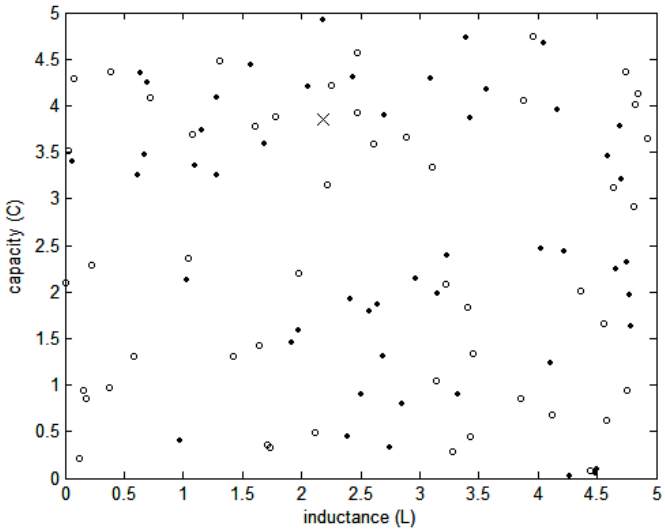

Figure 4.6. Initial population locations with replication 5 (●) and 11 (○). Best solution marked with '×'.

4.2. Control of Plate Angle with a Fan-motor (FM)

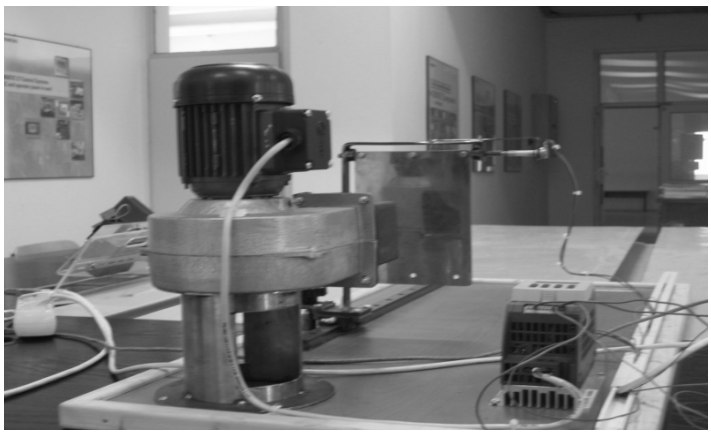

Figure 4.7. Fan-motor (FM) structure (Demir et al. 2005)

4.2.1. Model and the Control Structure

The system is composed of a plate hanged on a hinge and a fan motor directed towards the plate. The aim is to fix the plate at a desired angle by controlling the fan speed. Fan force on the plate is linearly proportional with the motor speed (W_M). A DC motor is

modeled in two sections, electric and mechanic. Electric part, which is controlled by the voltage (VC), drives the mechanic part and adjusts the motor speed. There is a discrete delay between the time that the motor speed is changed and the time plate is affected by that change due to the existing distance between the fan and plate. Here, we model this discrete delay as continuous by using a goal seeking formulation (see Figure 4.8). In Appendix A, we discuss what happens when we model the system with pure delay.

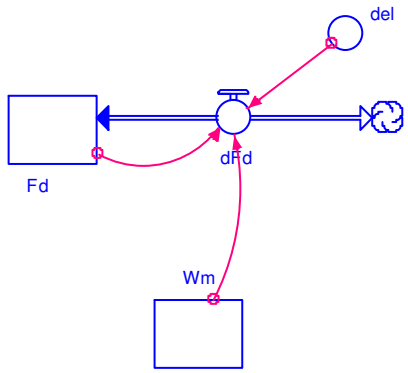

Figure 4.8. Fan force (F_d) follows the motor speed (W_M) after a delay due to the distance between the plate and motor

The control system works as a regular feedback system. The angle of the plate, which corresponds, to the final voltage (V_F) is continuously measured. Then the error or the difference between the measured and desired voltage (V_R) is calculated. A control signal is generated by a controller design using the error value (V_E), which has corresponding motor or fan speed. Below equation gives the control structure of the model.

$$V_C = K_P \cdot V_E + K_I \cdot \int V_E dt \tag{4.9}$$

The delay from the time the angle is measured to the time it affects the fan speed is small and can be omitted comparing it with the delay caused by the distance between the fan and plate. Both the measured and desired angles correspond to a certain voltage in the model; hence, we omitted the conversion from voltage to angle, which is only a multiplication by a scalar constant. Thus, we define a desired voltage rather than a desired angle and show all of our results in terms of voltage in the sensors. The Stella model is given below in Figure 4.9.

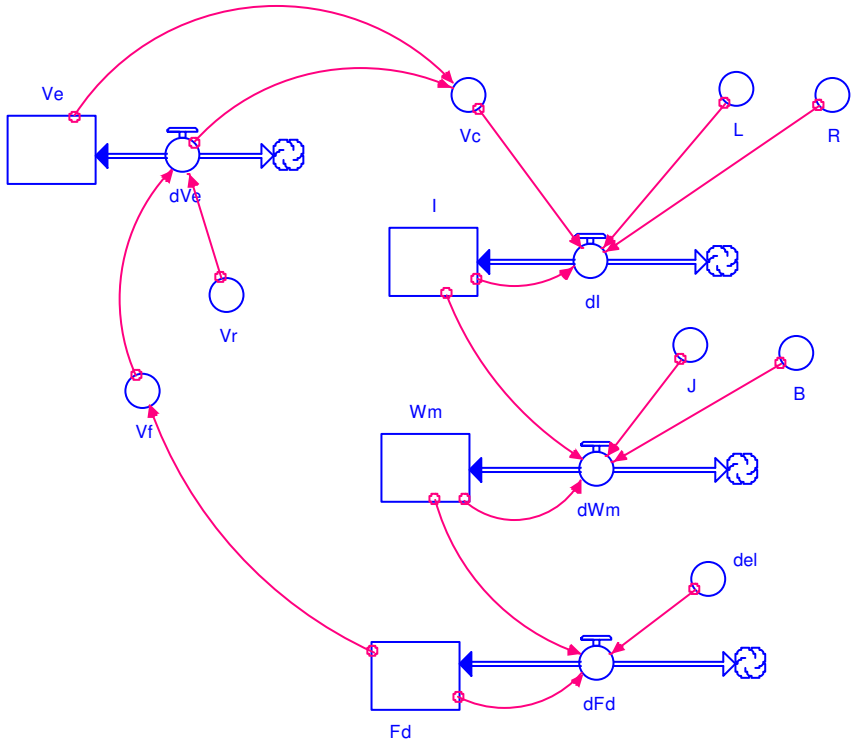

Figure 4.9. Fan-Motor model with PI control structure

The equations of the model in Figure 4.9 are as given below:

$$\dot{I} = -\frac{R}{L} \times I + \frac{V_C}{L} \tag{4.10}$$

$$\dot{W}_M = \frac{-B \times W_M + I}{J} \tag{4.11}$$

$$\dot{F}_d = \frac{W_M - F_d}{delay} \tag{4.12}$$

$$\dot{V}_e = V_f - V_r \tag{4.13}$$

For this system, besides the PI controller parameters K_P and K_I, the DC motor parameter L is also a design (policy) parameter. Initial voltage value is zero when the plate is vertical without any fan force. At time zero, the desired voltage is changed to 0.5. Figures 4.10-4.13 show different behaviors with variations on these parameters. In this model, first we look at the effect of proportional gain (i.e. dVe in Figure 4.9 or K_P) without any Integration gain (i.e. Ve in Figure 4.9 with K_I =0) (see Figure 4.10). Figure 4.11

indicates the effect of inductance (L) on behavior. Figures 4.12 and 4.13 demonstrate the effect of K_I and displays that inductance gain does not have influence on the pattern. In other words, if the pattern is growing oscillations, it cannot be changed by adjusting integration gain, K_I. These figures demonstrate that the pattern behavior is dependant on the proportional gain value, K_P.

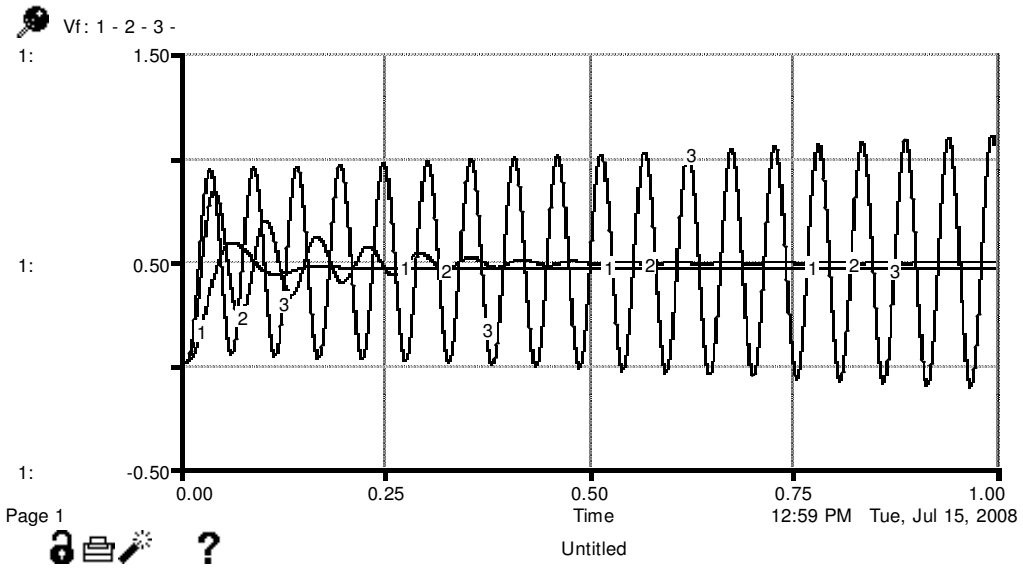

Figure 4.10. Final voltage value (V_F) with 1: K_P=1, 2: K_P=3, and 3: K_P=4.5 with L = 0.03 and K_I=0

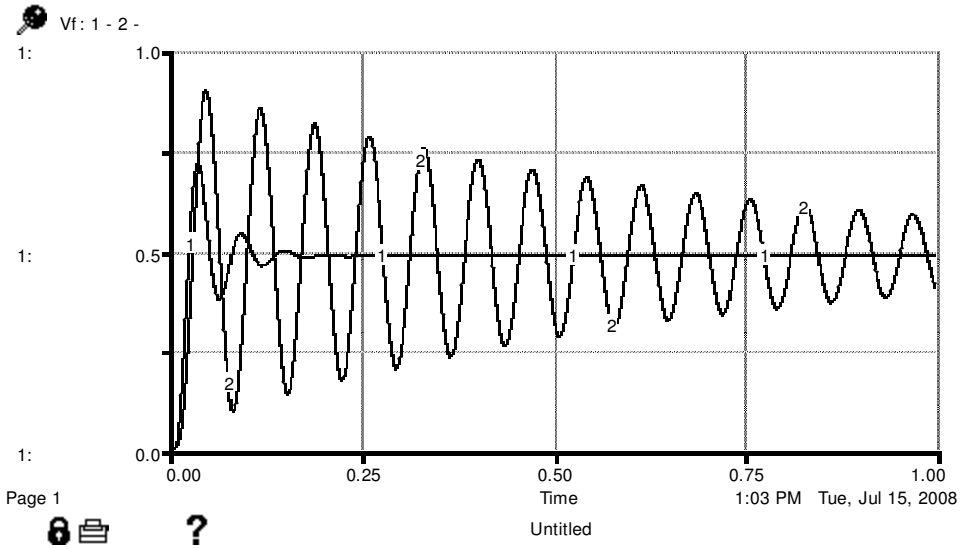

Figure 4.11. Final voltage value (V_F) with 1: L=0.01, and 2: L=0.05 with K_P=3 and K_I=0

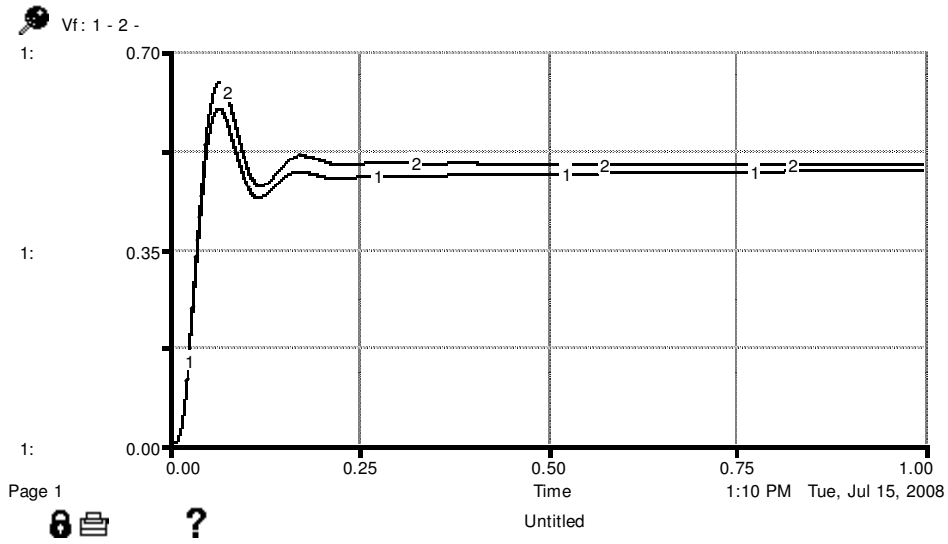

Figure 4.12. Final voltage value (V_F) with 1: K_I=1, and 2: K_I=5 with K_P=1 and L=0.03

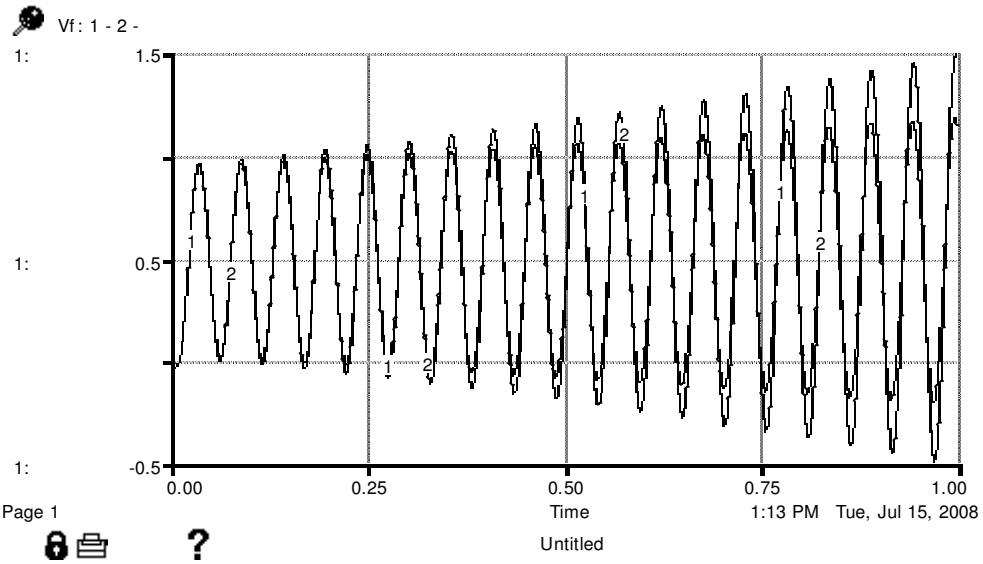

Figure 4.13. Final voltage value (V_F) with 1: K_I=1, and 2: K_I=5 with K_P=4.5 and L=0.03.

4.2.2. Fitness Function for the fan-motor model

The parameter search in this problem has the objectives to stabilize at the desired voltage (V^*) as quickly as possible without oscillation. In the fitness function, these objectives are tried to be reached by penalizing settling time (t_s), overshoot, and steady

state error (e_{ss}). Settling time is defined as the time the system variable goes into ±5 per cent band of the settled voltage value (V^S) and never goes out. Time horizon is the settling time if the system fails to settle. We define steady state error (e_{ss}) as the difference between V^S and V^*.

$$e_{SS} = \left| V^S - V^* \right| \tag{4.14}$$

Overshoot is the difference between maximum peak of the voltage (V^{MAX}) and V^S. For the fitness function, V^{MAX} is always greater than or equal to the settled value so the penalty is positive.

$$Overshoot = V^{MAX} - V^S \tag{4.15}$$

The fitness function is the weighted sum of t_s, overshoot, and e_{ss}. The weights of t_s (Pts), e_{ss} (Pess), and overshoot (Po) are all equal to 10.

$$Fitness = Po \times Overshoot + Pess \times e_{SS} + Pts \times t_S \tag{4.16}$$

4.2.3. Genetic Algorithm Results

The time horizon for the search is set to one second. K_P can have values between 0.2 and 10. The boundaries of K_I are 0.2 and 5. L is between 0.01 and 0.05. We have 20 replications of each GA. Best ($f_j^{\min}$), worst ($f_j^{\max}$) and mean ($\bar{f}_j$) fitness values obtained from these 20 replications by the different GA configurations are given in Table 4.2. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 3) vector composed of (K_P, K_I, L).

Table 4.2. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 20 replications

	Mean fitness ($\bar{f}_j$)	Best fitness ($f_j^{\min}$) ($b_j^{\min}$)	Worst fitness ($f_j^{\max}$) ($b_j^{\max}$)
Base GA	0.718034	0.71618 (0.54372, 1.5071, 0.01)	0.73504 (0.57804, 1.3525, 0.010009)
1. GA	0.764377	0.71618 (0.54372, 1.5071, 0.01)	1.0208 (0.36036, 1.1389, 0.05)
2. GA	0.940417	0.72859 (0.53119, 1.5018, 0.011145)	1.4612 (0.61479, 2.5453, 0.010076)
3. GA	0.837761	0.71622 (0.54378, 1.5067, 0.01)	1.0369 (0.62896, 0.755, 0.012463)
4. GA	0.748111	0.71618 (0.54372, 1.5071, 0.01)	1.0208 (0.36036, 1.1389, 0.05)
5. GA	1.098066	0.74701 (0.55465, 1.4427, 0.011223)	2.6996 (1.0686, 4.6813, 0.011156)
6. GA	0.721205	0.71619 (0.54373, 1.507, 0.01)	0.79861 (0.614, 1.2825, 0.01)
7. GA	0.716254	0.71621 (0.54373, 1.5071, 0.01)	0.78452 (0.51316, 1.3497, 0.014628)

The fittest individual is $b_{ij}^{\min *}$= (0.54372, 1.5071, 0.01) with fitness $f_{ij}^{\min *}$=0.71618. The parameter values that give worst fitness value, $f_{ij}^{\max *}$=2.6996, among all runs is $b_{ij}^{\max *}$= (1.0686, 4.6813, 0.011156). Figure 4.14 shows the behavior of these best and worst fit parameters.

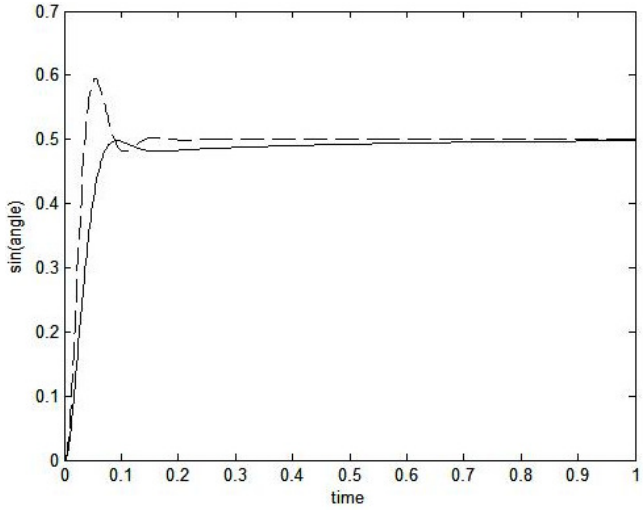

Figure 4.14. Behavior with best ($b_{ij}^{\min *}$) (solid line) and worst ($b_{ij}^{\max *}$) (dashed line) parameter sets among all runs

Table 4.3. Settling time, overshoot, and steady state error values of best (b_{ij}^{min*}) and worst (b_{ij}^{max*}) parameter sets

	b_{ij}^{min*}			b_{ij}^{max*}		
Fitness Value	0.71618			2.6996		
(K_P, K_I, L)	0.54372 1.5071 0.01			1.0686 4.6813 0.011156		
	t_s = 0.0673	O= 1.33x10^{-4}	e_{ss} = 0.0042	t_s = 0.0798	O = 0.1901	e_{ss} = 1.43x10^{-5}

The worst solution because of all GA runs is inferior to the best solution found in terms of overshoot and settling time (see Table 4.3). This worst solution (f_j^{max*}) is obtained with GA5 where single point crossover is used. The mean fitness ($\bar{f}_j$) of solutions of GA5 is also the highest. We conclude that heuristic crossover works better than the single point crossover in this search space.

In terms of the mean fitness, GA7 is the best configuration. This might mean that Gaussian mutation is better at scanning the search space. It might also be because Gaussian mutation is always applied to obtain a previously decided portion of the next generation whereas uniform mutation is applied to each gene of a parent with probability, 0.05. When we look at the best fitness ($f_{j=7}^{min}$) among 20 replications of GA7, it never converges to the best fitness value found (0.71618). This means the algorithm needs further iterations to converge. The same logic applies to GA6 with small crossover ratio; it takes longer time for the population to converge. Figure 4.15 shows the number of times the best solution is obtained by different GA configurations. Although GA4 obtained the best solution (b_{ij}^{min*}) highest number of times together with base GA, GA4 is fourth on mean fitness ranking. Since the number of elites is equal to five in GA4, it has a potential to get stuck on local optima. The worst solution obtained by GA4, $b_{j=4}^{max}$= (0.36036, 1.1389, 0.05), shows that the GA has converged to a local optimum and is not converging to the best solution found, b_{ij}^{min*}, since L is at its maximum value at the local optimum whereas the best solution obtained has L at its minimum.

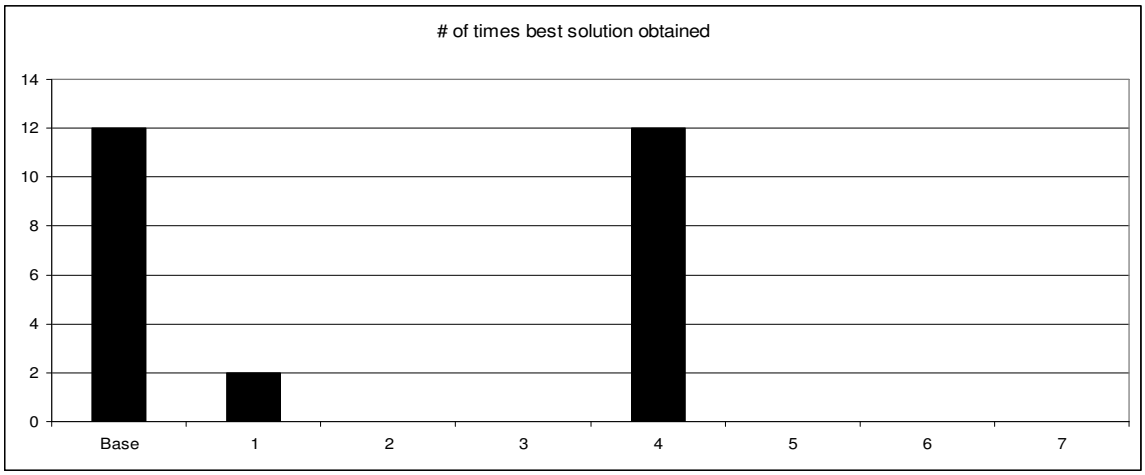

Figure 4.15. Number of times best fitness $f_{ij}^{\min *}$ = (0.71618) is obtained in 20 replications by different configurations

Figure 4.16 demonstrates that in terms of behavior pattern best solution ($b_{j=4}^{\min}$) and worst solution ($b_{j=4}^{\max}$) obtained by all runs with GA4 have the same dynamic pattern, damping oscillations. The worst solution of GA4 has a longer settling time.

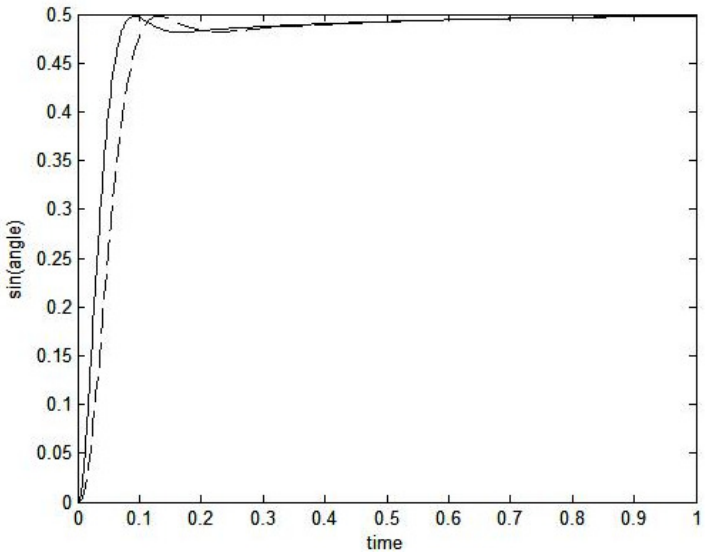

Figure 4.16. $b_{j=4}^{\min}$ = (0.54372, 1.5071, 0.01) (solid line) and $b_{j=4}^{\max}$ =(0.36036, 1.1389, 0.05) (dashed line) given by GA4 in 20 replications

In terms of number of times best solution obtained and mean fitness ($\bar{f}_j$) combined, base GA is the best configuration for this search space. GA6 and GA7 increases the chance of obtaining good solutions as they explore the search space well but they need longer time to converge.

4.3. Magnetic Ball Suspension System (MBS)

4.3.1. Model and Control Structure

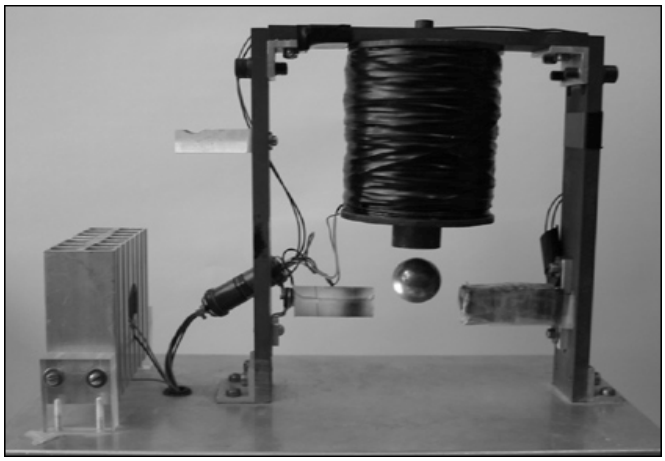

Figure 4.17. Picture of a MBS system (Online Image)

This system, also known as Magnetic Levitation System, is very popular and many studies on this system exist in control theory literature. The system here is a simpler version (single dimensional) of control of the ball via a magnetic force in two or three-dimensional space. The equations of the system are given below:

$$M\ddot{x} = Mg - \frac{I^2}{x} \tag{4.17}$$

$$L\dot{I} = V - Ri \tag{4.18}$$

The system is nonlinear since pulling force of the magnet is proportional with the square of the current (I), and inversely proportional with the displacement (x). The objective of the system is to control the vertical position of the ball by adjusting the current

in the electromagnet through the input voltage (V). The velocity is defined to be positive when it moves away from the magnet and negative when it moves towards it. In our model, the magnet can only pull the ball towards itself and cannot push it away. The voltage (V) is non-negative. Hence, the ball can only move downwards with gravity (g). This is done by the use of a simple IF-ELSE formulation. Sensors read the vertical position of the ball. Figure 4.17 shows the picture of a real MBS system. The picture shows that the ball can only be controlled when the ball is between the optical sensors. Hence, the ball is not allowed to make large oscillations in real applications.

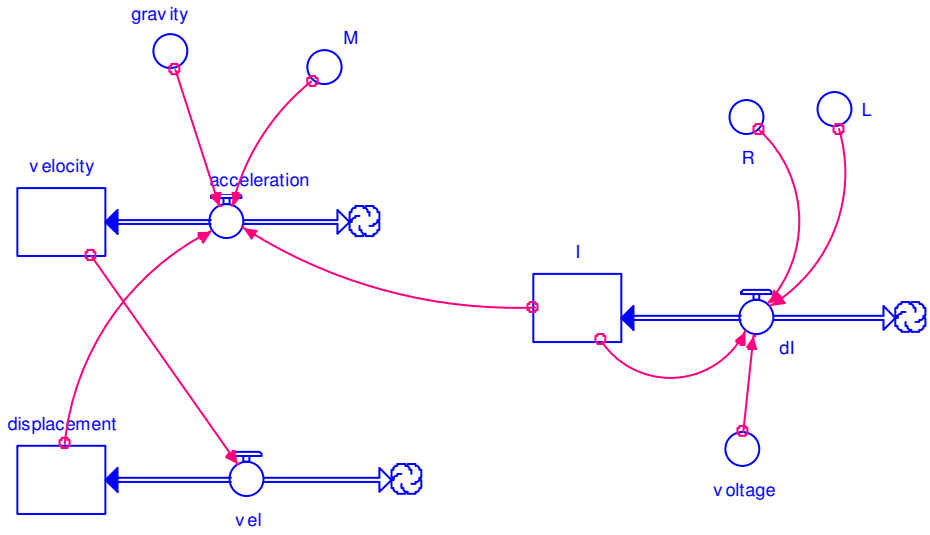

Figure 4.18. Stock-Flow diagram for the MBS model

Without the feedback structure, the system is as shown in Figure 4.18. The system has no goal defined such as the desired displacement. In all of the MBS models discussed, the following values will be used:

Weight of the ball (M) = 1g
Gravity = 10 m/s^2
Resistance (R) = 5 Ω
Inductance (L) = 0.002H
Initial displacement = 10cm
Initial velocity = 0 m/s

Initial current = 10 A

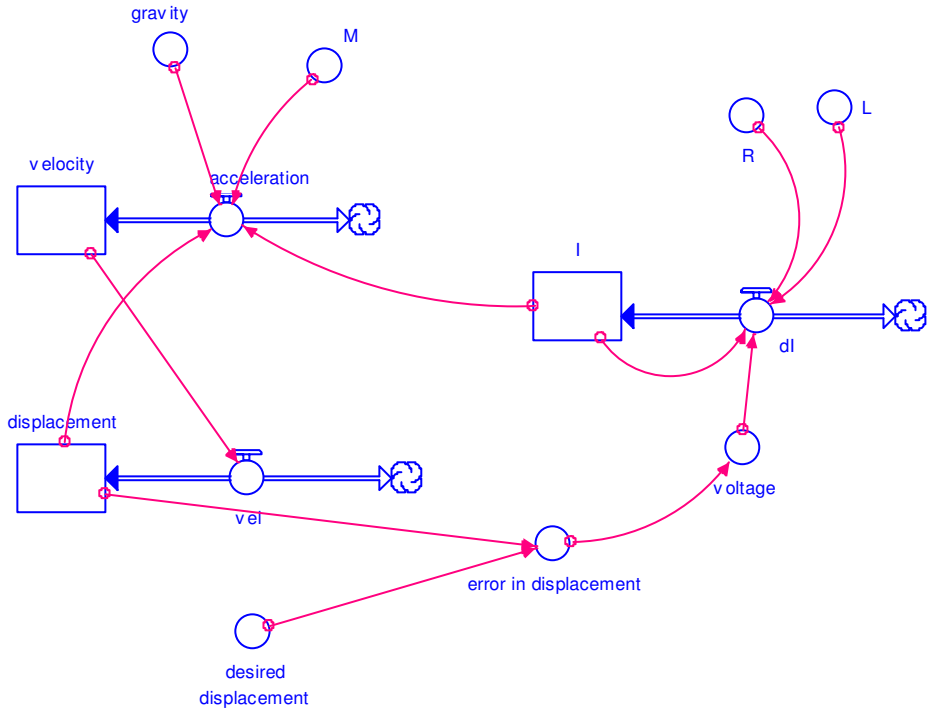

Figure 4.19. Stock-flow diagram for the MBS with P control.

Figure 4.19 gives the MBS stock-flow diagram with basic feedback structure, which is also known as proportional (P) control. The additional equations of feedback structure are as given below:

$$desired\_disp = 9 \tag{4.19}$$

$$error\_in\_disp = disp - desired\_disp \tag{4.20}$$

$$V = K_P \times error\_in\_disp \tag{4.21}$$

Desired displacement is nine cm. Figure 4.20 gives the behavior when K_P=1. As seen in Figure 4.20, the control signal is not strong enough to counteract the force of gravity.

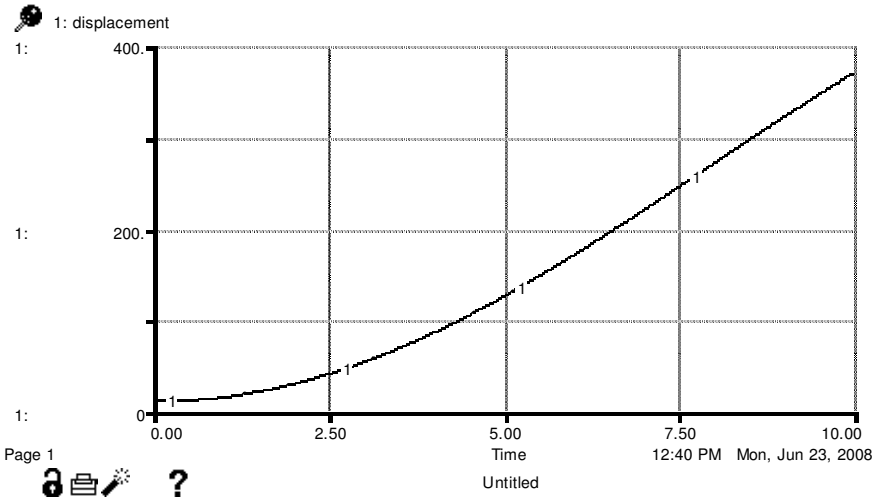

Figure 4.20. Displacement behavior with P control and K_P=1

In Figure 4.21, when the control signal is ten times the error, the system oscillates approximately between 10 cm and 20 cm. As we increase proportional gain (K_P), the amplitude of the oscillation decrease but it is not feasible to increase the K_P to large numbers as the control voltage begins to grow to unrealistic numbers. Hence, we conclude that proportional control is not enough to stabilize the system.

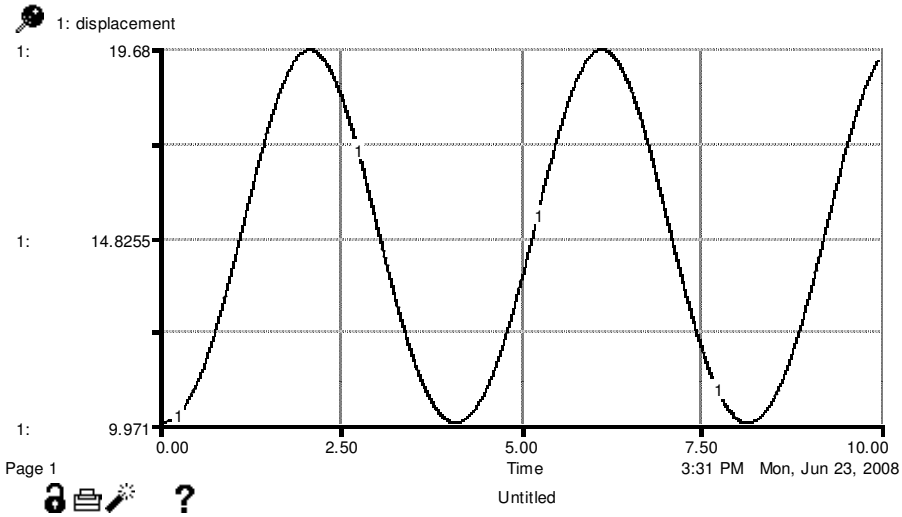

Figure 4.21. Displacement behavior with P control and K_P=10

Secondly, we add on an integrator control structure which accumulates the total error and tries to correct it in time (see Figure 4.22).

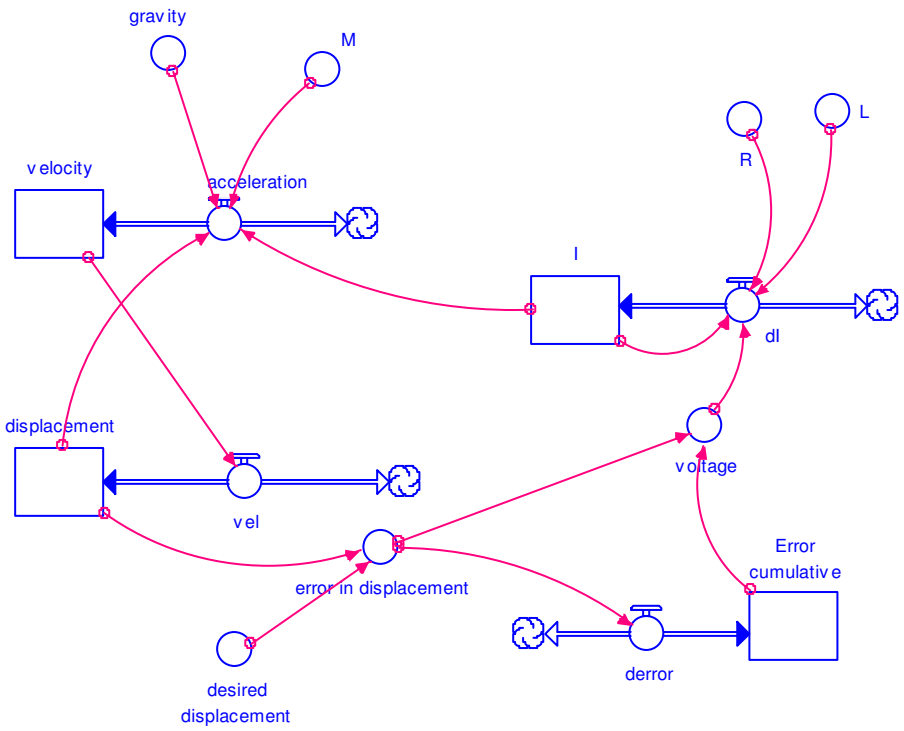

Figure 4.22. Stock-flow diagram for the MBS with PI control

The additional equations to the P control are as given below:

$$derror \quad = error \quad \_ in \_ disp \tag{4.22}$$

$$error\_cumulative\ (t) = error\_cumulative\ (t - dt) + derror \times dt \tag{4.23}$$

$$V = K_P \times error\_in\_disp + K_I \times error\_cumulative \tag{4.24}$$

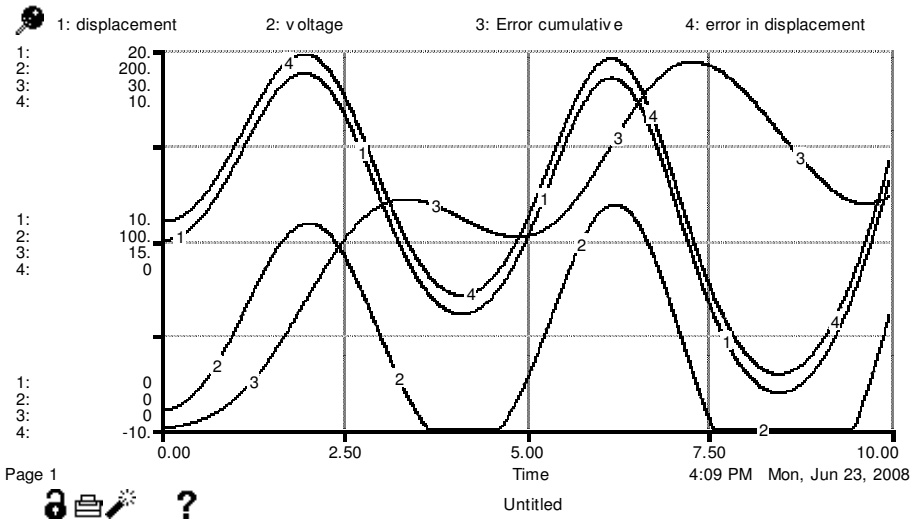

Figure 4.23. Displacement, voltage, error accumulation and error in displacement behavior with PI control (K_P=10 and K_I=1)

The PI control structure is also not enough to stabilize the system (see Figure 4.23). In fact, it can cause the system to show growing oscillations as it tries to correct the existing cumulative error. Hence, we conclude that none of these two classical control structures discussed, namely P and PI, are enough to stabilize the system. The PI control worked on the FM model but for the MBS system, we need an additional control structure that acts on the change in error. In other words, the PI control structure does not recognize at which speed the error in displacement is decreasing or increasing. This leads to oscillations in displacement. In control theory, the structure that considers the change in error is called derivative control. However, in SD modeling, it is not valid and possible to use the instantaneous derivative of a variable. Instead, we use trend functions. In Figure 4.24, we give the MBS system with PID control structure. Derivative structure calculates the trend of the error. Additional equations of the model are given below:

$$Change\_in\_average = \frac{(error\_in\_disp - Average\_error)}{Averaging\_time} \tag{4.25}$$

$$Average\_error(t) = Average\_error(t - dt) + Change\_in\_average \times dt \tag{4.26}$$

$$Trend\_in\_error = \frac{(error\_in\_disp - Average\_error)}{Averaging\_time} \tag{4.27}$$

$$V = K_P \times error\_in\_displacement + K_I \times error\_cumulative + K_D \times trend\_in\_error \tag{4.28}$$

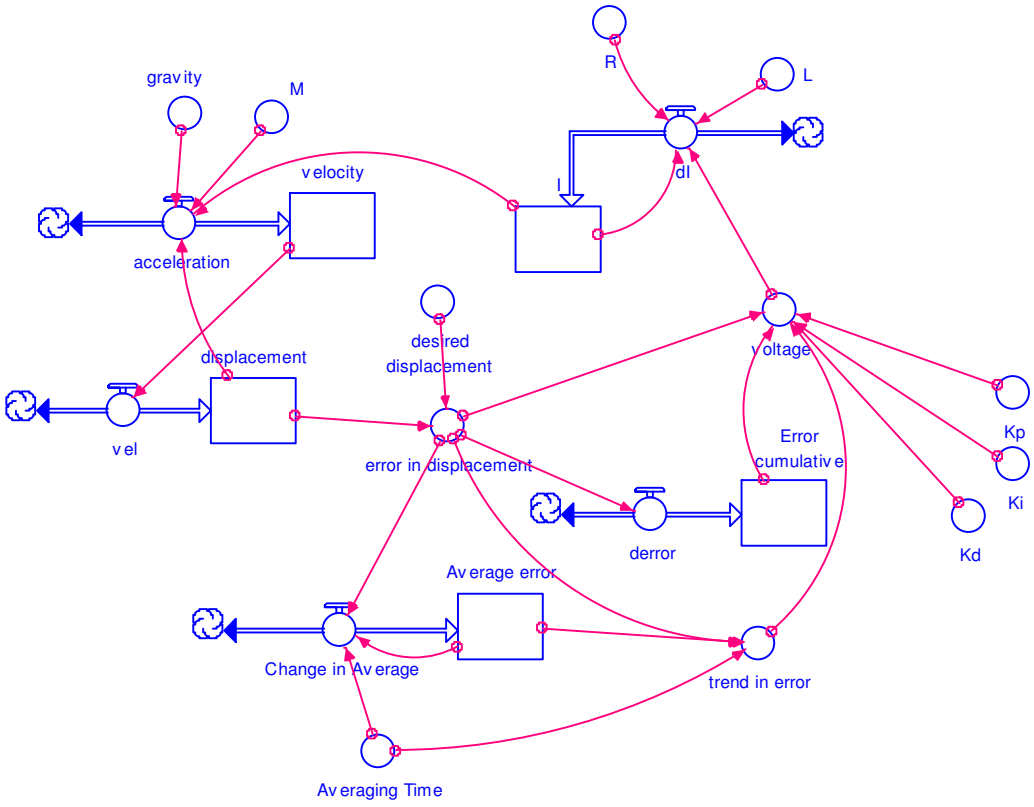

Figure 4.24. Stock-flow diagram for the MBS with PID control

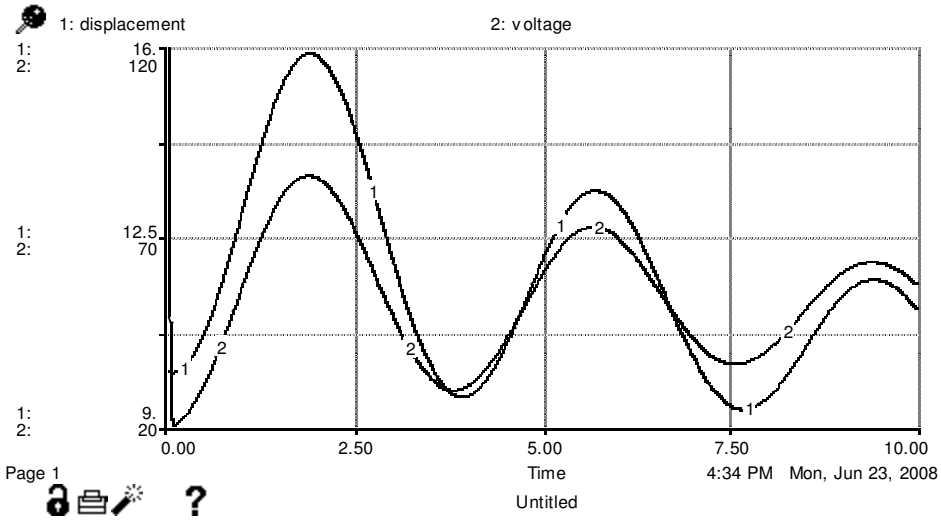

Figure 4.25. The displacement and voltage behavior with PID control (K_P=10, K_I=1 and K_D=1)

Figure 4.25 demonstrates that the system might show declining oscillations with certain parameter values of K_P, K_I, and K_D. This shows that adjusting control signal based on the change in the error is critical for MBS system. In summary, if the error is growing fast then the control signal should respond to it. In addition, when the error is decreasing from a large value, the control signal should take that rate of change in error into account.

The averaging time of the trend in error is 0.01 which is ten times the DT=0.001. For the trend function to behave close to instantaneous derivative, the averaging time should be as small as possible so that the response of the controller is sensitive to the changes in the error. Figure 4.26 shows that the behavior of displacement degrades when averaging time is one. However, the difference in terms of performance between the behaviors when averaging time is 0.1 and 0.002 is not that striking.

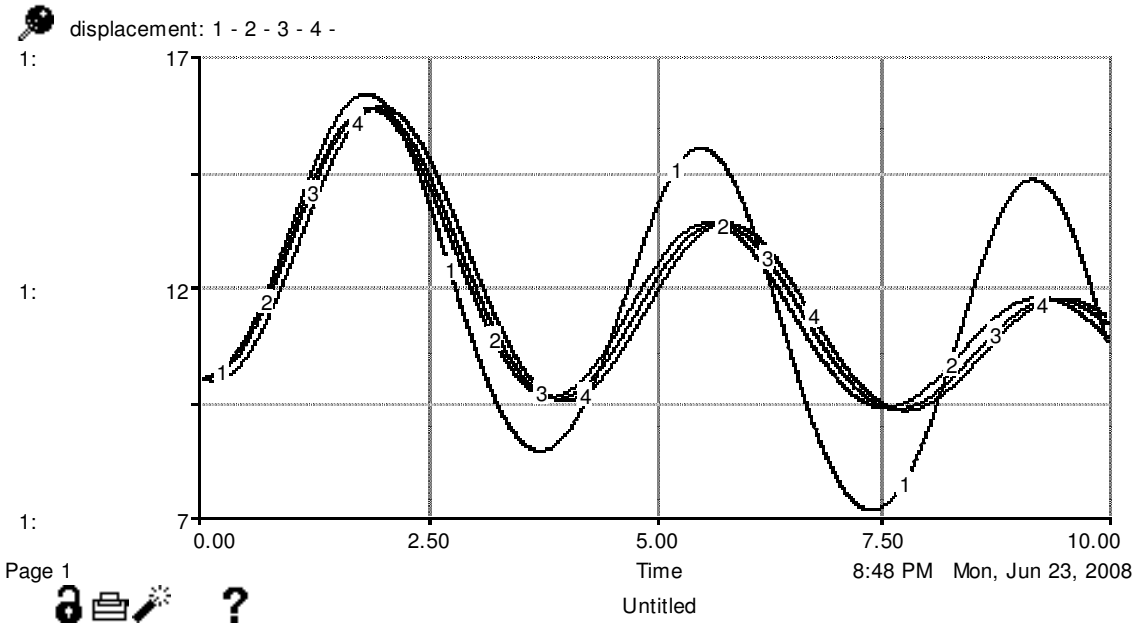

Figure 4.26. Behavior of displacement with averaging time equal to 1 (1), 0.1 (2), 0.01 (3), and 0.002 (4)

Even with PID structure, certain K_P, K_I and K_D values can provide unstable or undesired behavior (see Figure 4.27). In Figure 4.27, the ball hits the magnet causing the simulation to give a division by zero error. Hence, we need parameter search to find the weights that satisfy our goals.

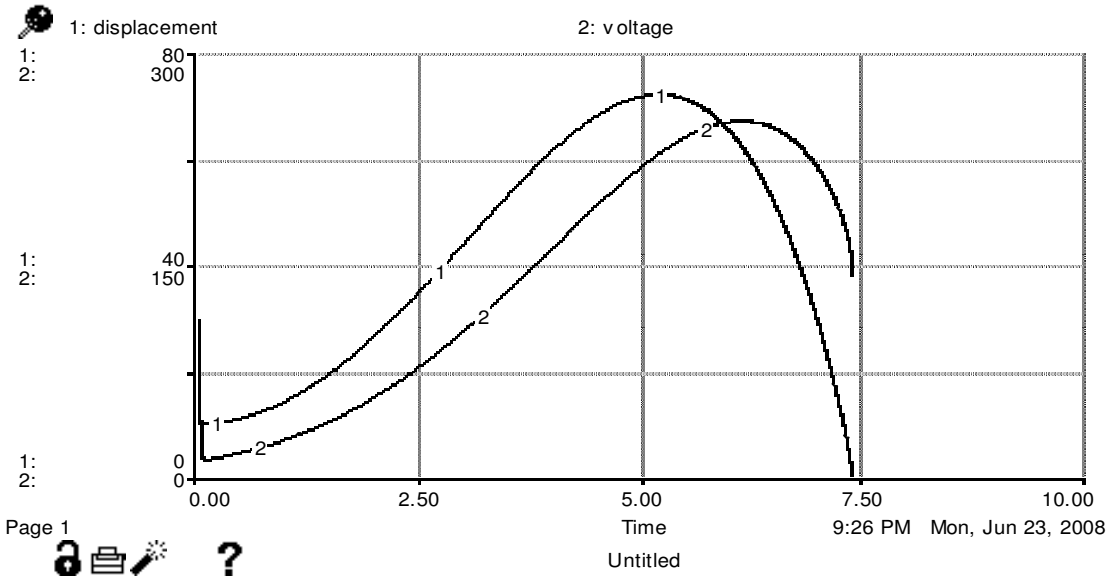

Figure 4.27. Displacement and voltage behavior with PID control (K_P=1, K_I=1 and K_D=1)

4.3.2. Fitness Function for the MBS model

The parameter search in this problem is similar to the fan-motor model. It has the objectives to stabilize at the desired displacement (D^*) as quickly as possible without much oscillation. In the fitness function, these objectives are tried to be reached by penalizing settling time (t_s), and steady state error (e_{ss}). Settling time is defined as the time the system variable goes into ±5% band of the settled displacement value (D^S) and never goes out. Settling time is normalized by the simulation time horizon. Time horizon is the settling time if the system fails to settle. Therefore, the penalty for settling time is at most equal to one. We define steady state error (e_{ss}) as the difference between D^S and D^* normalized by D^S.

$$e_{SS} = \frac{\left| D^S - D^* \right|}{D^S} \tag{4.29}$$

The fitness function is the weighted sum of t_s, and e_{ss}.

$$Fitness = Pess \times e_{SS} + Pts \times t_S \tag{4.30}$$

We choose Pess and Pts, 10 and 1, respectively. For this system, the controller parameters K_P, K_I and K_D are design (policy) parameters.

4.3.3. Genetic Algorithm Results

The K_P can have values between 0 and 40. The boundaries of K_I and K_D are 0 and 10. We have 10 replications of each GA. Best ($f_j^{\min}$), worst ($f_j^{\max}$) and mean ($\bar{f}_j$) fitness values obtained from these 10 replications by the different GA configurations are given in Table 4.4. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 3) vector composed of (K_P, K_I, K_D).

Table 4.4. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 10 replications

	Mean fitness ($\bar{f}_j$)	Best fitness ($f_j^{\min}$) ($b_j^{\min}$)	Worst fitness ($f_j^{\max}$) ($b_j^{\max}$)
Base GA	0.189992	0.18996 (34.315, 5.1017, 3.0575)	0.1902 (34.709, 5.1473, 3.1782)
1. GA	0.192583	0.19017 (34.541, 5.1328, 3.1635)	0.19636 (32.075, 5.1319, 3.6168)
2. GA	0.20305	0.19216 (33.698, 5.1252, 3.3759)	0.23827 (19.471, 5.4092, 3.331)
3. GA	0.198537	0.19207 (34.332, 5.1678, 3.4123)	0.20839 (30.411, 5.212, 4.0067)
4. GA	0.191422	0.18996 (34.292, 5.0995, 3.054)	0.19399 (32.64, 5.1165, 3.5045)
5. GA	0.197838	0.19106 (33.524, 5.0519, 3.1396)	0.2094 (28.939, 5.4167, 4.1289)
6. GA	0.190006	0.18997 (34.412, 5.1132, 3.0855)	0.19008 (34.058, 5.0733, 2.9937)
7. GA	0.190332	0.18998 (34.301, 5.0998, 3.0541)	0.20621 (8.7947, 4.0343, 0.99443)

The fittest individual is $b_{ij}^{\min *}$= (34.315, 5.1017, 3.0575) with fitness $f_{ij}^{\min *}$=0.18996. The parameter values that give worst fitness value, $f_{ij}^{\max *}$=0.23827, among all runs are $b_{ij}^{\max *}$= (19.471, 5.4092, 3.331). Figure 4.28 shows the behavior of these best and worst fit parameters.

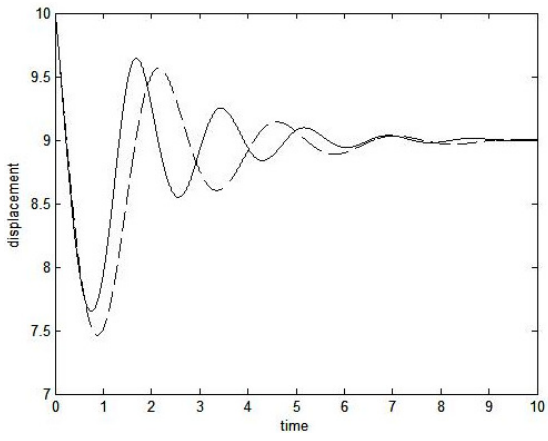

Figure 4.28. Behavior with best ($b_{ij}^{\min *}$) (solid line) and worst ($b_{ij}^{\max *}$) (dashed line) parameter sets among all runs

Table 4.5. Settling time, overshoot, and steady state error values of best (b_{ij}^{min*}) and worst (b_{ij}^{max*}) parameter sets

	b_{ij}^{min*}		b_{ij}^{max*}	
Fitness Value	0.18996		0.23827	
(K_P, K_I, K_D)	34.315 5.1017 3.0575		19.471 5.4092 3.331	
	t_s = 1.8996	e_{ss} = 0	t_s = 2.3827	e_{ss} = 0

The worst solution among all runs is inferior to the best solution found in terms of settling time (see Table 4.5) since both of them settle to the desired displacement. In all of the runs, solutions (b_{ij}) converge to a value around 34 for K_P, around 5 for K_I and between 3 and 4 for K_D except the worst solution obtained by GA7, $b_{j=7}^{max}$=(8.7947, 4.0343, 0.99443). The settling time is 1.55 and the difference between D^S and D^* is 0.05 for this solution. The behavior of the solution is given in Figure 4.29. This solution shows different behavior than the solutions that converge around the point (34.301, 5.0998, 3.0541). Even though the ball continues to oscillate, it has a small settling time since the oscillations are inside the 5 per cent band. However, these oscillations do not seem to dampen in time.

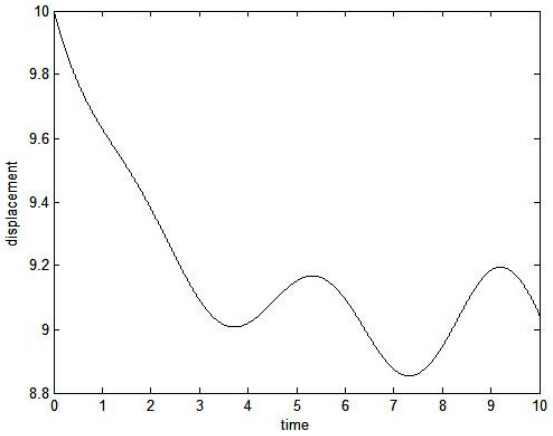

Figure 4.29. Behavior with $b_{j=7}^{max}$= (8.7947, 4.0343, 0.99443) in time horizon 10 seconds

In fact, Figure 4.30 shows that this solution fails to control the ball and results in growing oscillations. This is the only solution with this kind of behavior among all 80 GA runs. Figure 4.30 indicates that time horizon is insufficient. To minimize the possibility of obtaining this kind of solutions, the time horizon must be increased, or the allowed

oscillation band used for calculating settling time can be decreased or the weight on steady state error can be increased.

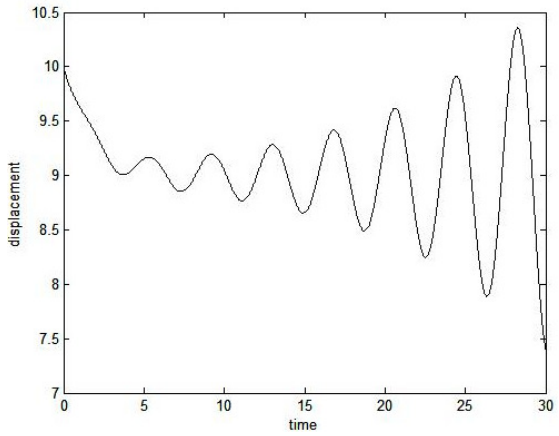

Figure 4.30. Behavior of $b_{j=7}^{\max}$= (8.7947, 4.0343, 0.99443) in time horizon 30 seconds

Except one result, all of the GAs results are able to obtain the same dynamic pattern. In terms of mean fitness and number of times the best solution obtained, base GA is the best configuration.

4.4. Predator-Prey Model

4.4.1. Model

The predator-prey is a generic system dynamics model. The equations of the model are as given:

$$\dot{prey} = \alpha \times prey - \beta \times prey \times predator \tag{4.31}$$

$$\dot{predator} = -\gamma \times predator + \delta \times prey \times predator \tag{4.32}$$

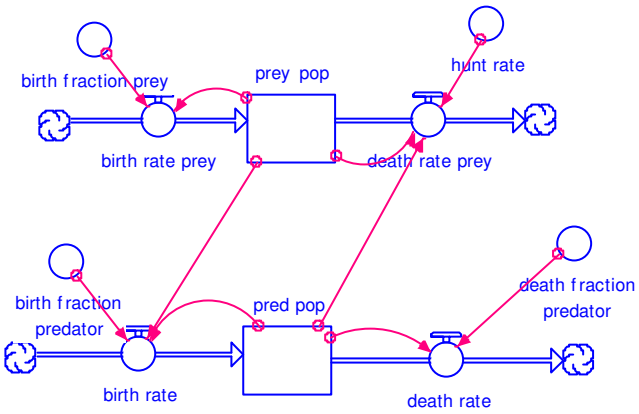

Figure 4.31. Stock-Flow model for the predator-prey model

α is the birth fraction of the prey. β denotes the hunting rate of the predator population which corresponds to the death rate of the prey population. γ is the death fraction of the predator population, standing for the percentage of the population dying per unit time step. Δ is the birth fraction of the predator.

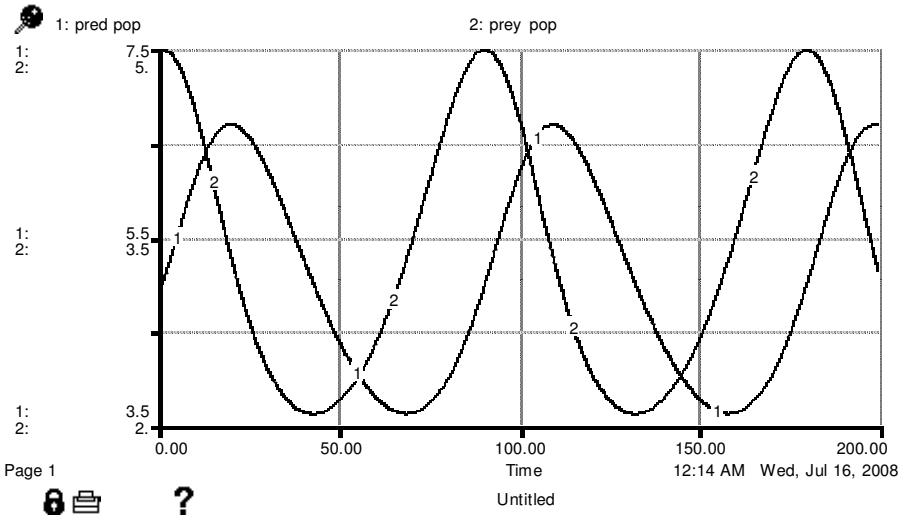

Figure 4.32. Behavior of populations with α=0.1, β= 0.02, γ= 0.05, δ= 0.015, and initial populations $Predator_o$= 5, and $Prey_o$= 5

4.4.2. Fitness Function for the Predator-Prey Model

The chosen aim in this ecological system is to keep the population levels at their initial levels. In other words, we are looking for the parameter values that satisfy

equilibrium. We choose this model as a test function since the global optimum is known. The equilibrium analysis is given below:

$$\dot{prey} = \alpha \times prey_O - \beta \times prey_O \times predator_O = 0 \tag{4.33}$$

$$predator_O = \frac{\alpha}{\beta} \tag{4.34}$$

$$\dot{predator} = -\gamma \times predator_O + \delta \times prey_O \times predator_O = 0 \tag{4.35}$$

$$prey_O = \frac{\gamma}{\delta} \tag{4.36}$$

The fitness function used to obtain the equilibrium calculates the mean of populations and compares them with the initial or desired population levels.

$$mean\_error\_prey = \left| mean(prey) - prey_O \right| \tag{4.37}$$

$$mean\_error\_predator = \left| mean(predator) - predator_O \right| \tag{4.38}$$

The fitness is the sum of mean errors multiplied by a constant.

$$fitness = Const \times (mean\_error\_prey + mean\_error\_predator) \tag{4.39}$$

4.4.3. Genetic Algorithm Results

The policy parameters are determined as α, β, γ. Their ranges are between zero and one. If any of the variables become negative, the value is moved 0.001 since negative is not meaningful for any of these parameters. δ is fixed at 0.1. The initial population levels are five for both of the species. For the fitness function, the constant multiplier, *Const*, is 100. The global optimum is α/β=5 and γ/δ=5. Best ($f_j^{\min}$), worst ($f_j^{\max}$) and mean ($\bar{f}_j$) fitness values obtained from these 20 replications by the different GA configurations are given in Table 4.6. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 3) vector composed of (α, β, γ).

Table 4.6. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 20 replications

	Mean fitness ($\bar{f}_j$)	Best fitness ($f_j^{\min}$) ($b_j^{\min}$)	Worst fitness ($f_j^{\max}$) ($b_j^{\max}$)
Base GA	0.2593	0 (1, 0.2, 0.5)	0.91725 (0.001, 0.00004, 0.49915)
1. GA	0.0638	0 (1, 0.2, 0.5)	0.99872 (0.001, 0.001, 0.49933)
2. GA	5.9508	0.01011 (0.4175, 0.083499, 0.5)	24.121 (0.8308, 0.15409, 0.4626)
3. GA	7.5798	0.16864 (0.005, 0.001, 0.49933)	34.91 (0.92111, 0.19206, 0.63994)
4. GA	0.1307	0 (0.84376, 0.16875, 0.5)	0.54847 (0.45062, 0.09056, 0.49964)
5. GA	10.8014	1.2343 (0.39076, 0.078555, 0.497)	30.931 (0.38439, 0.078847, 0.39925)
6. GA	0.239	0.0000215 (0.005, 0.001, 0.5)	0.99873 (0.001, 0.001, 0.49933)
7. GA	0.3836	0.00168 (0.99995, 0.19999, 0.5)	0.99875 (0.001, 0.001, 0.49933)

It is seen that only three of the configurations obtain solutions that converge to the global optimum. Figure 4.34 shows that base GA, and GA1 reached the global three times out of 20 replications. GA4 has only reached the optimum once. Hence, GA1 is the better configuration since it also has the smallest mean fitness ($\bar{f}_j$). We might conclude that in this system, the rank scaling element improves performance. Table 4.6 shows that the worst configuration is GA5 with single point crossover in terms of $\bar{f}_j$.

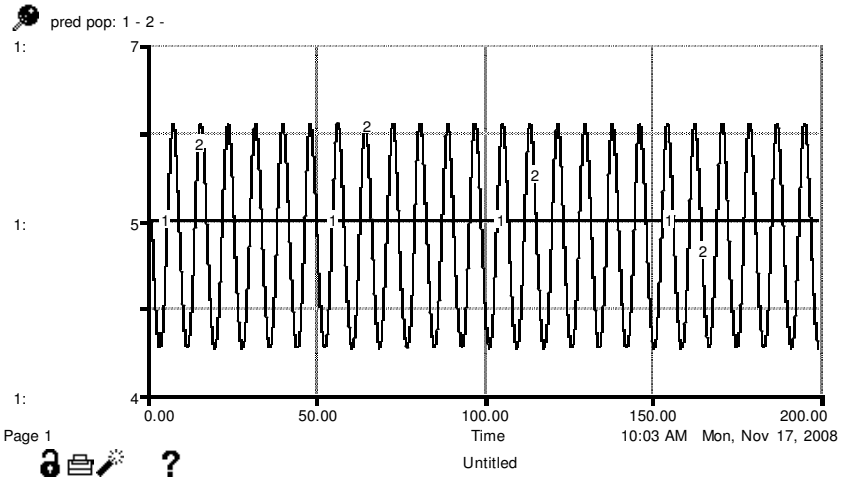

Figure 4.33. Behaviors with 1: f_j^{min*} = (1, 0.2, 0.5) and 2: f_j^{max*} = (0.92111, 0.19206, 0.63994) parameter sets among all runs

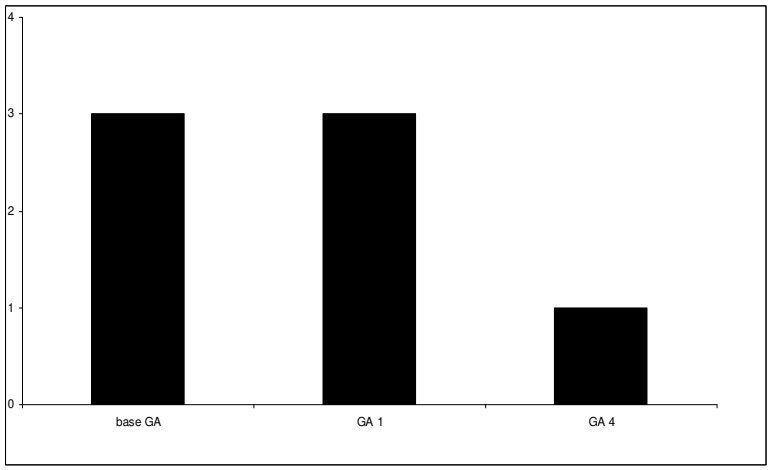

Figure 4.34. Number of times the solution converges to the optimum in 20 replications

4.5. Stock Management Model with Second Order Supply Line

4.5.1. Model

The model is a generic System Dynamics model. It is known that ignoring Supply Line may cause the Stock to produce undesired behaviors in Stock Management Models (Sterman 2000). When Weight of Stock (W_S) and Weight of Supply Line (W_{SL}) are both equal to one, the Stock cannot oscillate (Yaşarcan 2003). The general aim in the model is

for the Stock to reach the desired level fast and stably. The response time of the system decreases when the Stock Adjustment Time (T_{SA}) is decreased. However, this effect is not linear and after a point, decreasing T_{SA} does not cause the response time to decrease any further. Furthermore, very small T_{SA} may cause stability problems (Yaşarcan 2003). The general recommended policy approach in the literature for Stock Management is to decrease T_{SA} till the system responds fast enough or to the smallest possible value and to set W_S and W_{SL} to one (Yaşarcan 2003).

In the model used, Loss Flow is constant and is exactly known by the decision maker. The ordering decision includes the Supply Line Adjustment (SLA) and Stock Adjustment (SA) structures.

$$StockAdjustment = \frac{Desired\_Stock - Stock}{Stock\_Adjustment\_Time} \tag{4.40}$$

$$SupplyLineAdjustment = \frac{\text{Desired\_Supply\_Line} - Supply\_Line}{Stock\_Adjustment\_Time} \tag{4.41}$$

$$ControlFlow = LossFlow + \frac{W_S xSA + W_{SL} xSLA}{T_{SA}} \tag{4.42}$$

For our analysis, initial values of all stocks are four and the desired stock is set to five. Acquisition delay (T_{AD}) and T_{SA} are both equal to four. The system would be at equilibrium if the desired stock were four. We can think of perturbing the system at time zero with a sudden change at desired level. The equations of the model are given in Appendix B.

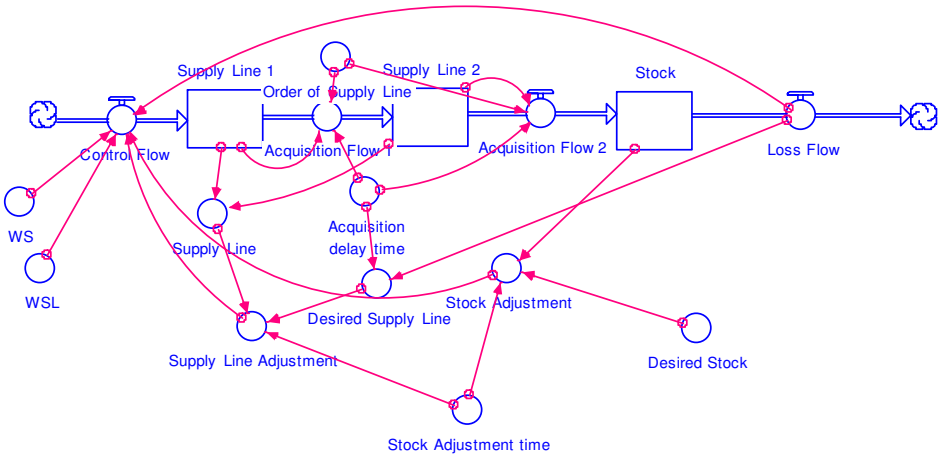

Figure 4.35. Stock-Flow diagram for Stock Management Model

4.5.2. Fitness Function for the Stock Management Model

For this model, we have chosen W_S and W_{SL} as the search parameters for the algorithm. The desired behavior in any stock management model is for the stock to reach desired level quickly, without oscillation. In the fitness function, these were defined as settling time (t_s), overshoot, and steady state error (e_{ss}). Settling time is defined as the time the stock level enters a band and never goes out. The band is an area defined between two points, which are obtained by adding, and subtracting a given percentage (p) of the settled level (S^s) to and from S^s. Hence, the middle point is S^s. t_s is divided by the time horizon to normalize its effect on fitness function to a number between zero and one. If the system fails to stabilize by the time horizon, the time horizon is taken as t_s. Overshoot is defined by subtracting S^s from the maximum level (S^{MAX}) the Stock reaches. For this model, overshoot is non-negative so we can use the below formulation.

$$Overshoot = S^{MAX} - S^S \tag{4.43}$$

Steady state error is simply the difference between the last value obtained at the end of time horizon and desired level (S^*).

$$e_{SS} = \left| S^S - S^* \right| \tag{4.44}$$

The fitness value returned by the function is defined as weighted sum of t_s, overshoot, and e_{ss}. These weights define the importance of each variable for the policy maker.

$$Fitness = Po \times Overshoot + Pess \times e_{SS} + Pts \times t_S \tag{4.45}$$

4.5.3. Genetic Algorithm Results

The time horizon for the search is set to 100. The parameters, W_S and W_{SL}, are set to have boundaries between zero and one since setting a limit that is higher than one for either one of them would mean decreasing T_{SA}. The percentage p of the band is set to 1%,

meaning that the band is between 1.01* S^s and 0.99* S^s. Initially, the weights of t_s (Pts), and e_{ss} (Pess) were set to one, and the weight of overshoot (Po) is set to 100. We have 20 replications of each GA. Best ($f_j^{\min}$), worst ($f_j^{\max}$) and mean ($\bar{f}_j$) fitness values obtained from these 20 replications by the different GA configurations are given in Table 4.7. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 2) vector composed of (W_S, W_{SL}).

Table 4.7. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 20 replications

	Mean fitness ($\bar{f}_j$)	Best fitness (f_j^{min}) (b_j^{min})	Worst fitness (f_j^{max}) (b_j^{max})
Base GA	0.322391	0.32228 (1, 0.90498)	0.32391 (1, 0.90113)
1. GA	0.32228	0.32228 (1, 0.90498)	0.32228 (1, 0.90498)
2. GA	0.324068	0.32228 (1, 0.90498)	0.3397 (0.88606, 0.77673)
3. GA	0.323831	0.32228 (1, 0.90498)	0.33704 (0.90722, 0.79878)
4. GA	0.32228	0.32228 (1, 0.90498)	0.32228 (1, 0.90498)
5. GA	0.337424	0.32355 (0.99719, 0.8997)	0.35952 (0.78336, 0.6528)
6. GA	0.322448	0.32228 (1, 0.90498)	0.3247 (1, 0.92213)
7. GA	0.322496	0.32228 (1, 0.90498)	0.32465 (1, 0.92293)

GA results always converge to same point. The fittest individual is $b_{ij}^{\min *}$= (1, 0.90498) with fitness $f_{ij}^{\min *}$=0.32228. The W_S and W_{SL} values that give worst fitness value, $f_{ij}^{\max *}$=0.35952 among all runs are $b_{ij}^{\max *}$= (0.7834, 0.6528). Figure 4.35 shows the behavior of these best and worst fit parameters.

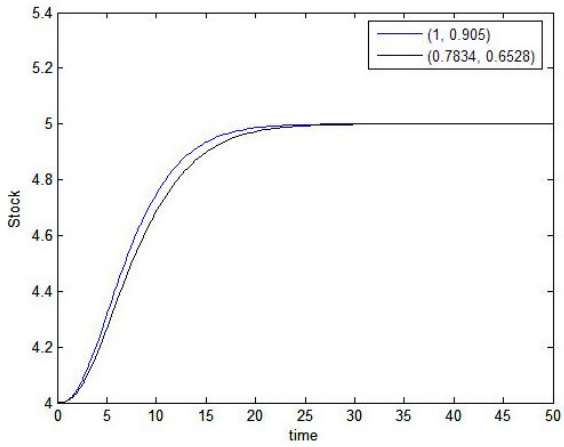

Figure 4.36. Behavior of best (b_{ij}^{min*}) and worst (b_{ij}^{max*}) parameter sets among all runs

The behavior shows that both of the worst and best parameter sets obtain good results. The best parameter set has a faster reaction, in other words, it has a smaller settling time (see Table 4.8 for other specifics of the behavior).

Table 4.8. Settling time, overshoot, and steady state error values of best (b_{ij}^{min*}) and worst (b_{ij}^{max*}) parameter sets

	b_{ij}^{min*}			b_{ij}^{max*}		
Fitness Value	0.32228			0.35952		
(W_S, W_{SLA})	1 0.905			0.7834 0.6528		
	t_s = 15.93	O= 3.74x10^{-5}	e_{ss} = 1.67x10^{-6}	t_s = 17.84	O = 2.76x10^{-5}	e_{ss} = 8.5x10^{-6}

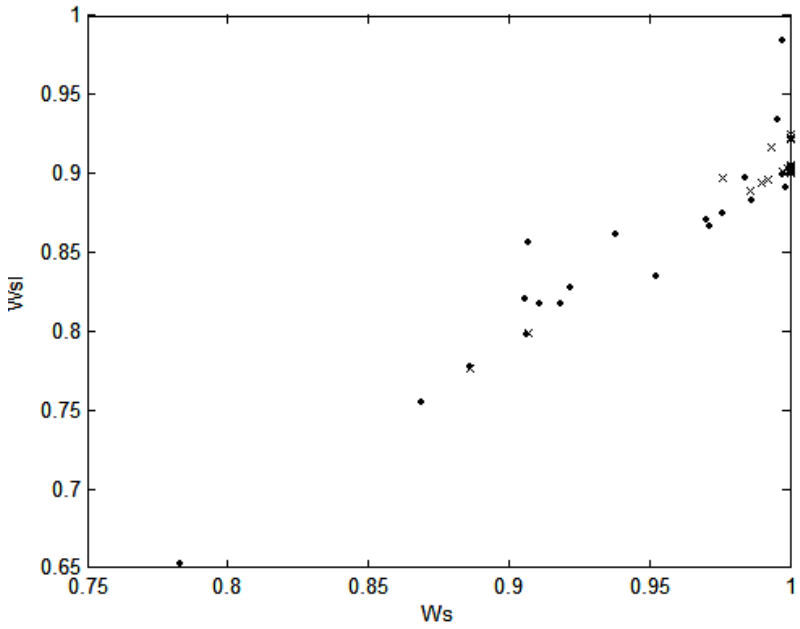

Figure 4.37. Plot of solutions provided by 160 GA runs with eight different configurations. (The dots (●) are the solutions provided by runs with 5th GA configuration ($b_{ij=5}$). Other solutions are marked with '×').

The solutions provided by GA show that generally different configurations work well and converge to a solution around (1, 0.9). Most of the results with higher fitness are from runs with GA5 (see Figure 4.37). Since all of the other configurations use heuristic crossover except 5th one, we conclude that heuristic crossover proves to be a more reliable operator than single point crossover for this search space.

It is also interesting that the weights of stock adjustment (W_S) and supply line adjustment W_{SL} do not converge to (1, 1) as recommended in other studies. The reason for that is when W_S=1 and W_{SL}<1, settling time decreases at the expense of allowing overshoot. The ratio W_S/ W_{SL} determines the overshoot and settling time values. Even when the penalty for overshoot is 100 times the penalty for the settling time, the GA solutions converge to the ratio W_S/ W_{SL}=1/0.9. W_S/ W_{SL} ratio increases to 1/0.6 when overshoot is given the same penalty as settling time (see Table 4.9). Notice that the settling times at Table 4.9 are all smaller than the settling time of best parameter set at Table 4.8, whereas overshoot is larger.

Table 4.9. Three runs with all the penalties equal to one with GA1

(Pts, Po, Pess)= (1,1,1)	Fitness Value	0.2705			0.2706			0.2705			0.3503		
	(W_S, W_{SLA})	1 0.6018			1 0.58			1 0.6018			1 1		
		t_s = 12.165	O= 0.0272	e_{ss} = 0	t_s = 11.965	O= 0.0312	e_{ss} = 0	t_s = 12.165	O= 0.0272	e_{ss} = 0	t_s = 17.515	O= 0	e_{ss} = 0

These results indicate that at the expense allowing small overshoot, we may obtain faster settling times. Appendix B further analyzes this trade-off. Next section explores the GA performance and policy changes when the starting conditions of the system are different.

4.5.4. Analysis of the Performance of GA with Different Initial Conditions and Different Non-policy Parameter Values

Previous analysis on the model is all done without altering the non-policy parameters and initial conditions of the stocks. In this section, we change some of the initial conditions and non-policy parameters and test their effects on search results. We change the initial conditions and non-policy parameters one at a time.

First initial condition that is altered is initial Stock value. The stock value is changed to one. The desired stock value is kept as five. We do five replications of GA. All of the search results converge to $(W_S, W_{SL}) = (1, 0.91724)$.

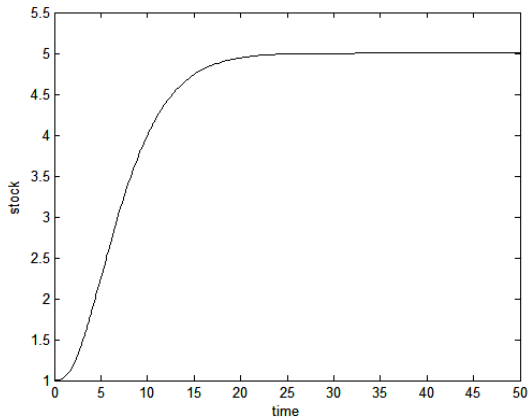

Figure 4.38. Behavior of stock with initial stock value 1 and $(W_S, W_{SL}) = (1, 0.91724)$

Secondly, initial conditions of Supply Line are increased from four to 12. Initial stock value is kept equal to four. In this case, the search results give (W_S, W_{SL}) = (1, 1). The behavior is as given on Figure 4.38.

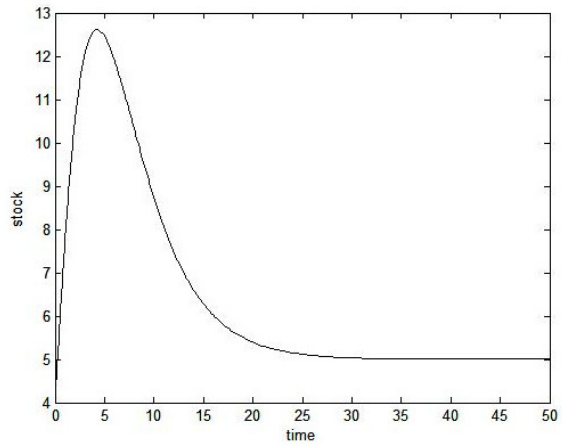

Figure 4.39. Behavior of stock with initial supply line values 12 and (W_S, W_{SL}) = (1, 1)

Next, we change, a non-policy parameter, T_{AD} from four to two. All of the search results converge to point (W_S, W_{SL}) = (0, 0.98953).

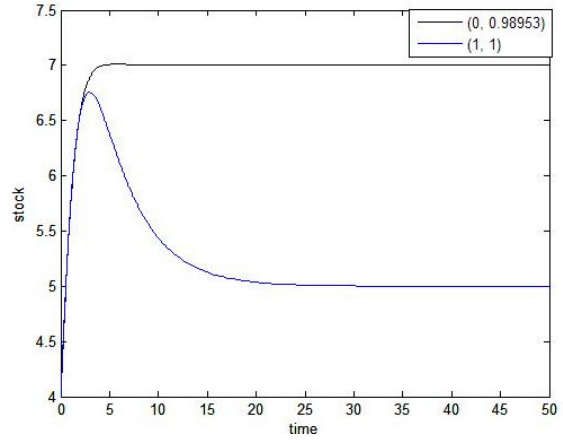

Figure 4.40. Behavior of stock with T_{AD} =2 and two different weight sets

The result obtained by the point (0, 0.98953) is more desired under circumstances of high penalty for overshoot. Unlike the behavior of point (1, 1), Figure 4.40 shows that

behavior of the point (0, 0.98953) has a very small overshoot but large steady state error. The choice of which values to use is up to the policy-designer.

Next, we examine when initial stock value (S_0) is one and T_{AD}=2. In the previous case, when T_{AD}=2 and initial stock was four, the possible policies resulted in either overshoot or steady state error. Now, in the case of T_{AD}=2 and S_0=1, as expected the GA results obtain a fast convergence to the desired stock value. The GA search results in W_S and W_{SL} values where $W_S<W_{SL}$ (and $W_S \approx W_{SL}$), unless $W_S=W_{SL}=0$. All of these weights result in the same behavior as given in Figure 4.41.

Table 4.10. W_S and W_{SL} values when T_{AD}=2 and S_0=1

Fitness	W_S	W_{SL}
0.11528	0.89876	0.90049
0.11529	0.91875	0.92036
0.11528	0.93452	0.93628
0.11536	0	0
0.11528	0.82253	0.82419
0.1153	0.50345	0.50482
0.11529	0.6258	0.62728
0.11536	0	0
0.11536	0	0
0.11528	0.99817	1

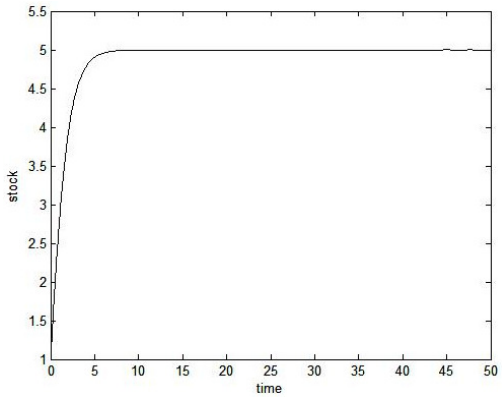

Figure 4.41. Behavior obtained from policies given in Table 4.10

Finally, we examine the possible strategies when T_{AD} = 8. All of the GA runs give the point (W_S, W_{SL}) = (1, 0.9549) and the behavior is as given in Figure 4.42.

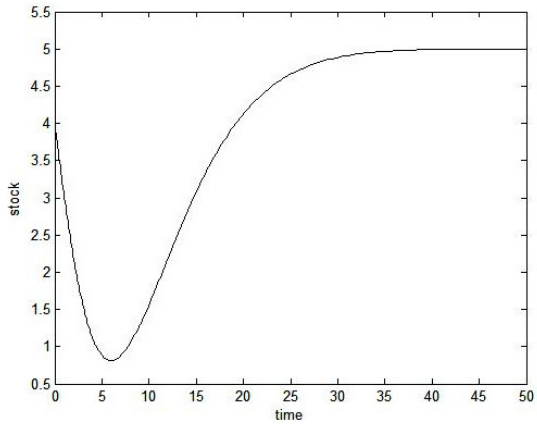

Figure 4.42. Behavior of stock when $(W_S, W_{SL}) = (1, 0.9549)$ and $T_{AD} = 8$

When $T_{AD} = 4$ and all of the initial stock and desired stock values equal to four, the system is at equilibrium. The rate that the goods flow from the supply line to the stock decreases when acquisition delay increases. We observe a decrease of the stock value in the dynamics at the start of the time horizon. Hence, the behavior of stock is very different from all of the previous analysis in this section. Although the solution provided by GA is reasonable, this kind of behavior makes the fitness function obsolete; this behavior also contains an under-shoot as well as a possible overshoot. A better fitness function would also include a penalty for the under-shoot. In cases where a different behavior is always observed, the objectives should be updated.

In this section, we used parameter search to test the reliability of the GA and our fitness function under different circumstances. Additionally, we used parameter search to explore alternative strategies based on various starting conditions.

4.6. Market Growth Model

4.6.1. Model

In this section, we consider a market growth model based on Forrester (1961). In this model, the number of salesmen changes by hiring/firing decision. *Hiring/firing* decision depends on the discrepancy between desired and actual s*alesmen*. Desired number of

salesmen is not constant. *Budget* for salaries divided by *salary* determines the desired salesmen. *Budget* is simply the multiplication of *Orders Booked* times *Revenue* per order which is constant ($10). *Salesmen* book orders so *Orders Booked* is *Salesmen* times *Sales Effectiveness*. *Orders Booked* causes the *Backlog* of orders to increase. *Backlog* decreases as goods are delivered. *Delivery Rate* is not constant; it increases with *Backlog Ratio* as given in Figure 4.43. *Backlog Ratio* is simply *Backlog* divided by the *Norm Backlog* (constant).

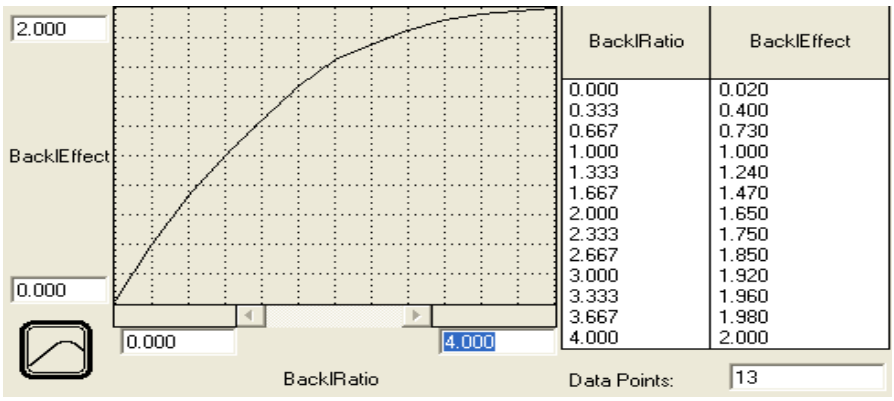

Figure 4.43. Backlog Effect vs. Backlog Ratio graph

The company is assumed to have a fixed capacity; its delivery rate deteriorates as the backlog approaches the company's production capacity (see Figure 4.44).

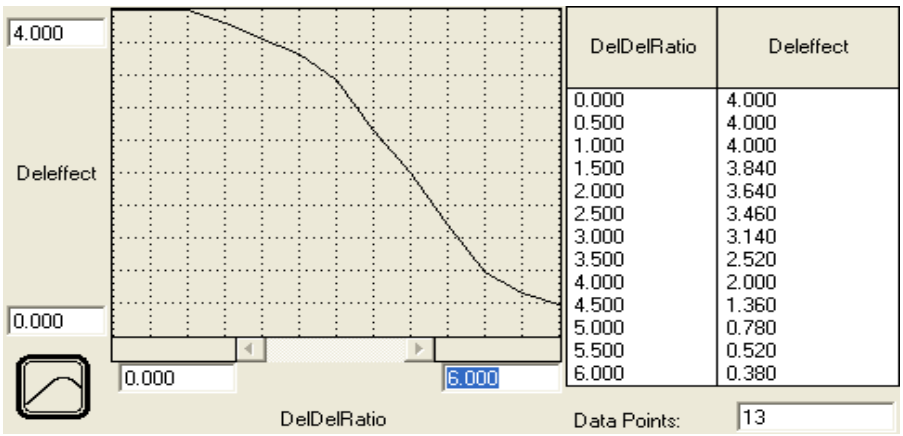

Figure 4.44. Delivery Delay Ratio vs. Delivery Effect graph

Two major loops in the model affect market growth. One of them is the reinforcing loop of "*salesmen* increases with sales and sales increase with increasing salesmen." Second is a balancing loop: "Increase in sales causes *backlog* to increase, which decreases *sales effectiveness* in time, then sales decrease due to decrease in *sales effectiveness*." See Appendix C for the list of model equations.

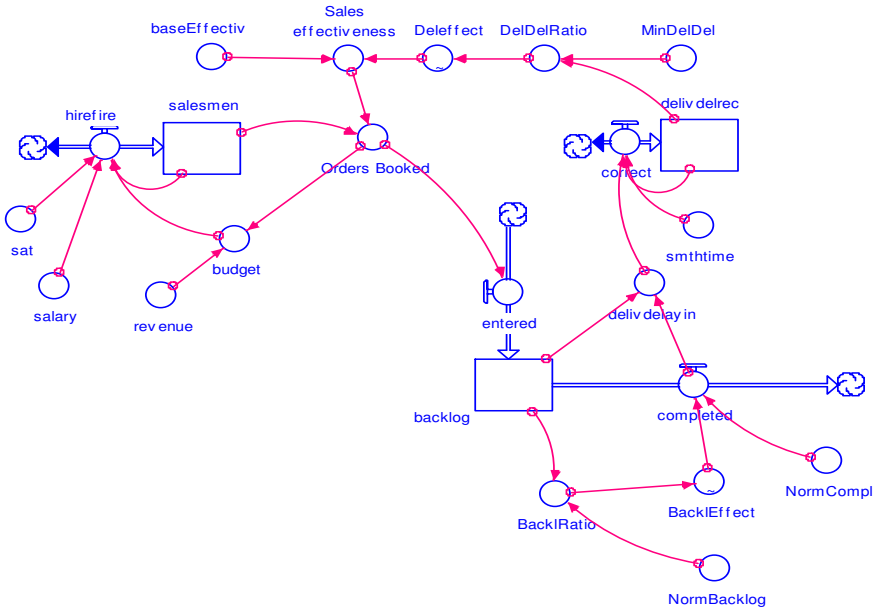

Figure 4.45. Stock-flow model of "Market Growth" model

The parameter values and the behavior of backlog and salesmen are as given below:

INIT backlog = 50000

INIT salesmen = 50

baseEffectiv = 100

MinDelDel = 1

NormBacklog = 25000

NormCompl = 10000

revenue = 10

salary = 1000

sat = 20

smthtime = 8

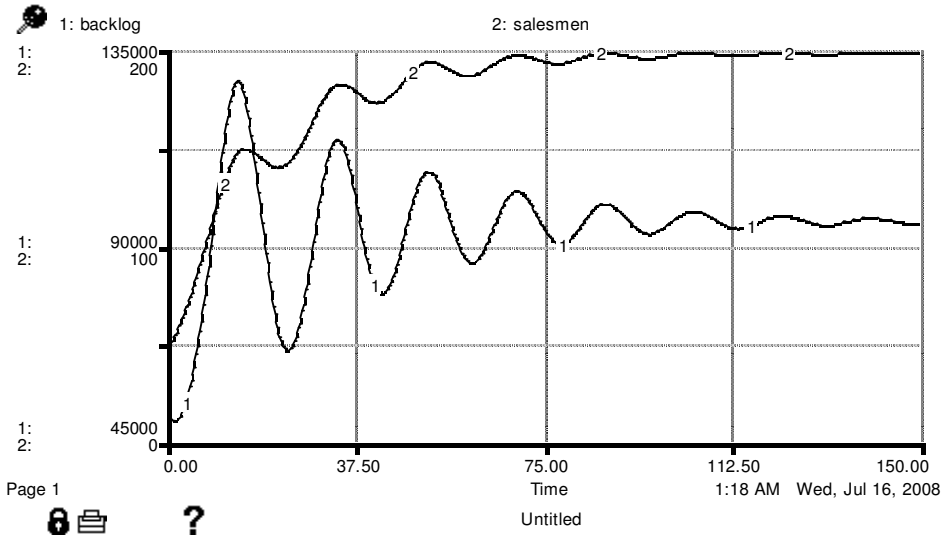

Figure 4.46. Behavior of backlog and salesmen with given parameter values

4.6.2. Fitness Function for the Market Growth Model

The parameter search in this problem has the objectives to stabilize the salesmen (Sal) and backlog (Bac) at their desired levels. Similar to previous problems, the aim is to stabilize quickly without oscillation. In the fitness function, these objectives are tried to be reached by penalizing settling time (t_s), undershoot, and steady state error (e_{ss}). Settling time is defined as the time the system variable goes into ±5% band of the settled value (Sal^S or Bac^S) and never goes out. Time horizon is the settling time if the system fails to settle. We define steady state error (e_{ss}) as the difference between settled level and desired level (Sal^* or Bac^*).

$$e_{SSS} = \frac{\left| Sal^* - Sal^S \right|}{Sal^S} \tag{4.46}$$

For the behavior of backlog and salesmen, similar to previous fitness functions, we must define overshoot. Overshoot is the difference between maximum of the value (Sal^{MAX} or Bac^{MAX}) and Sal^S or Bac^S normalized by the settled values. For the fitness function, Sal^{MAX} or Bac^{MAX} is always greater than or equal to their settled values.

$$Overshoot_S = \frac{\left| Sal^{MAX} - Sal^S \right|}{Sal^S} \tag{4.47}$$

The fitness function is the weighted sum of t_s, undershoot, and e_{ss} for salesmen and backlog and their associated weights (Po, Pess, and Pts) are 2, 5, and 2 respectively.

$$Fitness = Po \times (Overshoot_S + Overshoot_B) + Pess \times (e_{SSS} + e_{SSB}) + Pts \times (t_{SS} + t_{SB}) \tag{4.48}$$

We determine three parameters as policy parameters: stock adjustment time (*sat*), *salary*, and smoothing time (*smthtime*).

4.6.3. Genetic Algorithm Results

The time horizon for the search is set to 100 time units. The corresponding ranges are 5-30 for *sat*, 1000-3000 for *salary*, 1-15 for *smthtime*. Desired salesmen (Sal^*) and Desired backlog (Bac^*) values are 150 and 80000, respectively. We have 20 replications of each GA. Best ($f_j^{\min}$), worst ($f_j^{\max}$) and mean ($\bar{f}_j$) fitness values obtained from these 20 replications by the different GA configurations are given in Table 4.11. Each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 3) vector composed of (*sat*, *salary*, *smthtime*).

Table 4.11. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 20 replications

	Mean fitness ($\bar{f}_j$)	Best fitness ($f_j^{\min}$) ($b_j^{\min}$)	Worst fitness ($f_j^{\max}$) ($b_j^{\max}$)
Base GA	0.8419	0.82587 (5.6399, 1315.5, 1)	1.114 (6.0209, 1409.9, 1)
1	0.8425	0.82587 (5.6399, 1315.5, 1)	0.9927 (6.0892, 1370.7, 1)
2	1.1839	0.83737 (6.0181, 1316.7, 1)	2.0378 (13.118, 1461.1, 3.2248)
3	1.2045	0.82889 (5.5874, 1317, 1)	1.829 (13.165, 1315.1, 4.385)
4	0.8659	0.82587 (5.6399, 1315.5, 1)	1.0853 (8.7932, 1310.5, 1.5296)
5	1.2444	0.98315 (5.1403, 1326.5, 1.0247)	1.7676 (11.608, 1251.5, 2.2543)
6	0.8266	0.82587 (5.6399, 1315.5, 1)	0.83757 (6.0045, 1317, 1)
7	0.8455	0.82587 (5.6399, 1315.5, 1)	1.0083 (5, 1326, 1)

The fittest individual among all runs is $b_{ij}^{\min *}$= (5.6399, 1315.5, 1) with fitness $f_{ij}^{\min *}$=0.82587. The parameter values that give worst fitness value, $f_{ij}^{\max *}$=2.0378, among all runs are $b_{ij}^{\max *}$= (13.118, 1461.1, 3.2248). Figure 4.47 and 4.48 display the behavior of these best and worst fit parameters for backlog and salesmen, respectively. The worst solution has a higher settling time and steady state error for backlog and salesmen than the best solution. For salesmen, the best solution has very small overshoot penalty and steady state error. Despite the penalties for the objectives, Figure 4.47 and 4.48 demonstrate that both worst and best solutions obtain the same dynamic pattern.

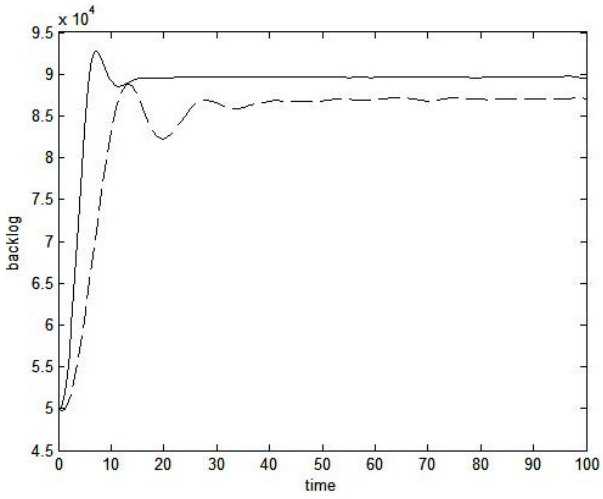

Figure 4.47. Behavior of backlog with best (b_{ij}^{min*}) (solid line) and worst (b_{ij}^{max*}) (dashed line) parameter sets among all runs

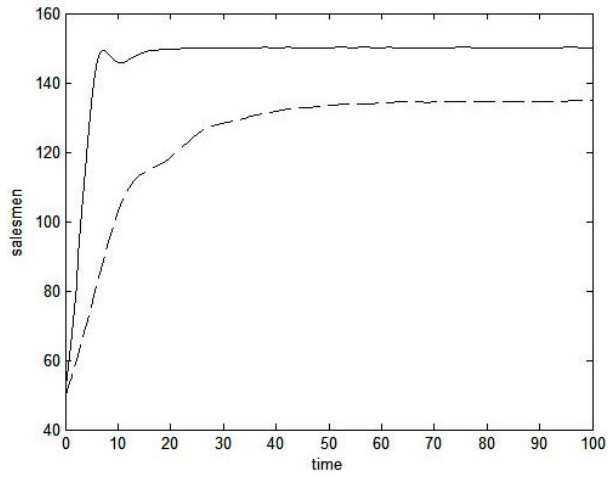

Figure 4.48. Behavior of salesmen with best (b_{ij}^{min*}) (solid line) and worst (b_{ij}^{max*}) (dashed line) parameter sets among all runs

Table 4.12 shows that GA2, GA3, and GA5 are among the configurations that obtain bad results. In this search space, heuristic crossover is again a better reproduction operator than single point crossover. According to mean of the best fitness values obtained, GA6 is the best configurations. This configuration scans the search space better than other configurations. However, Figure 4.49 indicates that they need larger number of generations in terms fine-tuning (i.e. for converging to the best solution, f_{ij}^{min*}= (0.82587), found). In terms of obtaining the best solution, Base GA has the best record. We conclude that Base

GA and GA4 exploit the search space better than GA6 and GA7. However, GA6 is a more reliable configuration to converge to the best solution. In this search space and using this fitness function, considering both the mean fitness and number of times best solution obtained, among these eight configurations, Base GA is the best configuration.

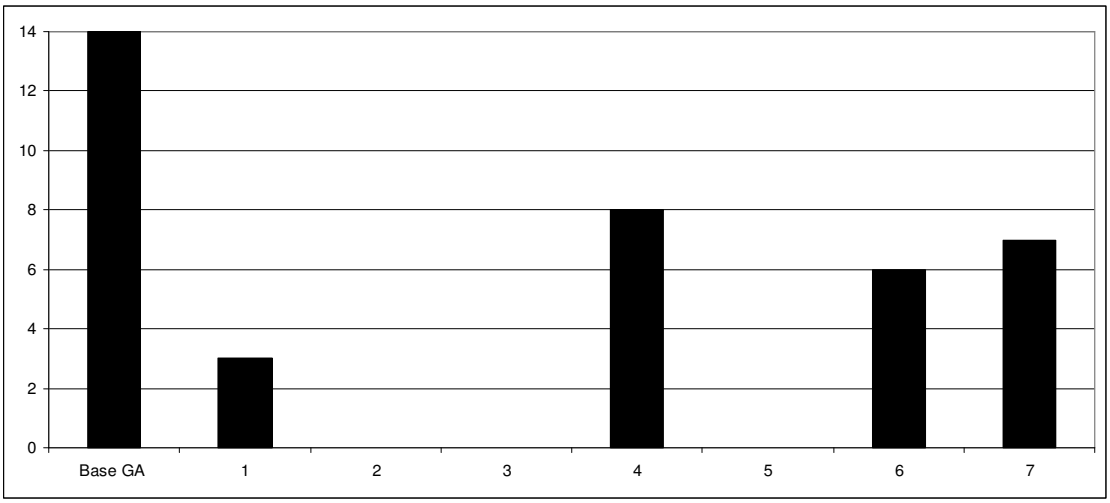

Figure 4.49. Number of times best solution $f_{ij}^{\min *}$ = (0.82587) is obtained in 20 replications by different configurations

4.6.4. GA Performance with Higher Number of Policy Parameters

In this section, we analyze the performance of the GA configurations, when the search dimensions are increased. We increase the number of policy parameters from three to five. We include normal backlog (*NormBacklog*) and normal completion (*NormCompl*) into the policy parameter set. The range for *NormBacklog* is between 10000 and 40000 and the range for *NormCompl* is between 1000 and 20000. We replicate each of the eight configurations 10 times. The results are summarized in Table 4.12 Notice, in this case, each element (b_{ij}) (i.e. best individual) of the **B** matrix is a (1 x 5) vector composed of (*sat*, *salary*, *smthtime*, *NormBacklog*, and *NormCompl*).

Table 4.12. Mean fitness value ($\bar{f}_j$), best fitness value ($f_j^{\min}$) and associated best individual ($b_j^{\min}$), and worst fitness value ($f_j^{\max}$) and associated worst individual ($b_j^{\max}$) of eight different GA configurations in 10 replications

	Mean fitness ($\bar{f}_j$)	Best fitness ($f_j^{\min}$) ($b_j^{\min}$)	Worst fitness ($f_j^{\max}$) ($b_j^{\max}$)
Base GA	0.6715	0.26644 (5, 1136.6, 1, 26932, 8907)	2.0313 (23.359, 1000, 1, 10000, 7853.3)
1	0.6971	0.2974 (5, 1128.1, 1.0281, 26620, 8824.7)	1.2729 (12.583, 1135.8, 4.6867, 24691, 8745.5)
2	1.3529	0.58378 (8.719, 1145.7, 1.4836, 26428, 8908.5)	2.2449 (16.527, 1345, 3.0792, 39012, 11595)
3	1.2187	0.54083 (8.7666, 1127.4, 1, 22267, 8604.5)	1.9158 (15.64, 1259, 5.3863, 24894, 8989.2)
4	0.6604	0.36685 (6.6155, 1135.9, 1, 10000, 8522.6)	1.1912 (7.9016, 1000, 3.8061, 36418, 9164.1)
5	1.4406	0.91093 (11.196, 1069.3, 1.5555, 20919, 8145.4)	2.1102 (11.12, 1720.5, 1.6879, 21168, 11759)
6	0.3734	0.2704 (5, 1136.2, 1, 26585, 8871.5)	0.56273 (6.1731, 1083.4, 1, 10000, 8351.9)
7	0.6122	0.27203 (5, 1136.5, 1, 25811, 8823.9)	0.96155 (5, 1000, 1, 10000, 7634.7)

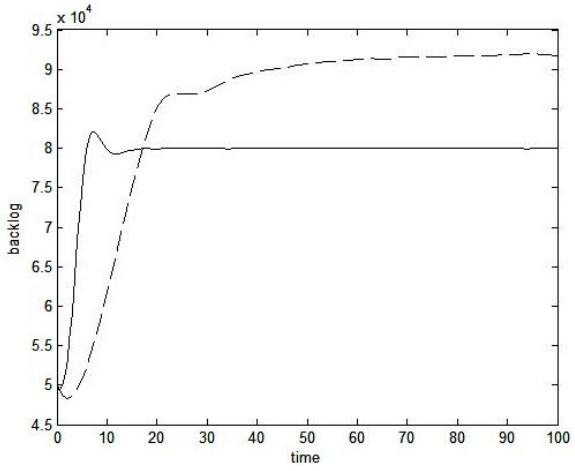

Figure 4.50. Behavior of backlog with best (solid line) and worst (dashed line) parameter sets among all runs

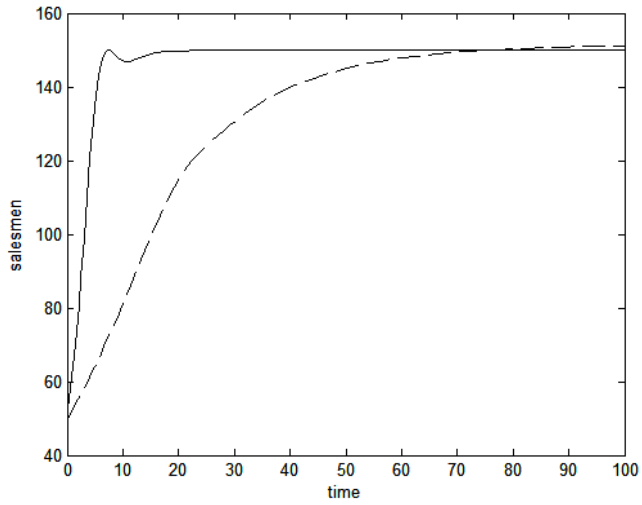

Figure 4.51. Behavior of salesmen with best (solid line) and worst (dashed line) parameter sets among all runs

Adding new parameters to policy parameter set enables the search to obtain better fitness values. Even though all of the GA configurations obtain better best fitness values, the mean fitness values compared to three-policy parameters design are higher for GA2, GA3, and GA5. Base GA and GA1 through GA5 obtain higher fitness values of worst solutions in 5-parameter search compared to three-parameter search (see Tables 4.11 and 4.12). Hence, we conclude that as the search space increases in dimension, algorithm needs more generations to exploit and explore. In search with five policy parameters, GA6 is the best configuration in terms of mean fitness ($\bar{f}_j$). GA6 is a more explorative configuration. Explorative configurations prove to work better as the search space grows.

4.6.5. Number of Policy Parameters versus Number of Generations

As the number of policy parameters increase, the search space grows. Hence, we need to increase the number of generations or come up with a different stopping criterion. To evaluate this, we have decided on stall generation limit as the stopping criterion. Stall generation limit is a limit on the number of generations without an improvement on the best individual. Stall generation limit is set to 50. This means the search stops after 50 iterations with no improvement in the best individual. We increase the number of policy parameters in the market growth model from two to six one by one. With each parameter set, we have 10 replications each with different random seed. The GA configuration

chosen for the runs is GA6 as it proved to be the best configuration in terms of mean fitness and obtaining the best individual in runs with five policy parameters.

Table 4.13. Fitness values and iteration number in GA configuration with stall time limit

# of parameters	2		3		4		5		6	
Replications	fitness value	iteration	fitness value	iteration	fitness value	iteration	fitness value	iteration	fitness value	iteration
1	2.5461	69	0.8259	88	0.35258	202	0.56273	124	0.32826	62
2	2.5472	67	0.82589	120	0.35657	182	0.26992	299	1.5873	88
3	2.5538	87	0.82591	136	0.3557	104	0.2704	120	0.25836	171
4	2.5703	70	0.82587	115	0.35122	147	0.43935	83	0.24793	162
5	2.5462	201	0.82591	131	0.54018	68	0.32377	140	0.2849	82
6	2.546	124	0.82587	123	0.35498	118	0.41684	157	0.24832	458
7	2.547	91	0.82587	130	0.7222	74	0.34736	87	0.23288	155
8	2.5468	195	0.82588	161	0.41687	79	0.39974	99	0.25197	192
9	2.5477	80	0.8261	95	0.71971	103	0.27143	148	0.35995	183
10	2.5461	119	0.82589	93	0.3603	114	0.27382	154	0.27261	174
average	2.54972	110.3	0.825909	119.2	0.453031	119.1	0.357536	141.1	0.407248	172.7

Table 4.13 gives us the fitness value obtained from each replication with different policy parameter set sizes and their associated last generation number. When we look at average fitness values, we see that as the number of parameters increase from two to five, the mean fitness value declines. This decline is expected since additional dimension to the search space enables us with higher control over the system, and hence it is easier to obtain desired objectives. Notice that the best fitness value among all runs is obtained with six policy parameters (replication seven with fitness 0.23288). From five policy parameters set to six policy parameters set, the average fitness value increase which shows that GA needs more iterations to explore and converge. If the GA had enough iterations to explore and converge then the average fitness value would at least be the same or smaller than the runs with one less parameter in the policy set.

Secondly, Table 4.13 gives the final iteration numbers of the search. With increasing number of policy parameters, the average of iteration numbers increases. The increase is expected since as the search dimension grows, higher number of iterations is necessary for convergence. In some cases, we can observe a decrease in the average number of iterations as the search space further grows. This decrease is because as the dimension further grows,

50 stall generations is not enough to explore the search space. In other words, as the dimension further grows, the algorithm can get stuck at a local optimum and hence, it will need more than the given number of stall generations for convergence to the global optimum.

Finally, the stall generation stopping criterion shows that for the evaluation of two, three, and four policy parameters, 100 iterations is enough, referring to our analyses in Section 4. The average number of generations is approximately 120 for three-policy parameter search, which means that the best individual is first obtained around the 70^{th} generation since the algorithm terminates after 50 iterations without any improvement in the fitness value. When the policy parameter number is five or more, 100 iterations are clearly not adequate. Note that, this analysis is done for one of the configurations, GA6, and will be different for other configurations. In short, we can decide whether the number of function evaluations is adequate by looking at the decrease in the average fitness values since search should at least obtain the same fitness value, preferably smaller fitness value, as the search with one less parameter.

From SD point of view, we prefer reaching the objectives by manipulating smaller number of parameters. In terms of policy design, change in smaller number of parameters is easier to grasp and implement. In terms of sensitivity and validation, it might be necessary to deal with large number of parameters to grasp the total effect of parameters such as in the study by Miller (1998).

5. INTERPRETATION OF THE RESULTS OF GENETIC ALGORITHMS

In the previous chapter, we have given the results, showing the performance of different GA configurations on different models. This chapter interprets and summarizes the results of the previous chapter.

In all of the models, parameter search with GA generally provides the desired behavior and meets the desired goals. In FM, MBS, Stock Management, and Market Growth models, we use variations of the same fitness function where overshoot, steady state error, and settling time are penalized. In these fitness functions, we pay attention to normalize the penalties or try to keep the penalties in between zero and one. Hence, the fitness raw scores are in "good" range. As a result, the proportional scaling and roulette wheel selection works well in all of these models. However, in predator prey model, the fitness function value can vary in range since there is no normalization to the error in the means. In predator prey model, GA1 with rank scaling is the best configuration, which eliminates the problem of varying raw scores. In all of the models, GA5 with single point crossover performs worse than the rest of the configurations with heuristic crossover.

In general, although we have not explicitly stated, we have discussed the differences in the performance of GA configurations in terms of convergence velocity and convergence reliability (i.e. in terms of exploitation versus exploration trade off). Mean fitness is an indicator of convergence reliability and number of times the best solution found indicates the convergence velocity. Some elements such as ranking, top scaling have a greater selection pressure than proportional scaling, which means that without any genetic operator (such as crossover or mutation) the time that the population consists of copies of the best individual is shorter (also known as takeover time). Similarly, tournament selection has a greater selection pressure than roulette wheel selection. These elements are more exploitative rather than explorative (see Chapter 5 of Bäck, 1996 for more formal discussion on selection pressure and takeover time). GA4 with five elites proves to have high convergence velocity by converging to the best solution highest number of times compared to the other configurations in fan-motor, MBS, and market

growth models. GA6 and GA7 prove to be more explorative by obtaining small mean fitness values while often failing to converge to the best solution. Except the supply chain model, GA4 has a higher mean fitness than GA6 and GA7. Base GA generally lies in between GA4 and GA6 in terms of exploration and exploitation adjustment. Hence, in most of the cases, Base GA may be preferred as it seems to balance exploration versus exploitation well.

Although System Dynamics shares the aim to optimize parameters according to a fitness function, it has the primary aim of obtaining the desired behavior. Hence, we have to decrease the possibility of obtaining undesired behavior by extensively exploring the search space. We suggest using more explorative configurations such as GA6 and GA7. Another way to enhance exploration or convergence reliability is to increase mutation probability of a gene in uniform mutation for the base GA.

One aim of this thesis is to suggest guidelines for application of GA to SD. Following that goal, we analyze the number of random replications that are needed to guarantee a good solution. Here, we define good solution as the solution that has the same dynamic behavior as the best solution in 20 replications and the good individual is close to the best individual. Since there is no straightforward answer to determine the exact number of replications needed, we look at whether 20 replications would be needed by comparing it to the results of 10 replications. For this analysis, we choose Base GA as our configuration since it is the configuration that we suggest for these models. Table 5.1 gives the summary of Base GA performance in nonlinear electric circuit (RLC), FM, predator-prey (PP), stock management (SM), and market growth (MG) models for 20 replications. We exclude MBS model from this list since we originally had a total of 10 replications with this model.

Table 5.1. Mean fitness value ($\bar{f}_{j=0}$), best fitness value ($f_{j=0}^{min}$) and associated best individual ($b_{j=0}^{min}$), and coefficient of variation (CV) of Base GA results in 20 replications

	Mean ($\bar{f}_{j=0}$)	$f_{j=0}^{\min}$	$b_{j=0}^{\min}$	CV ($\frac{\sigma}{\mu}$)
RLC	0.0031	0.0011607	(2.1833, 3.8522)	0.9625
FM	0.7180	0.71618	(0.54372, 1.5071, 0.01)	0.0073
PP	0.2593	0	(1, 0.2, 0.5)	0.8552
SM	0.3224	0.32228	(1, 0.90498)	0.0011
MG	0.8419	0.82587	(5.6399, 1315.5, 1)	0.0762

The Table 5.2 gives the summary of Base GA performance in 10 replications. In this table, we include how the best individual in 10 replications was ranked in 20 replications.

Table 5.2. Mean fitness value ($\bar{f}_{j=0}$), best fitness value ($f_{j=0}^{min}$) and associated best individual ($b_{j=0}^{min}$), and coefficient of variation (CV) of Base GA results in 10 replications

	Mean ($\bar{f}_{j=0}$)	$f_{j=0}^{\min}$	$b_{j=0}^{\min}$	CV ($\frac{\sigma}{\mu}$)	Rank of the best among 20 replications
RLC	0.0035	0.0011657	(2.1833, 3.8522)	0.8197	3
FM	0.7178	0.71618	(0.54372, 1.5071, 0.01)	0.0067	1
PP	0.3784	0	(1, 0.2, 0.5)	0.6438	1
SM	0.3224	0.32228	(1, 0.90498)	0.001	1
MG	0.8560	0.82587	(5.6399, 1315.5, 1)	0.106	1

In terms of mean fitness, the performance of the Base GA with 10 replications is close to performance with 20 replications except PP model. PP model has a fitness function that is not normalized (i.e. raw fitness values), so a bad solution may have very high fitness value which would affect the mean more with smaller number of replications. In all of the models except RLC model, Base GA finds the best solution of 20 replications at least once in 10 replications. In other words, the best solution found does not improve with further replications beyond 10. The best solution found in 10 replications for RLC model is ranked third 20 replications. Still the best individual in 10 replications is very close to the best individual in 20 replications. The two points differ after the fourth digit, which has negligible effect on system behavior. CV is defined as standard deviation divided by the mean. In both tables, CV gives us information on how much the solutions deviate from the mean. This deviation is normalized by the mean. For RLC model, CV is

high due to mean fitness being a very small number. Although the deviation looks high compared to the mean, the results are satisfactory in terms of behavior. PP model also has a high CV but this is again due to its fitness function being not normalized. In general, these results indicate that 10 replications are enough to obtain good solutions. For MBS, we followed similar analysis and compared 10 replications versus five replications. The results were more than satisfactory with means equal to 0.0031, and 0.0034, with CVs equal to 0.0003, and 0.0005, respectively for 10 and five replications. Furthermore, Base GA obtained the best individual in five replications. The results indicate five replications are enough to obtain good solutions for the MBS system. The same analysis shows that in FM, PP, SM, and MG models, five replications can also yield the best individual. RLC model again obtains a solution very close to the best individual. Thus, we conclude that 20 replications are not necessary to guarantee good solutions. In practice, five replications are enough for all of the models to obtain good solutions. When the parameter size grows, for instance MG model with five policy parameters, we suggest using a more explorative GA configuration like GA6, and five trials are again enough to obtain good solutions.

Next, we analyze run time of GA. The run time of the algorithm depends on the simulation time of the models. Each GA has 100 iterations with population size 50. There are 5000 function evaluations or simulations of the model. The run time of a single GA for each model is given on Table 5.1. The run time of the GA depends greatly on simulation time of the model. Single GA run in MBS model may take up to three hours since the solver has to work with very small DT values for that model.

Table 5.3. Average run time of a single GA run in seconds for each model

Model	Average run time
RLC	400 s
FM	350 s
MBS	10000 s
PP	35 s
SM	110 s
MG	100 s

6. CONCLUSION

SD aims to model complex dynamic systems of interesting or problematic behavior and come up with improvement policies. Given the complexity caused by feedbacks, delays and nonlinearities, it is hard and time-consuming to determine policy parameter values by human decision makers. Hence, it is convenient to make use of search algorithms to obtain good parameter values. Search algorithms have been used for parameter optimization in SD in sensitivity analysis, validation, and policy design. The adequacy of these search algorithms has not been systematically tested. In this research, we evaluate the adequacy of GA in parameter optimization of dynamic systems. We analyze the performance of GAs in different models with different objective functions. We suggest variations in GA to increase the efficiency of the search when it is possible.

In Chapter 3, after we explain how a GA works, we give eight different configurations of genetic algorithm. Our research methodology is to compare these different configurations in dynamic models.

Chapter 4 is the section where we analyze the adequacy of GA in six dynamic models. The models are selected from control theory and generic system dynamics models. The first model is a simple nonlinear electric circuit composed of a resistance, inductance, and capacity. The stated goal is to adjust the period and amplitude of the oscillation by choosing the inductance and capacity values. The second model is a linear fan motor model where the fan force tries to stabilize a plate hanged on a hinge at a desired angle. Oscillations and steady state error are penalized in the fitness function. The third model is a magnetic ball suspension system where the goal is to keep the ball on air at a desired distance from the magnet. The magnet pulls the ball and tries to counteract the gravitational force. The other three models that we discuss are generic system dynamics models. The first one of these is the predator prey model. In this model, we search for the parameter sets that keep the system at equilibrium. The next model is a stock management model with a second order supply line delay. We aim to stabilize the stock at a desired level by determining the weights of supply line and stock adjustment terms. We further test the performance of the genetic algorithm and our objective function by changing initial

stock values and non-policy parameter values. The results indicate that some initial conditions can necessitate the updating of the fitness function accordingly. The final model is the well-known market growth model. Our objective is to stabilize salesmen and backlog stocks at their desired values.

Chapter 5 summarizes the performance outcomes of the genetic algorithm. The results of Chapter 4 indicate that more explorative configurations such as GA7 with Gaussian mutation and GA6 with smaller crossover fraction generally have lower objective values and are better at finding the desired behavior. The downside of more explorative search is that it takes longer to converge. Another result is that some of the configurations such as top scaling, tournament selection with higher selection pressure may get stuck in local optima and their performance depends on the distance of the initial individuals to optimal solutions. In all of the models, heuristic crossover is a better operator than single point crossover. Since system dynamics is not always concerned with obtaining *the* optimum solution, we can say that genetic algorithms generally work well. In the second part of Chapter 5, we evaluate the performance when we increase the number of policy parameters in the market growth model. As expected, better solutions are possible with additional parameters but the configurations have higher mean fitness. This means that we must increase the number of populations or generations. We must again prefer configurations that explore the search space well when the number of policy parameters increase.

A secondary aim of this thesis is to combine control theory models and techniques with SD tools. In three of the models, we have evaluated and modeled physical systems with SD tools. We proposed control techniques such as PID. We have adapted PID to SD modeling since using the derivative of a variable is not possible or realistic. Instead of derivatives, we have used trend function to estimate the change in the variable that we track. This PID structure can be used to control nonlinear models in SD literature. We hope that this study will be a step towards stronger communication between the fields of control theory and system dynamics.

In conclusion, the aim of testing the adequacy of GAs for SD is accomplished. GAs lead us to spot regions where good solutions lie which are generally satisfactory for SD models. We have also shown that parameter search gives some essential information on

dynamics of systems and can help us clarify our objectives. We believe that the use of GAs in SD literature will increase in real life applications. The study is also a step towards creating some guidelines for the use of GA in parameter search. In fact, for any large model, the modeler can do similar analysis to test the performance of the GA, such as exploring *number of parameters vs. number of generations* is needed. Although, we have extensively tested the GA performance in different models, further research is needed on the evaluation of solution quality, such as testing the performance of the algorithm by varying ranges of parameters. Future research may also focus on exploring performance of GA with other fitness functions. Furthermore, in this study, we have taken arbitrary values for the weights of errors or penalties in the fitness function. We have not focused on analyzing GA's performance as a function of different weights of penalties in the fitness function. It is likely that different GA configurations perform better with different set of penalty weights. Note that we have compared the performance of GA with different initial conditions only in the Stock Management Model. This comparison can be extended to other models. A modeler can make use of this type of comparison when doing parameter search on any model for policy design, or validity testing. Finally, the comparison of GA with other search algorithms is a research area that is not explored for SD models. It is a promising research area since SD modelers look for algorithms that yield "good enough" solutions for problems for which optimum solutions are typically intractable.

APPENDIX A: ANALYSIS OF FAN-MSOTOR MODEL WITH DISCRETE DELAY

This Section discusses the system behavior of the Fan-Motor Model when the actual force on the plate (F_d) is obtained via using a pure delay rather than using continuous goal-seeking formulation. The control-flow model is given below:

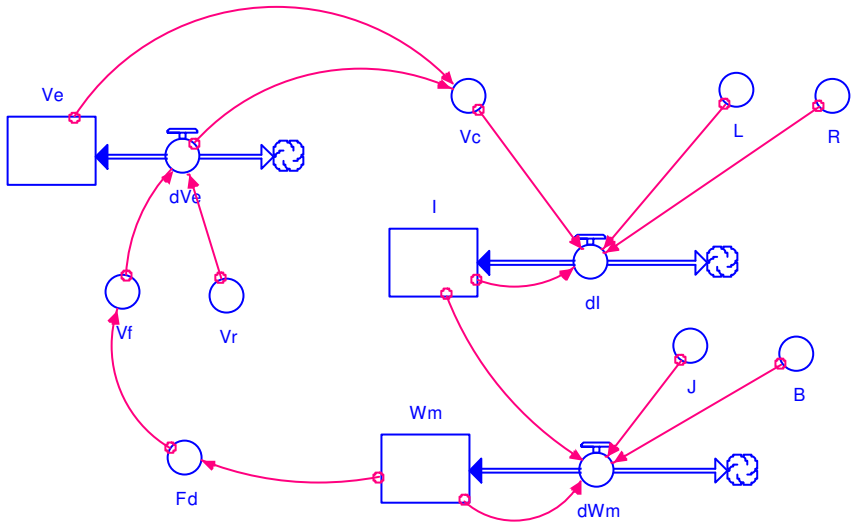

Figure A.1. Fan-Motor model with PI control structure and discrete delay

The only difference from the previous model with the continuous exponential delay is the equation of F_d. The altered equation compared to the model in section 4.2 is given below. The rest of the equations are the same as the Fan-Motor model with continuous exponential delay.

$$F_d = W_m[\text{t-delay}] \qquad \text{for t} \geq \text{delay}$$
$$F_d = 0 \qquad \text{for t} < \text{delay} \tag{A.1}$$

The value of the delay is 0.225 seconds, the same as the delay time in continuous exponential delay. The final voltage (V_F) behavior with K_P= 1, K_I=1, and L=0.03 is given in the next figure, comparatively for continuous and discrete delays.

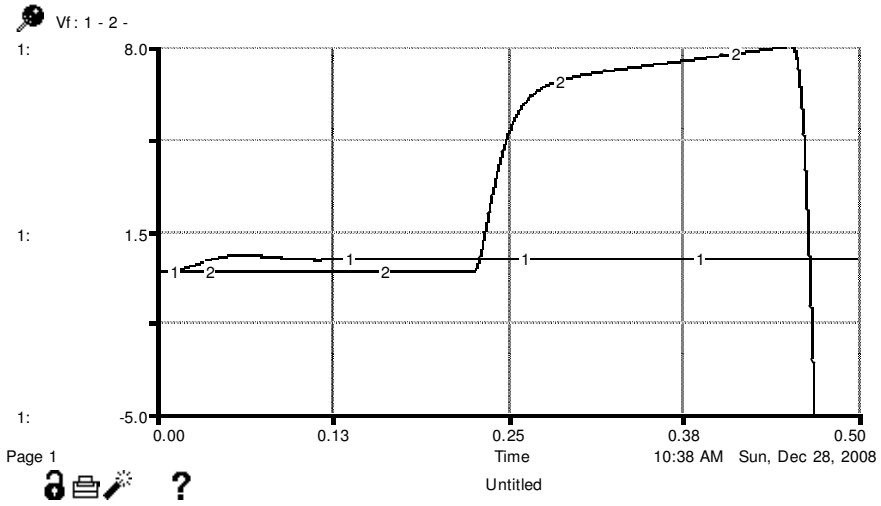

Figure A.2. Final voltage (V_F) behavior with 1: Continuous exponential delay, 2: Pure delay

Figure A.2 shows that parameter values that can stabilize V_F with continuous exponential delay can no longer achieve this objective when the system has pure delay. Next, we use GA to search for parameter values that can stabilize the final voltage. The search uses Base GA and as usual, we make 20 replications. The summary of the search is given below in Table A.1:

Table A.1. Mean, best, and worst fitness values of 20 runs with Base GA configuration and their associated (K_P, K_I, L)

	Mean fitness	Best fitness	Worst fitness
Base GA	20.005	20.004 (0.5266, 5, 0.05)	20.006 (0.2, 5, 0.01)

The fitness values compared to the search for continuous exponential delay case is much higher. In fact, all of the search results fail to stabilize the system and all show growing oscillations. Figure A.3 shows the behavior of best and worst parameter sets:

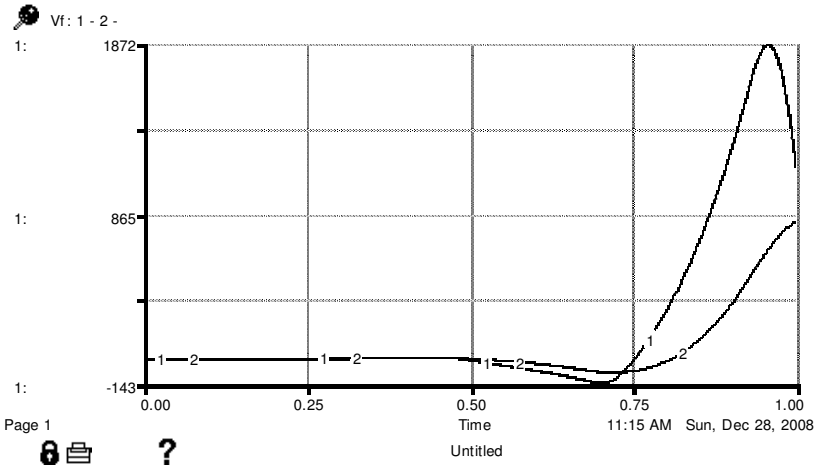

Figure A.3. Behavior with 1: Best and 2: Worst parameter sets among all runs

The search results hint that PI controller cannot stabilize the system when the delay type is pure discrete delay. Next, we look at whether PI control structure is able to stabilize the system at a certain delay value. For this analysis, we fix the values of K_P, K_I, and L to 1, 1, and 0.03 respectively. We know that this set of values is able to control the system when the delay type is continuous exponential. Then, we decrease the delay value starting from 0.225 until 0.001 when the system is finally stabilized. In these runs, DT is set to 0.0001. Any delay value smaller than 0.001, the system shows declining oscillatory behavior with these fixed set of parameter values.

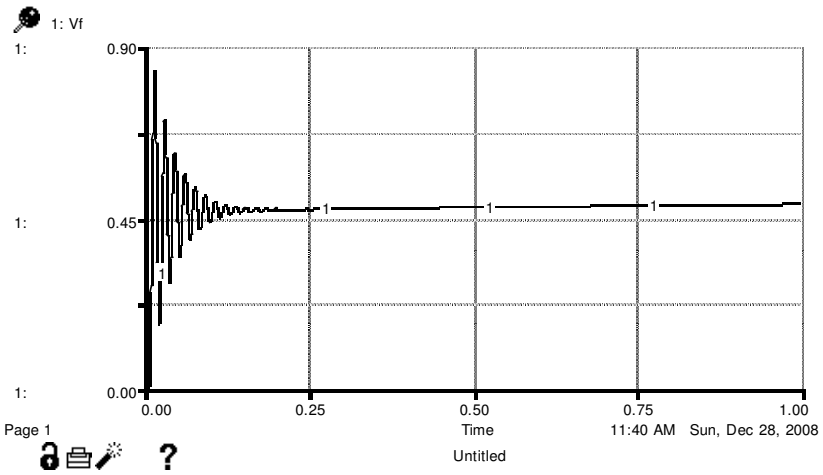

Figure A.4. Final voltage (V_F) behavior with K_P= 1, K_I =1, and L=0.03, delay set to 0.001

The results indicate that pure delay can cause the system to be unstable. Furthermore, the system can only be stabilized with the PI control structure if the pure delay time is short enough.

APPENDIX B: ANALYSIS OF THE STOCK MANAGEMENT MODEL

Equations of the Stock Management Model given in the Figure 4.35 are as follows:

*Stock(t) = Stock(t - dt) + (Acquisition_Flow_2 - Loss_Flow) * dt*
INIT Stock = 4
INFLOWS:
*Acquisition_Flow_2=Supply_Line_2*Order_of_Supply_Line/(Acquisition_delay_time)*
OUTFLOWS:
Loss_Flow = 2

*Supply_Line_1(t) = Supply_Line_1(t - dt) + (Control_Flow - Acquisition_Flow_1) * dt*
INIT Supply_Line_1 = 4
INFLOWS:
*Control_Flow = Loss_Flow+WS*Stock_Adjustment+WSL*Supply_Line_Adjustment*
OUTFLOWS:
*Acquisition_Flow_1=Order_of_Supply_Line*Supply_Line_1/(Acquisition_delay_time)*

*Supply_Line_2(t)=Supply_Line_2(t-dt)+(Acquisition_Flow_1-Acquisition_Flow_2)*dt*
INIT Supply_Line_2 = 4
INFLOWS:
*Acquisition_Flow_1=Order_of_Supply_Line*Supply_Line_1/(Acquisition_delay_time)*
OUTFLOWS:
*Acquisition_Flow_2=Supply_Line_2*Order_of_Supply_Line/(Acquisition_delay_time)*

Acquisition_delay_time = 4
Desired_Stock = 5
*Desired_Supply_Line = Loss_Flow*Acquisition_delay_time*
Order_of_Supply_Line = 2
Stock_Adjustment = (Desired_Stock-Stock)/Stock_Adjustment_time
Stock_Adjustment_time = 4
Supply_Line = Supply_Line_1+Supply_Line_2

Supply_Line_Adjustment=(Desired_Supply_Line-Supply_Line)/Stock_Adjustment_time
WS = 1
WSL = 1

The parameter search indicates that it is possible to obtain improvements in settling time with allowing very small overshoot. In this appendix, we further analyze the trade off between overshoot and settling time. The effect of T_{SA} on the system is included in the analysis.

Table B.1. Overshoot and settling time with different T_{SA} values $W_S/W_{SL}=1$

W_S	W_{SL}	T_{SA}	t_s	Overshoot
1	1	4	7.43	0
1	1	4/3	1.3	0
1	1	1/2	0.1	0
1	1	1/10	9.94	0

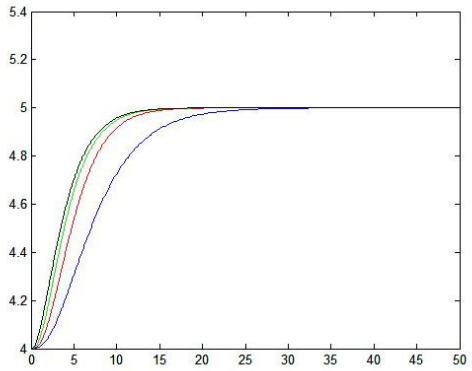

Figure B.1. Behavior of Stock under the parameter values given on Table B.1

Table B.2. Overshoot and settling time with different T_{SA} values $W_S/W_{SL}=1.5$

W_S	W_{SL}	T_{SA}	ts	Overshoot
0.75	0.5	4	16.1	0.0043
0.75	0.5	2	10.09	0.0365
0.75	0.5	1	11.42	0.0523
0.75	0.5	2/3	6.66	0.0479
0.75	0.5	1/4	5.88	0.0256
0.75	0.5	1/10	5.75	0.0164

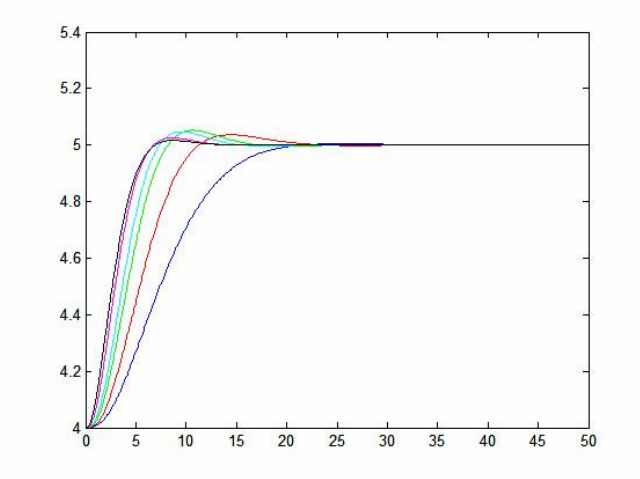

Figure B.2. Behavior of Stock under the parameter values given on Table B.2

The analysis shows that the increase in W_S/W_{SL} ratio causes quicker responses at the expense of higher overshoot. However, when the overshoot or the ratio is too high, the stock passes the boundary twice rather than once. In our case, when the overshoot is greater than 0.05, crosses the band more than once. Hence, the settling time is longer (see Table B.3).

Table B.3. Overshoot and settling time with different T_{SA} values $W_S/W_{SL}=2$

W_S	W_{SL}	T_{SA}	ts	Overshoot
1	0.5	4	11.4	0.0492
1	0.5	2	14.93	0.1139
1	0.5	1	11.6	0.139
1	0.5	2/3	10.27	0.1317
1	0.5	1/4	8.16	0.0855
1	0.5	1/10	7.14	0.0587

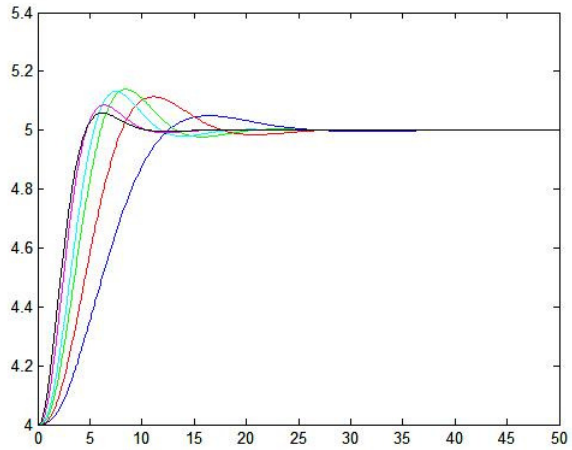

Figure B.3. Behavior of Stock under the parameter values given on Table B.3

The results indicate that, decreasing T_{SA} or increasing the weights W_S, and W_{SL} over 1 cause the system to respond faster (see Figures B.1, B.2, and B.3). In cases with overshoot, decrease of T_{SA} while keeping the weights same causes overshoot to increase until a certain value is reached. From that point on, the decrease of T_{SA} causes overshoot to decrease together with settling time (see Figures B.2, and B.3).

Table B.4. Change in overshoot and t_s with altering W_S/W_{SL}

W_S	W_{SL}	ts	Overshoot	Change in t_s w.r.t W_S, W_{SL}=(1,1)
1	1	17.43	0%	
1	0.8	14.46	0.29%	-17.04%
1	0.6666	12.95	1.65%	-25.70%
1	0.5	11.4	4.92%	-34.60%

Table B.4 displays that, without decreasing T_{SA} substantial increase in settling time can be achieved with allowing very small overshoot by adjusting W_S, and W_{SL}. This section is a direct result of parameter search according to a performance index. It enables the designer to state the goals clearly and their associated weights. To summarize, if the designer does not want any overshoot, then equal weights of stock and supply line adjustment should be used. The designer should keep in mind that faster response of stock level can be obtained by allowing small overshoot.

APPENDIX C: EQUATIONS OF THE MARKET GROWTH MODEL

*backlog(t) = backlog(t - dt) + (entered - completed) * dt*
INIT backlog = 50000
INFLOWS:
entered = Orders_Booked
OUTFLOWS:
*completed = BacklEffect*NormCompl*

*delivdelrec(t) = delivdelrec(t - dt) + (correct) * dt*
INIT delivdelrec = delivdelayin
INFLOWS:
correct = (delivdelayin-delivdelrec)/smthtime

*salesmen(t) = salesmen(t - dt) + (hirefire) * dt*
INIT salesmen = 50
INFLOWS:
hirefire = (budget/salary-salesmen)/sat

BacklRatio = backlog/NormBacklog
baseEffectiv = 100
*budget = Orders_Booked*revenue*
DelDelRatio = delivdelrec/MinDelDel
delivdelayin = backlog/completed
MinDelDel = 1
NormBacklog = 25000
NormCompl = 10000
*Orders_Booked = salesmen*Sales_effectiveness*
revenue = 10
salary = 1000
*Sales_effectiveness = Deleffect*baseEffectiv*
sat = 20

smthtime = 8

BacklEffect = GRAPH(BacklRatio)

(0.00, 0.02), (0.333, 0.4), (0.667, 0.73), (1.00, 1.00), (1.33, 1.24), (1.67, 1.47), (2.00, 1.65), (2.33, 1.75), (2.67, 1.85), (3.00, 1.92), (3.33, 1.96), (3.67, 1.98), (4.00, 2.00)

Deleffect = GRAPH(DelDelRatio)

(0.00, 4.00), (0.5, 4.00), (1.00, 4.00), (1.50, 3.84), (2.00, 3.64), (2.50, 3.46), (3.00, 3.14), (3.50, 2.52), (4.00, 2.00), (4.50, 1.36), (5.00, 0.78), (5.50, 0.52), (6.00, 0.38)

APPENDIX D: BRUTE FORCE OPTIMIZATION OF ELECTRIC CIRCUIT MODEL

In this section, we compare the brute force optimization with the base GA performance for the nonlinear electric circuit model. The parameter boundaries are between 0.01 and 5 for both *L* and *C*. We divide range space of *L* and *C* into equal parts with precision 0.01. Hence, we have 500 points for both *L* and *C*. The brute force approach evaluates the fitness function 500x500 = 250000 times. The best solution of the brute force optimization is (*L, C*) = (2.18, 3.87). The corresponding fitness value is 0.0078. Recall, GA found the fittest individual as b_{ij}^{min*}= (2.1833, 3.8522) with fitness f_{ij}^{min*}=0.00116. The difference between the desired and the actual period of the GA solution is 0.00011 whereas the error in periods for the brute force solution is 0.0069. This difference in period error makes up almost all of the difference in fitness values. For Base GA, only two out of 20 replications of the results are worse than the solution obtained by brute force optimization. None of the solutions of GA6 is worse than the fitness value of the brute force optimization.

Since best GA result is at (2.1833, 3.8522), it would be wrong to expect the brute force approach would find the exact best solution at (2.18, 3.85). In fact, the fitness value of the point (2.18, 3.85) is 0.0085. Thus the precision of the value of the policy parameter that can be set in real life application should be the same as the precision of the parameter during the search. Hence, if the precision of *L* and *C* can only be 2 digits, then the algorithmic search should be done in increments of 0.01.

Contrary to GA, brute force optimization checks all the points in the search space. Hence, the run time is likely to explode with growing search space. Even a simple case with two parameters evaluates the fitness function 250000 times. (This corresponds to 50 replications of a GA with population 50 and number of generations 100). Run time takes 280 minutes approximately. A single GA replication takes 400 seconds for nonlinear electric circuit model (see Table 5.3). This means 10 replications of GA would take 130 minutes approximately. The run time does not depend on number of parameters for GA. However, run time grows exponentially with increasing number of parameters in brute force optimization.

APPENDIX E: ANALYSIS OF FAN-MOTOR MODEL

Consider Fan-Motor model with P control:

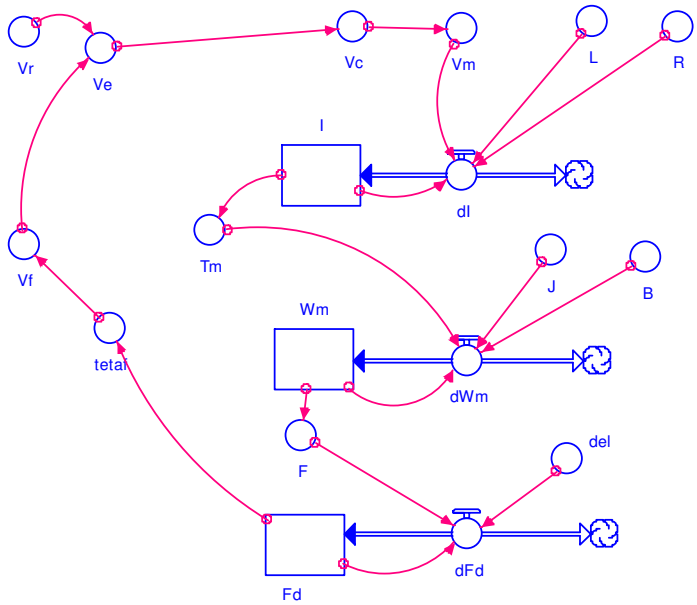

Figure E.1. Fan-Motor model with P control

Equations of the model are as given below:

*Fd(t) = Fd(t - dt) + (- dFd) * dt*

INIT Fd = 0

OUTFLOWS:

dFd = (Fd-F)/del

*I(t) = I(t - dt) + (- dI) * dt*

INIT I = 0

OUTFLOWS:

*dI = -(-R/L*I+Vm/L)*

*Wm(t) = Wm(t - dt) + (- dWm) * dt*

INIT Wm = 0

OUTFLOWS:

*dWm = -(-B*Wm/J+Tm/J)*

B = 0.015

del = 0.225

F = Wm

J = 0.0002

L = 0.03

R = 5

tetaf = Fd

Tm = I

Vc = Ve

Ve = Vr-Vf

Vf = tetaf

Vm = Vc

Vr = STEP(0.5,0)

Notice some parameters such as *Vm, tetaf, Tm, F* are in the model for their physical meaning. They are usually equal to some constant *c* times a variable. Those constants are all taken as equal to one for computational purposes. The Laplace transform of the system is given as:

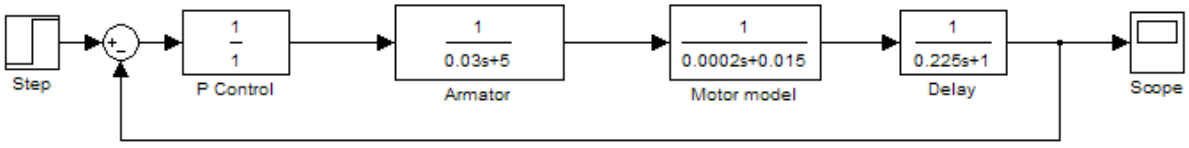

Figure E.2. Laplace Transform of P-controlled FM system

Notice the proportional control gain K_P is chosen to be 1. Then the transfer function *T(s)* can be calculated as:

$$T(s)=\frac{V_f(s)}{V_r(s)}=\frac{(1\cdot\frac{1}{0.03s+5}\cdot\frac{1}{0.0002s+0.015}\cdot\frac{1}{0.225s+1})}{1+(1\cdot\frac{1}{0.03s+5}\cdot\frac{1}{0.0002s+0.015}\cdot\frac{1}{0.225s+1})} \qquad \text{(E.1)}$$

When we do the simplification and further calculation:

$$T(s)=\frac{V_f(s)}{V_r(s)}=\frac{740740.74}{s^3+246.11\cdot s^2+13574.07407\cdot s+796296.296} \tag{E.2}$$

The factorization of the denominator gives:

$$T(s)=\frac{V_f(s)}{V_r(s)}=\frac{740740.74}{(s+197.84)(s+24.133-58.672i)(s+24.133+58.672)} \tag{E.3}$$

The transfer function shows that the system is stable since the roots of the polynomial in the denominator, characteristic polynomial, is on the left side of the s-plane. Since V_r is a step function, its Laplace transform is $V_r(s) = 1/s$. Then the behavior of V_f(s) is given as:

$$V_f(s)=\frac{740740.74}{s(s+197.84)(s+24.133-58.672i)(s+24.133+58.672)} \tag{E.4}$$

Using partial fractions method:

$$V_f(s)=\frac{0.1113779}{s+197.84}+\frac{0.93}{s}+\frac{0.409437+0.356183i}{s+24.133-58.672i}+\frac{-0.409437+0.356183i}{s-24.133+58.672i} \tag{E.5}$$

Then the inverse-Laplace transform gives:

$$V_f(t)=0.1113779\cdot e^{-197.84t}+0.93+e^{-24.133t}(-2\cdot 0.409437\cdot\cos(58.672t)-2\cdot 0.356183\cdot\sin(58.672t)) \tag{E.6}$$

This is the exact mathematical solution of the behavior of the output voltage given a step input of one.

Next, we discuss some other PI structures constructed in SD interface. In the first one, cumulative error corresponding to I of the PI control is delayed, it is called PercErr. Then Vc is calculated similarly by the addition PercErr (delayed I) and dVe (P):

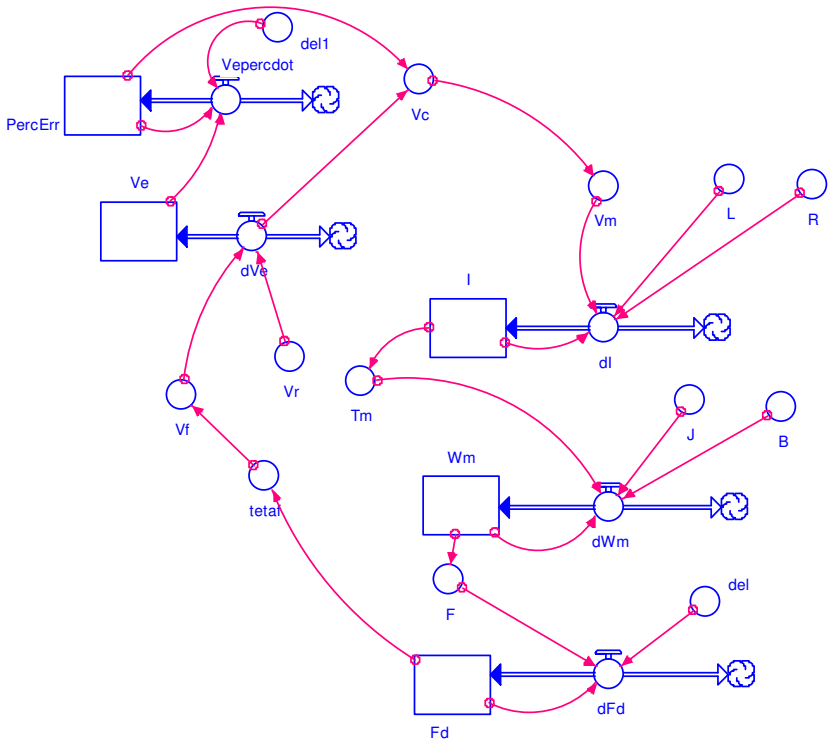

Figure E.3. PI with delayed I for FM model

The second one is PI with delayed P, meaning the error between the measured voltage and the desired voltage affects the control signal after a first order continuous delay. The control signal is denoted as Vcdot and is equal to the addition of percVedot and Ve.

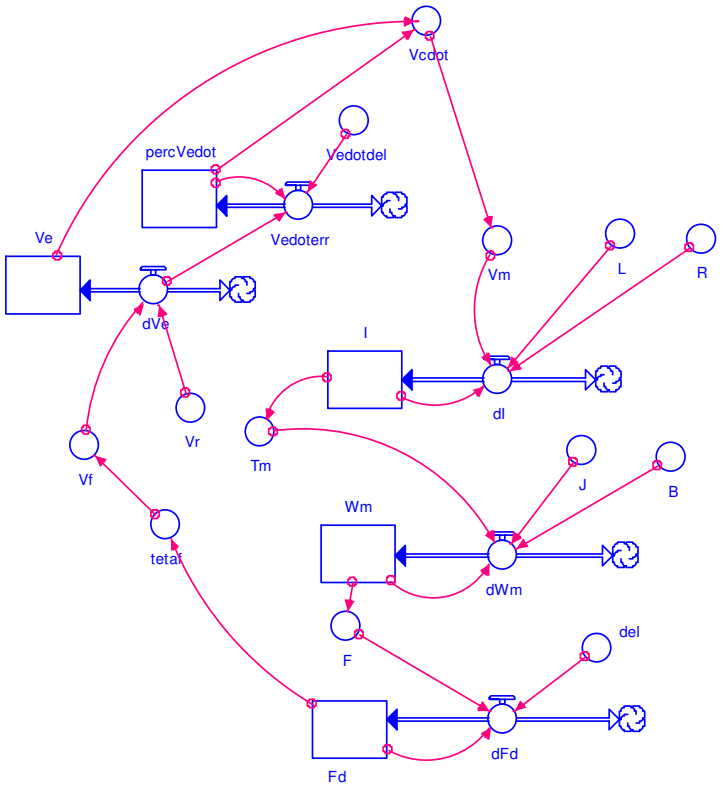

Figure E.4. PI with delayed P for FM model

The last one is a delayed version of PI structure. The delay is applied after the calculation of the control signal. Vcdot in this structure is equal to Vc in Figure 4.9 (i.e. addition of Ve and dVe):

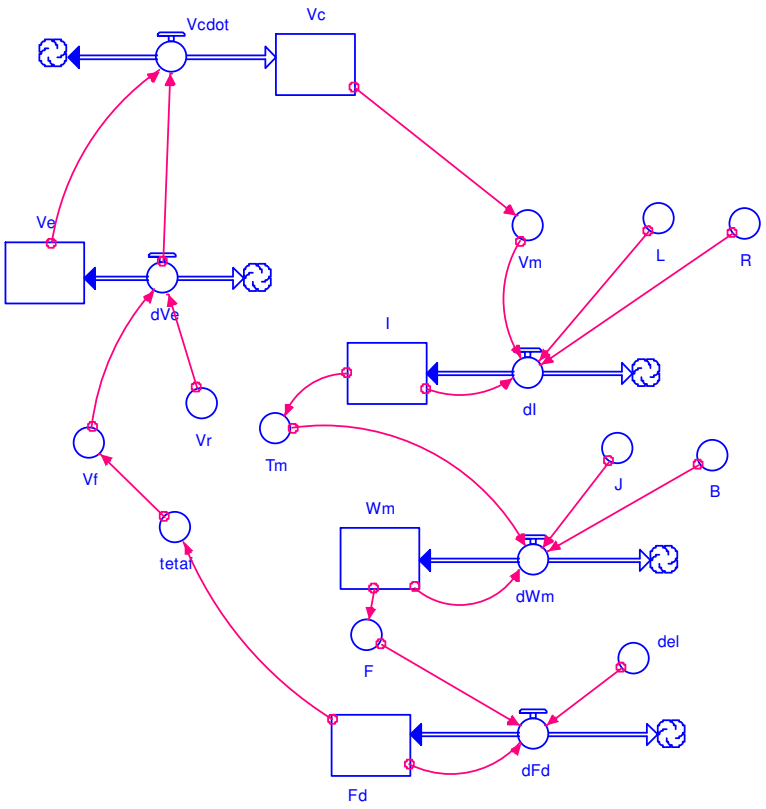

Figure E.5. Delayed PI for FM model

All of these constructed control structures are able to stabilize the system.

REFERENCES

Bäck, T., 1996. *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press.

Chen, Y.T., B. Jeng. 2004. *Policy Design by Fitting Desired Behavior Pattern for System Dynamics Models.* Internet Source.

Clark, R.R., 2004. *An Introduction to dynamical systems: continuous and discrete,* Pearson Prentice Hall.

Coyle, R.G. 1977. *Management System Dynamics.* New York: Wiley.

Coyle, R.G. 1985. The use of optimization methods for policy design in a system dynamics model. *System Dynamics Review*. 1(1): 81-91.

Coyle, R.G. 1996. *System Dynamics Modelling: A Practical Approach*. Chapman & Hall.

Dangerfield, B. and C. Roberts. 1999. Optimization as a statistical estimation tool: An example in estimating the AIDS treatment-free incubation period distribution. *System Dynamics Review* **15**(3): 273-291.

De Jong, K. 1975. *Analysis of Behavior of a Class of Genetic Adaptive Systems.* Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

Duggan, Jim. 2008. Using system dynamics and multi objective optimization to support policy analysis for complex systems. In *Understanding Complex Systems,* 59–81. Springer Berlin/Heidelberg.

Demir, O. C. Eksin, and A. U. Aktaş. "Application of Control Techniques to a Plant with Under-damped Response." Paper presented at the final term project of Control System Design Course, Istanbul, Turkey, May 5, 2005.

Forrester, J. 1961. *Industrial Dynamics,* MIT Press, Massachusetts.

Graham, A.K., and Ariza, C.A. 2003. Dynamic, hard and strategic questions: using optimization to answer a marketing resource allocation question. *System Dynamics Review,* 19(1): 27-46.

Grossmann, B. 2002. Policy Optimization in Dynamic Models with Genetic Algorithms. International System Dynamics Conference, Palermo.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Kuo, B.C., 1991. *Automatic Control Systems.* Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.

Macedo, J. 1989. A reference approach for policy optimization in system dynamics models. *System Dynamics Review,* 5(2): 148-175.

McSharry, P. E. 2004. Optimisation of system dynamics models using genetic algorithms. *World Bank Report,* May 11.

Miller, J. H. 1998. Active Nonlinear Tests (ANT) of Complex Simulation Models. *Management Science* **44**(6): 820-830.

Mohapatra, P.K.J., and S. K. Sharma. 1985. Synthetic design of policy decisions in system dynamic models: a modal control theoretical approach. *System Dynamics Review*. 1(1): 63-80.

Oops English Introduction. Online Image. 16 June 2008. <http://www.myoops.org/twocw/mit/NR/rdonlyres/Global/6/665DD43B-A252-47AC-9925-72337D7A4F6C/0/chp_magnetlevitate.jpg>

Özveren, C.M., and J.D. Sterman. 1989. Control theory heuristics for improving the behavior of economic models. *System Dynamics Review*. 5(2): 130-147.

Porter, B. (1969). *Synthesis of Dynamical Systems.* Newton, J.J.: Nelson.

Reeves, C.R., and J.E. Rowe. 2003. *Genetic Algorithms: Principles and Perspectives- A Guide to GA Theory.* Kluwer Academic Publishers.

Sterman JD. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Irwin McGraw-Hill: Boston, MA.

Wolstenholme, E.F. 1986. Algorithmic control modules for system dynamics models. *System Dynamics Review.* 2(1): 1-19.

Wolstenholme, E. F. and A.S. Al-Alusi. 1987. System dynamics and heuristic optimisation in defense analysis. *System Dynamics Review*. 3(2): 102-115.

Wolstenholme, E.F. 1990. Systems Enquiry: A System Dynamics Approach. John Wiley & Sons.

Yaşarcan, H. 2003. Feedbacks, Delays And Non-linearities in Decision Structures. Industrial Engineering Dept. Istanbul, Boğaziçi University. **PhD**.

Yaşarcan, H., and Y. Barlas. 2005. A generalized stock control formulation for stock management problems involving composite delays and secondary stocks. *System Dynamics Review*. 21(1): 33-68.

Yücel, G., and Y. Barlas. 2007. Pattern Based System Design/Optimization. International System Dynamics Conference, Boston.

REFERENCES NOT CITED

Burns, J., and D. Malone. 1974. Optimization Techniques Applied to the Forrester Model of the World. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-4 (2): 164-171.

De Jong, K. 2006. *Evolutionary Computation A Unified Approach.* MIT Press.

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, Addison-Wesley.

Keloharju, R. 1982. *Relativity Dynamics.* Series. A-40. Helsinki: School of Economics.

Keloharju, R. and E. F. Wolstenholme. 1988. The basic concepts of system dynamics optimization. *Systems Practice* **1**(1): 65-86.

Marion, J.B., and S.T. Thornton. 1995. *Classical Dynamics of particles and systems,* Saunders College Publications.

Richardson, G. P. 1999. Reflections for the future of system dynamics. *Journal of The Operational Research Society.* 50(4): 440-449.

Van de Vegte, J. 1994. *Feedback Control Systems.* Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.