Policy Design by Fitting Desired Behavior Pattern for System Dynamics Models

Yao-Tsung Chen[a], Bingchiang Jeng[b, *]

[a]Institute of Information Science, Academia Sinica, No. 128, Sec. 2, Yanjiouyuan Rd., Taipei City 115, Taiwan
[b]Department of Information Management, National Sun Yat-sen University, 70 Lien-hai Rd., Kaohsiung City 804, Taiwan

Abstract

The paper proposes a policy design method for system dynamics models based on neural network and genetic algorithms. Algorithmic approaches to policy design traditionally are accomplished by either optimization or modal control methods, which achieve the designer's goal through an indirect way. The approach presented here instead is more directly. A model designer can specify any desired behavior pattern and let the learning algorithm to point out where to consider for changes. It needs no objective function as required by optimization algorithms nor suffers the limitations of linearization and complex control mechanism as modal control approaches usually do. The approach is based on our previous work that shows a system dynamics model (i.e., a flow diagram) is equivalent to a specially-design partial recurrent network which both operate under the same numerical propagation constraints. Several experimental studies are conducted to evaluate performance of the new approach. The results show that it is at least as effectiveness as other competent approaches but more convenient and straightforward.

[*] Corresponding author. Tel:+886-7-5252000 ext. 4719; fax: +886-7-5254579.
E-mail Address: ytchen@mail.iis.sinica.edu.tw(Y.-T. Chen), jeng@mail.nsysu.edu.tw(B. Jeng)

1. Introduction

The utmost goal of studying system dynamics is to develop policies that improve the dynamic behavior of a system [Coyle, 1996]. Traditional or analytic (called by Porter [1969]) approaches to policy design rely on a domain expert's insight and experiences to solve the problem in a trial-and-error style [Starr, 1980; Coyle, 1977]. Thus, strategies derived for some specific models are usually hard to be generalized. Talavage [1980] points out that traditional practices of policy design are time-and-money-consuming. Synthetic approaches [Porter, 1969], on the other hand, use algorithmic methods to automatically search for a candidate policy, and are generally applicable to an arbitrary model.

Current synthetic approaches are based on two categories of theory: (1) **optimization theory** and (2) **modal control theory**. The former generates policies through a process of searching for a combination of (all) parameter values in order to optimize an objective function (not system behavior) [Burns and Malone, 1974; Coyle, 2000]; the latter changes the behavior patterns of a model by shifting the eigenvalues of the differential equations that define the model [Talavage, 1980; Mohapatra and Sharma, 1985].

Each of the above approaches has strengths and weaknesses. The success of optimization algorithms depend on assigning an appropriate objective function so that the optimization result will reflect in changes of behavior as user specified. If choosing a wrong objective function, the changes indicated by the algorithm may not necessarily result in the expected model behavior. In a multi-criteria condition, identifying such an objective function is not an easy task (i.e., "black art" by Coyle [1996; 1999]).

On the other hand, approaches using modal control theory can directly modify a

system's behavior. But again the relationships between the eigenvalues and the system behaviors are not straightforward; it requires strong mathematical training background in order to do so. Besides, the approaches can only adjust overall behavior patterns. It cannot fine-tune individual variable's behavior as may specified by users.

In this paper, we present an algorithm that is different from the above two approaches but inherits their strengths; it adjusts a system's behavior directly without the needs of an objective function or changing eigenvalues. The idea behind is based on our previous research [Chen and Jeng, 2002], which demonstrates that a system dynamics model is equivalent to a specially designed neural network. Given this, we therefore view policy design for a system dynamics model as a learning problem, and a genetic algorithm (an evolutionary approach) based on the neural network model is applied.

To use this kind of algorithms, one needs first to encode the problem, i.e., to translate the problem/solution into a code string that represents a chromosome, and define how the operations of mutation and cross-over are implemented. Therefore, in our problem, we have to encode a structure of a system dynamics model into a string, and this is not easy. Fortunately, we have shown that a system dynamics model is a partial recurrent network. Thus, the easiest way is to encode the network structure and there are already many methods doing this [Prusinkiewicz and Lindenmayer, 1992]. We'll show this in the later of the paper. The remaining of the paper is organized as follows. Section 2 is a brief survey of past approaches in policy design. Section 3 introduces the SDM-PRN model transformation and the genetic algorithm approach to generating policies by fitting the desired system behavior pattern. Section 4 presents an experimental study that evaluates the performance of this new approach. The last section concludes the paper.

2. Review of Traditional Methods

According to Forrester's [1968a] description, policies are "… the rate equations … that tell how 'decisions' are made. The policy (rate equation) is the general statement of how the pertinent information is to be converted into a decision." Traditionally, policies and rate equations are usually synonymous terms. Designing a new policy is an activity of (1) assigning alternative values for parameters, (2) changing linkages among system elements, and/or (3) inserting alternate elements into a model [Starr, 1980].

As it is mentioned, synthetic policy design methods fall into two categories: optimization theory and modal control theory. We briefly introduce each of them here.

Optimization is the act of obtaining the best result under given circumstances. On design, construction, and/or maintenance of an engineering system, engineers have to make many technological and managerial decisions at different stages. The ultimate goal of all such decisions is either to minimize the required effort/cost or to maximize the desired benefit/utility. Since cost or utility can be expressed as an objective function of certain decision variables, optimization can be defined as a process of searching for the conditions that generate the maximal or minimal value of the function. The existence of optimization methods can be traced to days of Newton, Lagrange, and Cauchy [Rao, 1996]

Several optimization methods have been used in policy design to modify system behavior patterns according to a user's expectation. Burns and Malone [1974] are possibly the first pioneers who apply optimization algorithms in the context of system dynamics modeling. They propose two algorithms. One uses the Powell method [1964], which searches the conjugate directions of the matrix of system equations, to

determine a set of optimal parameter values; the other uses the steepest descent algorithm [Bryson and Ho, 1969] to determine the direction of a set of time-varied parameter trajectories that optimizes the objective function. Both algorithms are applied to the Forrester model of *World Dynamics* [1973], and the latter is reported to have a better performance. However, the first method is later adopted by the software package VENSIM [1994].

Different from the Powell method, the search algorithm used in DYSMAP Optimizer [Keloharju, 1983] (which utilizes the pattern search strategy [Hooke and Jeeves, 1961]) is simpler. It only allows one parameter value be modified at each iteration in the search procedure. This method has been applied to many applications. Coyle [1996; 1999] uses it to solve the problem of a domestic manufacturing company. Wolstenholme and Al-Alusi [1987] apply it to a military defense model in order to generate a better formation change strategy for the attacking force in response to the fire strategies made by the defending force. Kivijärvi and Tuominen [1986] use it to solve economic optimal control problems. Keloharju and Wolstenholme [1989] show the value of DYSMAP Optimizer in the optimization of a project management model.

Kleijnen [1995] proposes a heuristic optimization method, which is later called the robust concept exploration method (RCEM) by Bailey et al. [2000]. Treating a system dynamic model as a black box, it first creates a set of regression equations to fit the input (or parameter)/output (or system state, level) patterns of the model. The statistical Design Of Experiments (DOE) is applied to determine which parameters are significant. It drops the insignificant parameters, views the regression equations as the response surfaces (or constraint), and then optimizes the objective function using the traditional Lagrange multiplier method. The parameter values obtained through the procedure are the final solution. A case study in the Wolstenholme's coal

transportation model [Wolstenholme, 1990] shows the advantages of the approach.

Bailey et al. [2000] enhance RCEM and use it in a high-level model of an industrial ecosystem. They do not view the solution of RCEM as a final answer, but instead, consider it as a guide to the real one. Their approach identifies the exploration points surrounding the solution of RCEM and then find a set of real best-combination of parameters from them.

Finally, Grossman [2002] proposes a genetic algorithm that searches the solution space in multiple and random directions, and demonstrates his algorithm in the Information Society Integrated Systems Model.

Except RCEM, all other methods described above treat the optimization problem as a search problem in a solution space. The only difference is in searching directions, which determine efficiency and effectiveness of the algorithms. The Powell method might be the most efficient since it searches only in the conjugate directions. On the other hand, Grossman's genetic algorithm might be the most effective since it explores the largest solution space and does not get trapped in a local optimum.

All methods reported better performance than an experts' analytical approach. However, there are a couple of weaknesses in these approaches [Coyle, 1996; 1999]. First, the algorithms do not always guarantee to find an optimal solution. The other and the most serious problem is that the objective function for a problem is not easily to choose, which means that a poor objective function might be "truly disastrous" [Coyle, 1996; 1999].

Another mainstream of synthetic policy design methods applies the modal control theory. As Mohapatra and Sharma [1985] stated, a linear dynamical system can be represented by a state differential equation of the following type:

$$\mathbf{x}(t+\Delta t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (1)$$

where **x** is a n*1 vector of state or level variables, **u** is a m*1 vector of control (or

policy or rate) variables, **A** is an n*n system matrix, and **B** is an n*m matrix.

If the control vector **u** is a linear feedback of the state vector **x** according to the following control law

$$\mathbf{u}(t) = \mathbf{F}\mathbf{x}(t) \quad (2)$$

where **F** is the feedback matrix of dimension m*n, then Eq. 1 can be represented as

$$\mathbf{x}(t+\Delta t) = (\mathbf{A} + \mathbf{BF})\mathbf{x}(t) = \mathbf{A_1}\mathbf{x}(t) \quad (3)$$

where $\mathbf{A_1}$ is an n*n closed-loop system matrix.

Wonham [1967] has shown that if a system with equations like Eq. 1 is state controllable, then the closed-loop system matrix $\mathbf{A_1}$ can be assigned with any set of eigenvalues by an appropriate choice of the feedback matrix, **F**. In addition, Porter and Crossley have shown how to derive control rules of shifting eigenvalues, from locations λ_1, λ_2,..., λ_m, of the **A** matrix to desired locations, ρ_1, ρ_2,…, ρ_m. Since eigenvalues determine the response characteristics of a system behavior, ability to generate the desired eigenvalues makes it possible to produce a desired system behavior pattern [Mohapatra and Sharma, 1985].

Based on the modal control theory, Talavage [1980] describes a procedure called MODEMAP that uses a piecewise-linearization process to linearize the model. In addition, he shows an example to apply the method to a version of the market-growth model described by Forrester [1968b].

Mohapatra and Sharma [1985] also present a policy design approach based on the modal control theory and use the production and stock system as an example to illustrate the method. In their approach, policy variables are treated as control variables by isolating them from other variables, which leads to a greatly simplified model that is linear. With a reduced system being linear and controllable, control policies then is generated synthetically by their method to ensure any prescribed degree of stability.

Although the modal control theory approaches can generate desired patterns, they still suffers a problem when being applied in policy design. That is, there is no explicit relationship between the patterns generated by the modified eigenvalues and the behavior that a policy designer wants to generate. This makes the approaches difficult to be applied. In addition, these approaches can only adjust the overall behavior patterns. It cannot fine-tune individual variable's behavior as may specified by users.

3. Policy Design by Fitting a Desired Behavior Pattern

In our previous work [Chen and Jeng, 2002], we have shown that a traditional flow diagram for a system dynamic model can be equal to a specially-designed partial recurrent neural network. Since the latter is adaptable, we can therefore view policy design for a system dynamics model as a learning problem for the neural network structure in order to fit a specified (output) behavior pattern. Our learning method here is a genetic algorithm based on the neural network model. In the following section, we will show how to do this. But before that, we need to give a brief touch how the SDM-PRN Transformation looks like.

3.1. The SDM-PRN Transformation

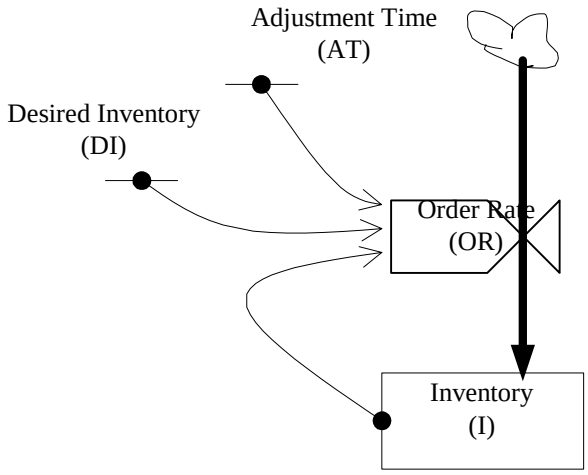


Figure 1 An inventory model

Shown in Fig. 1 is a simple flow diagram for an inventory control system. In this model, there is a decision point (Order **Rate;** *OR*) that controls the **flow** into a **level** (Inventory; *I*). The goal is to minimize the difference between desired inventory (*DI*) and *I* within adjustment time interval (*AT*), where *DI* and *AT* are constants and propagated to *OR* through **wires**.

The numeric equations/constraints related to the system in Fig. 1 are

$$I(t) = I(t - DT) + (OR) \times DT$$
$$OR = (1/AT) \times (DI - I)$$
$$DI = 6000$$
$$AT = 5$$

We have shown that a system dynamics model can be transformed into a partial recurrent neural network through the algorithm FD2PRN [Chen and Jeng, 2002].The above model in neural network form will be look like Fig. 2(c). This diagram may be hard to understand at a first sight without knowing its structure. It composes of three parts: the top left two nodes are input units, which feed data into the network at initialization; the bottom two nodes are output units; and the top right two nodes are state units, which keep the previous values of the output units. The relationship with the original model is following. As illustrated in Fig. 2(a), **level** (inventory: *I*) and **constant** (*DI*) are each mapped to three units: **input** I_I, **output** O_I, **state** S_I ; and **input**

I_{DI}, **output** O_{DI}, **state** $S_{DI,}$ respectively. **Rate** (*OR*) and the **flow**, as shown in Fig. 2(b), are mapped to a hidden unit H_{OR}, and the link from H_{OR} to O_I, respectively. The other type of **constants** (e.g., *AT*) that appear as a coefficient in a rate equation is mapped to the weights of the links from S_I to H_{OR} and from S_{DI} to $H_{OR,}$ respectively, as shown in Fig. 2(c).

For your reference, the details of the individual relationships between the corresponding components in two models are listed in Table 1. Not shown in Fig. 2 are the values that propagate within the network. We have shown also that the equations between the two models are the same. However, in order to maintain the clarity of the paper, we will omit this part here. Interested readers may find the proof of the equivalence in detail from [Chen and Jeng, 2002].

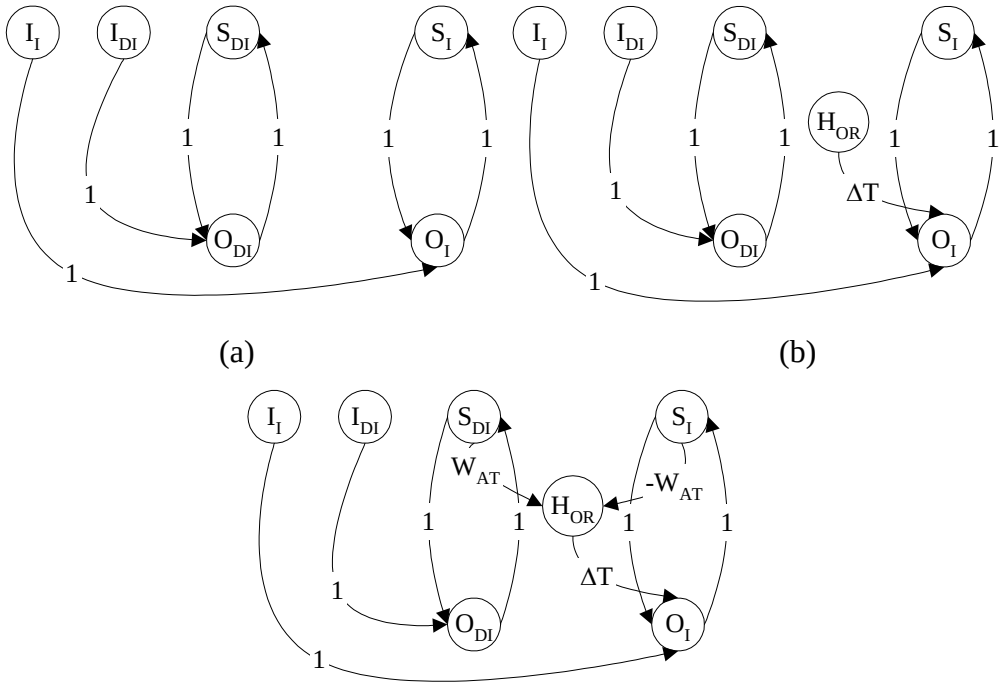


Figure 2 The inventory model in PRN format

Table 1 The component mappings between SDM and PRN

Components for SDMs	*Components for PRNs*
Level variable, constant (not for coefficient)	*A triple of input, output, and state units*
Rate (or auxiliary) variable	*Hidden unit*
Wire	*Link from a state unit to a hidden unit*
Flow	*Link from a hidden unit to output unit*
Level equation	*A weighted sum of the values of hidden and state units connecting to an output unit via links*
Rate equation (including constants as coefficients)	*Any function of the values of state units connecting to a hidden unit via links*
Equation for initial value	*Link from an input unit to an output unit*
Constant equation	*Link from a state unit to an output unit*

3.2. Genetic Algorithms

Genetic algorithms (GAs), first introduced by John Holland for the formal investigation of the mechanisms of natural adaptation [Holland, 1975], are now widely applied in science and engineering as adaptive algorithms for solving problems. Any problem that is suitable to be solved by the generate-and-test paradigm can use this approach. A unique characteristic of genetic algorithms is that they create an environment to mimic the biological evolution process and let various combinations of parameters related to the problem to compete with each other in order to generate the best breed of solution without the intervention of human beings. Such an algorithm requires the creation of a special kind of data called “chromosome” which includes the whole information that is needed to solve the problem. So it is a solution in fact. The algorithm initially generates a set of such solutions as the first generation. Then the biological evolution process comes in and each solution is evaluated based on its “fitness”. Those with a higher “fitness” score receive a better chance in the competition to win the right to breed their next generation using the operations of cross-over and mutation; otherwise, they die. (“Survival of the fittest”, as introduced by Darwin [1859]) The process stops either after a certain generation or

at a special termination criterion. The best one for the final solution is then chosen from the pool of the created chromosomes.

Because of the unique way of solving problems, GA is particularly known to be suitable for multidimensional global search problems where the search space potentially contains multiple local optima. Unlike other search methods, it does not converge to local optima and correlations between the search variables are generally not a problem either. In addition, GA basically does not require extensive knowledge of the search space, such as solution bounds or functional derivatives, etc. since it solves problems by “guess”, not through an analytical process. This fact ensures that GAs may be readily applied on fitness landscapes (or potential surfaces) which may contain discontinuities or singularities without any special treatments. Past research has shown that genetic algorithms are applied very successfully in various areas and different applications [Koza and et al., 2003].

Under the same process, however, there is much flexibility in the detail implementations of the algorithm. For example, someone might prefer to apply the mutation operator only to generate new individuals without a selection operator from the entire population. Others might choose from different selection schemes like "Roulette wheel" selection, tournament selection, random selection, or stochastic sampling. There is also a multitude of methods to create a new individual from two parents. Crossover is the process by which two parents get combined but there are many ways to implement. The most commonly used form is single-point crossover. Another alternative is chromosome mixing, where intact chromosomes are randomly swapped. As a random algorithm, there are also parameters to be set, i.e., mutation rate, crossover rate, etc. There is no agreement in the GA community over which is the preferred variation; their relative performance is dependant upon the particular problem [Adcock, 2003].

3.3. Fitting a Desired Behavior Pattern

Now we introduce our method how to make policy design become a learning problem. We will explain it in four steps in the following subsections.

3.3.1. Genetic Encoding

The first step of using genetic algorithms in problem solving is encoding, i.e., to encode a solution of the problem as a chromosome. In our case, the solution is a SDM, and to encode it directly is not easy. Since we have rephrased a SDM in terms of a PRN representation, we have a choice by encoding the neural network structure instead, and there are existing ways to do it [Gruau and Whitley, 1993, Prusinkiewicz and Lindenmayer, 1992].

The encoding schemes can be divided into direct ones and indirect ones. In the former, each link weight of a network is directly encoded. In the later, only some characteristics of a network are encoded, such as the number of hidden layers, the number of hidden nodes in each layer, the number of connections between two layers, etc [Yao, 1999]. For simplicity, our current approach will only encode the network link that represents a wire into the chromosomes. In addition, those that currently do not exist but have the potential to be considered later in policy design will also be included. On the other hand, the ones that cannot be modified in policy design will be excluded from the code since they can be treated as constants. At this moment, we will not consider the possibility of introducing new network nodes or changing a rate equation, which touches the boundary problem of policy design.

How to represent a chromosome is the third problem. Previous way to this is to represent it into a binary format. But this will create a long string with imprecise information [Holland, 1975]. So later approaches use real numbers to represent the

chromosome, and so is ours. We will let each weight in a wire correspond to a real number in the code.

3.3.2. The Fitness Function

In order to evaluate a solution (i.e., chromosome) generated by the genetic algorithm, we need to define a fitness function. The one we used here is very straightforward, just a set of desired behavior patterns derived from a policy designer's intention for system variables. The patterns specify what outcomes s/he wants the system to generate. The fitness function then measures the closeness between the actual pattern generated by the solution and target pattern, and the closer the higher of the fitness score. Our equation is based on the reciprocal of SSE (Sum of Squared Error):

$$fitness = \frac{1}{\sum_i \sum_t (y_{it} - \hat{y}_{it})^2},$$

where y_{it} is an desired output, $\hat{y}_{it}$ is the real output of a chromosome, and t represents the tth time point, i represents the index of a variable.

The policy designer may multiply the SSE by a weight w_i to emphize the importance of the variables, i.e.

$$fitness = \frac{1}{\sum_i w_i \sum_t (y_{it} - \hat{y}_{it})^2}$$

There are many possible ways to prepare the desired behavior patterns. The policy designer can either edit existing behavior data by him- or herself or use any tool to create them. However, there is a heuristic rule in preparation of the data depending on the type of a problem. If the goal is to search for a policy that will generate a stable trajectory, then a flat line is enough to represent the intention. Otherwise, the goal is to search for a growing policy and a growing trajectory has to

be generated, e.g. constant growth, exponential growth.

3.3.3. The Evolution Process

Having defined the fitness function, the following step is to set up the evolution process to reproduce the individuals generation by generation. The first parameter to be chosen is the size of population, which determines the number of parallel searching path. It's the bigger the faster to find better solutions but more computation efforts. We set the population size to 120 and the evolution process will stop at the 1000th generation. Then we define the schemes and parameters of operators. There are three operators in GA: selection, mutation and crossover. When a chromosome wins the right to enter the next generation, it has three possible choices depending on a stochastic process. One is untouched and enters the next generation pool directly; the second is modified by a mutation operation; and the third is to combine with another one through a crossover operation [Adcock, 2003]. In this research, we choose only second and third which are defined by following scheme and rate: selection scheme is Stochastic Sampling, mutation scheme is single-point randomize, crossover scheme is double-point crossover, crossover rate is 0.9, and mutation rate is 0.3. The detail of scheme described above can be referred in [Adcock, 2003].

3.3.4. Generating a Policy

The evolution process stops either because a certain number of generations has reached or a termination criterion is satisfied. After that, we need to inspect whether the best one(s) that generated from the evolution process is (are) a good one. If yes, then use that one as the final solution for the problem and output the content of the policy for the human designer to consider; otherwise, another evolution process starts. Genetic algorithms are a kind of stochastic searching processes. So it may require

more than a certain number of trails in order to obtain an expected outcome. However, there is a limitation of the trials and endless of loop is meaningless. A usual convention is to repeat the evolution for 10 times.

If the result is still unsatisfied, there is another choice in our method, which is to trigger the learning algorithm for neural networks. Remember that our encoding scheme is based on the PRN model, which is a neural network. So we can further fine-tune the solution with a gradient-descent learning algorithm. It is well known that genetic algorithms are good at large range searching but inappropriate for something that needs iterative fine-tuning. Another benefit of our approach is that it combines both of the advantages of genetic algorithms and neural networks in solving problems.

4. Experimental Study

We will use the model of *World Dynamics* to test our method, which is developed by Forrester [1973] in order to model the real world as a system of continuous flows of time-varying commodities interrelated by complex nonlinear feedback and coupling mechanisms. The model consists of a set of equations that describe the interactions among five level variables: population (P), natural resources (NR), pollution (POL), capital investment in agriculture fraction (CIAF) and capital investment (CI). Variables P, POL, and CI have both input and output follows; NR has only an output follow; and CIAF has only an input follow. According to the model's description, policy is designed 70 years after the model starts running at the time of 1900. The policy goal aims at reducing NR consumption while maintaining CI, P and POL. Besides, an additional variable QL (Quality of Life) is defined as an indicator, which should be at least maintaining at the level of 1970 year. It is not a level variable that participates the operations and feedbacks to the system. Instead, it is more like an objective function to pursue. In our experiment, it corresponds to an

additional unit added to the PRN and generating an output behavior trajectory. There are five adjustable parameters, which are: BRN (Birth Rate Normal), NRUN (Natural Resource Usage Normal), POLN (Pollution Normal), FC (Food Coefficient), CIGN (Capital Investments Generation Normal). Their original parameter setting is given in the second column of Table 3, and Forrester's parameter setting is listed in the third.

4.1. Generating the Original Policy

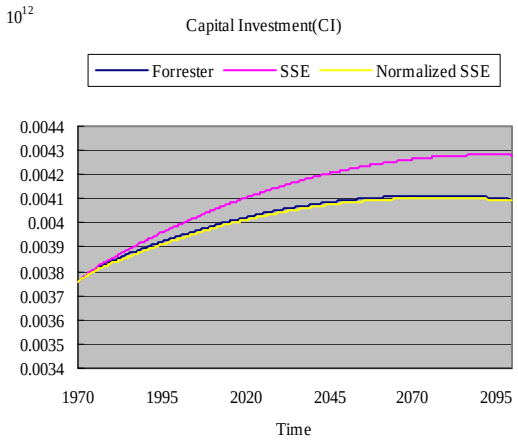
In order to test whether our method can really generate policies without any need of domain or prior knowledge, we first use the proposed policy given by Forrester for the model as the learning target (their parameter values are listed in the third column of Table 2). Then we start the execution of our algorithm as described in section 3. The execution stops at the 1000th Generation, and the best solution from the one thousand generations is selected. The whole process iterates 10 times. Each requires about 30 minutes on an Intel Pentium III 750 machine. The result of each iteration in fact is very close, and the best one is shown in the 4^{th} and 5^{th} column of Table 2. The difference between the two columns is that the 4^{th} column uses the fitness function based on the original SSE without normalization while the 5^{th} column does.

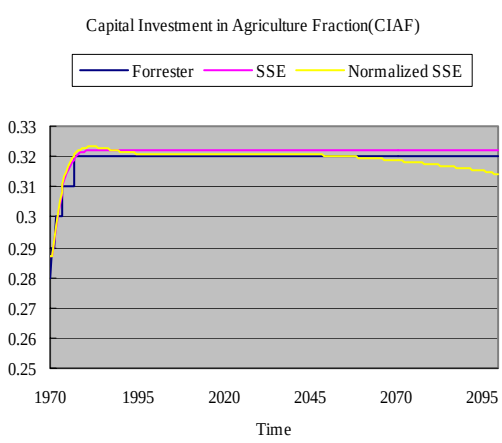
Taking a close look at the table, one can see that the parameter values are very close to each other. The three policies generated are drawn in Figure 3 where only the one from the 4^{th} column has a little large difference in POL variable. This is because variable QL, CIAF, and NR have relative large values than others and that biases the fitness function to favor the solutions that have close trajectories with three variables but ignore others. This is evidence that weighting a different variable can affect the final policy the algorithm generates. The unbiased version given in the 5^{th} column shows that the policy generated by our algorithm does match the one by Forrester. Except variable P, all other variables have almost coincident trajectories with

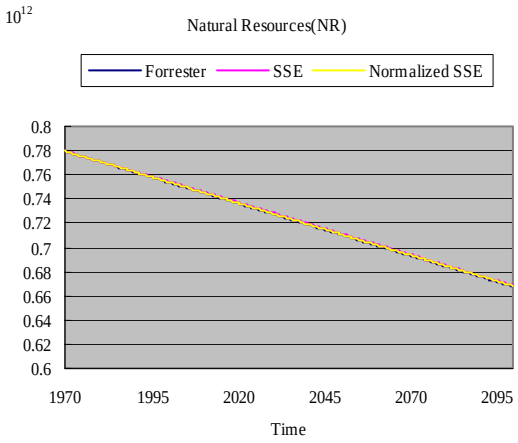
Forrester's ones.

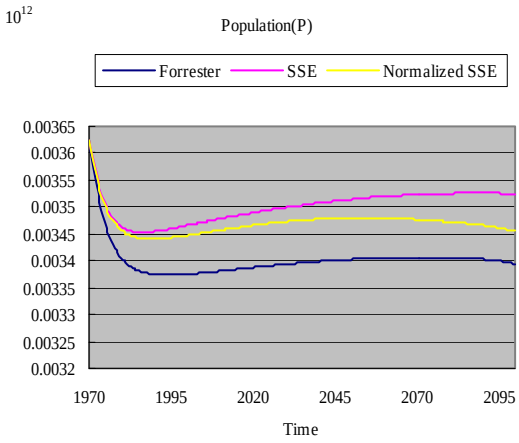
Table 2 comparison of parameter settings

Parameters	Original	Forrester	Evolve from Forrester's Trajectories (SSE)	Evolve from Forrester's Trajectories (Normalized SSE)
BRN	0.04	0.04*0.7=0.028	0.02815	0.02754
NRUN	1	1*0.25=0.25	0.24676	0.24775
POLN	1	1*0.5=0.5	1.11789	0.51352
FC	1	1*0.8=0.8	0.84451	0.84095
CIGN	0.05	0.05*0.6=0.03	0.02992	0.02981









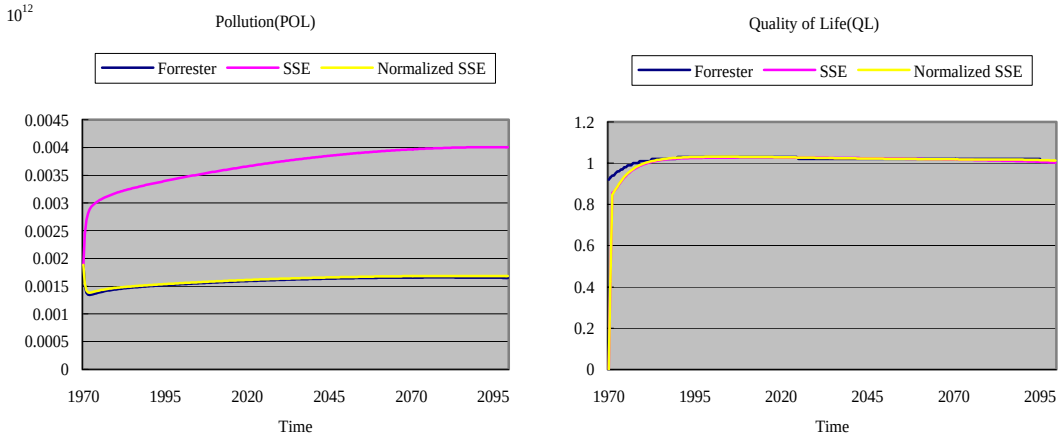


Figure 3 Comparison of model behaviors

4.2. Exploring Policy Limitation

Generating a good policy is important. But it is sometimes equally important to know the limitations of policy that can be taken. So we want to further explore the potential of our algorithm. Our question is simple. Can we obtain an even better policy if using a higher expectation trajectory for QL as the target? We start another two experiments and the results are compared with those by Burns and Malone [1974], who use Powell method to generate the policy.

Burn and Malone conduct two experiments. The "Burns and Malone 1" uses the original model of *World Dynamics* as the starting point of their optimization process while the "Burns and Malone 2" uses Forrester's policy as the starting point. The results of their experiment results are listed in 2rd and 3th column in Table 3. Burns and Malone's objective function consists of three terms: Max QL, Min P's fluctuation, and Min the changes in parameter values. Its equation is following:

$$Min\ I = \int_{t_0}^{t_f} -QL^2 dt + \int_{t_0}^{t_f} 10^{-16} \cdot (P - 3.767 \times 10^9)^2 dt + \int_{t_0}^{t_f} \left\| u - 1 \right\|^2 \mathbf{w} dt$$

We also conduct two experiments. "OURS 1" is trained with the exemplar data that increases QL's level up to 30% but maintains the same trajectory; "OURS 2"

changes the QL trajectory to continuing increasing. Except that, all other variables' trajectories are untouched. The results of the experiments are listed in the 4th and the 5th column in Table 3.

Table 3 comparison of parameter settings

Parameters	Burns and Malone 1	Burns and Malone 2	OURS 1	OURS 2
BRN	0.04*0.6=0.024	0.04*0.64=0.0246	0.02240	0.00988
NRUN	1*0.52=0.52	1*0.44=0.44	0.22891	0.20061
POLN	1*1.16=1.16	1*1.02=1.02	0.42672	0.44410
FC	1*0.89=0.89	1*.0.84=0.84	0.76039	0.45825
CIGN	0.05*0.71=0.0355	0.05*0.64=0.032	0.03177	0.02786

Model behaviors defined by these different parameter settings are drawn in Fig. 4. As shown in the figure, the QL trajectory of OURS1 does reappear the behavior extracted from the training data and generate a stable trajectory (i.e., does not drop) which is higher than all previous approaches including Burns and Malone's. The QL trajectory of OURS2 is even more interesting; it follows the trainer's expectation and keeps growing as is shown in the training trajectory. This result outperforms all known policies and is much better than Burns and Malone's one.

Based on the result of the experimental study, we can safely state that the PRN representation indeed is beneficial to the applications of system dynamics. The two experiments show that the structure of the new representational form can be adapted according to the will of a policy designer, simply by assigning appropriate exemplar training data. The method requires neither mathematical background (as required by some optimization methods) nor any insight about the target domain (as required by a manual approach) in order to generate a policy, but its performance in the experiments is much better than that by an optimization algorithm (e.g., Burns and Malone's) or by a human expert (e.g., Forrester's).

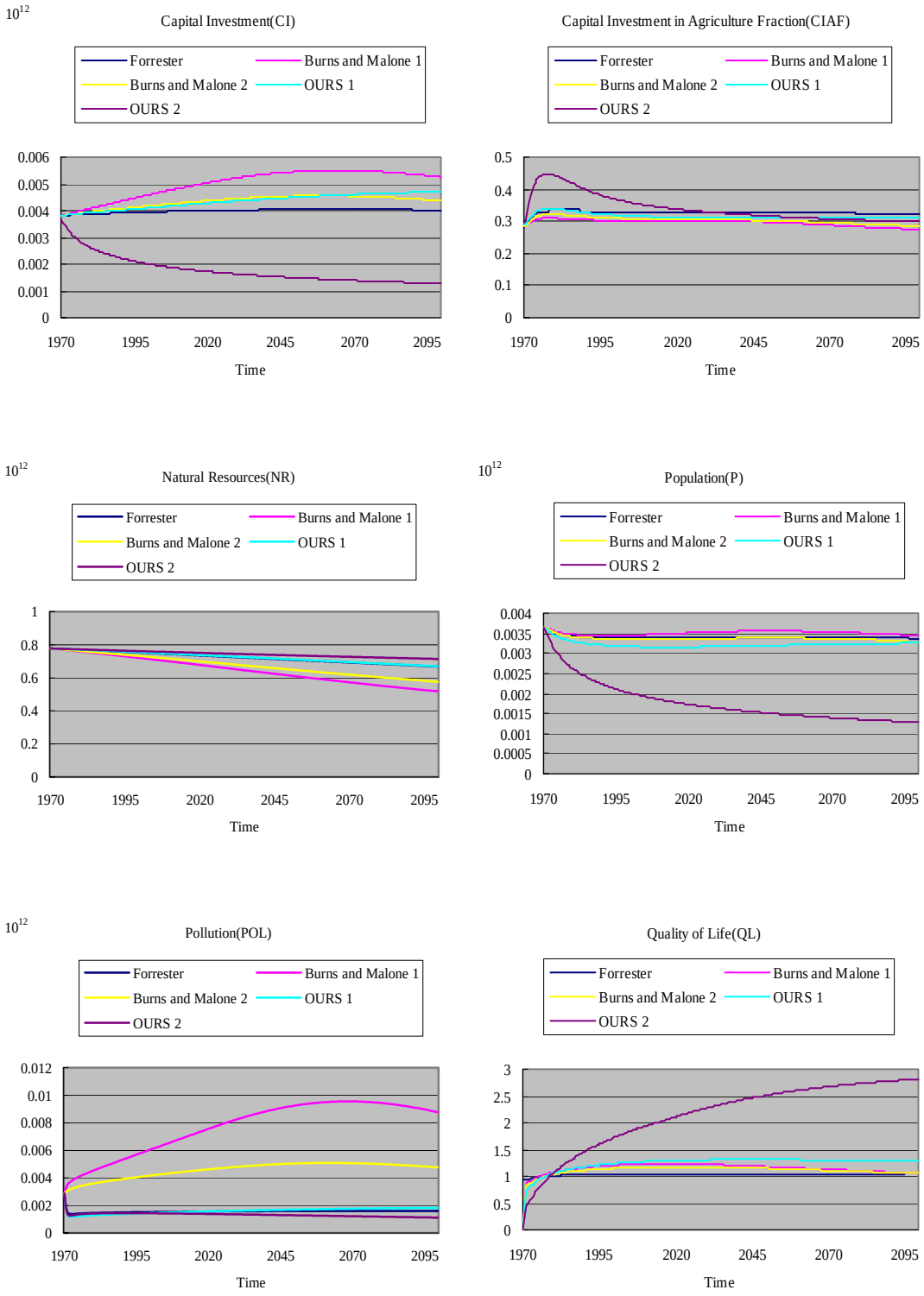


Figure 4 Comparison of Model Behaviors Induced by Different Parameter Settings

5. Conclusion

This paper has presented a policy design method which is based on partial recurrent networks and fitting desired behavior patterns. First, the model in which a policy to be designed is transformed into a partial recurrent network and then GA is applied to learn a better parameter setting. The model of *World Dynamics* is used to test our method. The results described in section 4 show that our method indeed design customized parameter settings based on expectation trajectories given by policy designers. In the same concept, the method may apply in model construction, too. Since the combination of PRN and GA can find the parameter setting based on expectation trajectories, it may find the parameter setting based on real-world trajectories. However, the real-world data contain many noises and the structure of model is not sure in model construction stage. Further research will be conducted in this direction.

Acknowledgements

This research was partially supported by National Science Council of Taiwan, under contracts NSC-89-2416-H-110-033.

Reference

Adcock, S. 2003. GAUL Online Document. http://gaul.sourceforge.net/.

Bailey, R., B. Bras, and J. K. Allen. 2000. Using Response Surfaces to Improve the Search for Satisfactory Behavior in System Dynamics Models. *System Dynamics Review* **16(2)**:75-90.

Bryson, A. E. Jr., and Y. C. Ho. 1969. *Applied Optimal Control*. Waltham, Mass.: Blaisdell.

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

Chen, Y.-T. and B. Jeng. 2002. Yet another Representation for System Dynamics Models, and Its Advantages. In *Proceeding of the 20th International Conference of the System Dynamics Society*. Palermo, Italy.

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

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

Coyle, R. G. 1999. Simulation by Repeated Optimisation. *Journal of the Operational Research Society* **50(4)**:429-438.

Darwin, C. 1859. *The Origin of Species*. London: Murray.

Forrester, J. W. 1968a. *Principles of Systems*. Cambridge, Mass.: MIT Press.

Forrester, J. W. 1968b. Market Growth as Influenced by Capital Investment. *Sloan Management Review* **9**:83-105.

Forrester, J. W. 1973. *World Dynamics*. Cambridge, Mass.: Wright-Allen Press, Inc.

Grossman, B. 2002. Policy Optimization in Dynamic Models with Genetic Algorithms. In *Proceeding of the 20th International Conference of the System Dynamics Society*. Palermo, Italy.

Gruau, F., and D. Whitley. 1993. Adding Learning to the Cellular Development of Neural Networks. *Evolutionary Computation* **1(3)**:213-233.

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

Hooke R., and T. A. Jeeves. 1961. 'Direct Search' Solution of Numerical and Statistical Problems. *Journal of the Association for Computing Machinery* **8**:212-229.

Keloharju, R. 1983. *Relativity Dynamics*, Helsinki School of Economics, Acta Academiae Oeconomicae Helsingiensis, Series A:40.

Keloharju, R., and E. F. Wolstenholme. 1989. A Case Study in System Dynamics Optimization. *Journal of the Operational Research Society* **40(3)**:221-230.

Kivijärvi, H., and M. Tuominen. 1986. Solving Economic Optimal Control Problems with System Dynamics. *System Dynamics Review* **2(2)**:138-150.

Kleijnen, J. P. C., 1995. Sensitivity Analysis and Optimization of System Dynamics Models: Regression Analysis and Statistical Design of Experiments. *System Dynamics Review* **11(4)**:275-288.

Koza, J. R., M. A. Keane, and M. J. Streeter. 2003. Evolving Inventions. *Scientific American* **288(2)**:52-59.

Mohapatra, P. K. J. and Sharma, S. K. 1985. Synthetic Design of Policy Decisions in System Dynamics Models: A Modal Control Theoretical Approach. *System Dynamics Review* **1**: 63-80.

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

Powell, M. J. D. 1964. An efficient method for finding the minimum of function of several variables without calculating derivatives. *Comput. J.* **7**:155-162.

Prusinkiewicz, P., and A. Lindenmayer. 1992. *The algorithmic Beauty of Plants.* Springer-Verlag.

Rao, S. S. 1996. *Engineering Optimization*. New York: Wiley.

Starr, P. J. 1980. Modeling Issues and Decisions in System Dynamics. *TIMS Studies in the Management Science* **14**:45-59.

Talavage, J. J. 1980. Modal Analysis to Aid System Dynamics Simulation. *TIMS Studies in the Management Sciences* **14**:229-240.

VENSIM 1994. Belmont Mass.: Ventana Systems Inc.

Wolstenholme, E. F. 1990. *System Enquiry: A System Dynamics Approach*. New York:

Wiley.

Wolstenholme, E. F., and A. -S. Al-Alusi. 1987. System Dynamics and Heuristic Optimization in Defense Analysis. *System Dynamics Reviews* **3(2)**:102-115.

Wonham, W. M. 1967. On Pole Assignment in Multi-Input Controllable Linear Systems. *IEEE Transactions on Automatic Control* **12**:660-665.

Yao, X. 1999. Evolving Artificial Neural Networks. *Proceedings of the IEEE* **87(9)**: 1423-1447.

Back to the Top