Working Paper Series

Adaptive Agents in the House of Quality

Thomas Fent

Working Paper No. 43
July 1999

Working Paper Series

July 1999

SFB
'Adaptive Information Systems and Modelling in Economics and Management Science'

Vienna University of Economics
and Business Administration
Augasse 2–6, 1090 Wien, Austria

in cooperation with
University of Vienna
Vienna University of Technology

http://www.wu-wien.ac.at/am

This piece of research was supported by the Austrian Science Foundation (FWF) under grant SFB#010 ('Adaptive Information Systems and Modelling in Economics and Management Science').

Abstract

Managing the information flow within a big organization is a challenging task. Moreover, in a distributed decision-making process conflicting objectives occur. In this paper, artificial adaptive agents are used to analyze this problem. The decision makers are implemented as Classifier Systems, and their learning process is simulated by Genetic Algorithms. To validate the outcomes we compared the results with the optimal solutions obtained by full enumeration. It turned out that the genetic algorithm indeed was able to generate useful rules that describe how the decision makers involved in new product development should react to the requests they are required to fulfill.

keywords: *adaptive systems, artificial intelligence, classifier systems, genetic algorithms, house of quality, quality function deployment, total quality management*

1 Introduction

Designing new products is often a process that involves several departments of a company. Moreover, the people dealing with all aspects of launching a new product have varying educational backgrounds and, therefore, do not speak the same technical language. In some organizations it can indeed be a big challenge to manage the communication required to capture all aspects of product development. However, nowadays decreasing product life-cycles require an efficient communication in order to launch new products before the competitors do it. An efficient information-flow is an indispensable prerequisite for a high speed of innovation. A well-known technique for interfunctional planning and communications is the *house of quality* introduced by (Hauser and Clausing, 1988).

"The foundation of the house of quality is the belief that products should be designed to reflect customers' desires and tastes" (Hauser and Clausing, 1988). A brief summary of the model can be found in (Chase and Aquilano, 1995). The main idea is a conceptual map that helps two departments to collect all the data that express the influence of the decisions of one group on the problem of the other group. The information flow works like it is shown in figure 1.

In the first step the purpose of the house of quality is translating customer attributes into engineering characteristics. Subsequently, the engineering characteristics are transformed into parts characteristics, which are then used to generate the key process operations, which, in the last step, are the

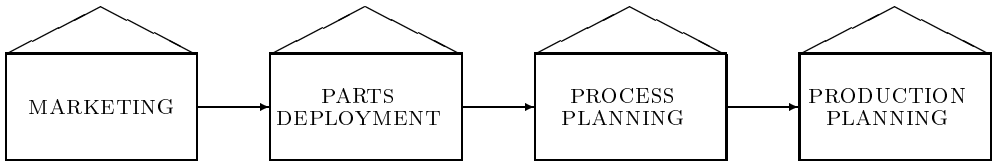

Figure 1: Flow of information

basis of the production requirements. In all these steps data have to be communicated between different departments. The House of Quality approach helps them to talk together.

The remainder is organized as follows. Section 2 explains the model and the algorithms we used. In section 3 we present the simulation results with various parameter settings. Finally, in section 4 we interpret the obtained outcomes and give a summary of some possible extensions we would like to examine in the future.

2 The Model

For our attempt to simulate the behaviour of the individuals involved in the product design process we ascended the basic ideas of the house of quality to a quantitative level. Hence, all the relevant information that is exchanged between the particular departments is supposed to be encoded in a string of fixed length containing only the digits $0, 1, 2, 3$, and 4.

2.1 The customer attributes

The customer attributes are the inputs of the first house. If we run our simulation with strings of length 3, there are 125 (5^3) possible input vectors. To initialize the customers' wishes we create all the 125 input strings, and select 20 of them for the validation set. The remaining strings are used to train the system while the validation data are stored for later comparisons between different simulation runs.

Obviously in case of a real product a vector of length 3 will never be suitable of carrying all the characteristics required to describe the product. However, since the algorithms we used do not take advantage of the structure of the functions that have to be optimized, increasing the size of the vector only increases computing time without any qualitative changes of the obtained results.

2.2 The decision making process

The messages created in the above section enter the first house - the marketing department, where the customer attributes are transformed to engineering characteristics. In our simulation this is done by a classifier system (CS). The output - the engineering characteristics - is the input of the next house - the parts deployment. Yet, another CS uses this information to decide about the parts characteristics. In the same way the third house - process planning - transforms parts characteristics into key process operations, and, finally, these are transformed into production requirements by the department of production planning.

For an introductory description of CSs see (Dawid, 1996, p. 13ff) or (Holland, 1976). Here, we used a very simple CS with only one conditon part per rule and each message entering the CS alone. Thus, a typical situation of our model is illustrated in figure 2, where '#' is the don't care symbol.

message	rules	actions
2 4 1	# 4 1	0 2 4
	3 4 #	2 3 1
	1 3 0	4 1 3
	2 4 1	1 0 2
	# 0 1	3 4 0

Figure 2: A typical decision situation

In this particular situation the first and the fourth rule are fulfilled by the incoming message. Thus, we choose one of them randomly. However, in most situations not all the fulfilled rules have the same chance to be chosen. We use a weight which depends on the strength gained in the past and on the strictivity of the rules. The strictivity depends on the number of don't care symbols appearing in the considered rule. Thus, in this example the fourth rule is more strict than the first one.

Later, the selected rule posts its action part which, in turn, is the input of the following house, where a similar decision making process is applied again. Note that each of the departments has it's own independent set of rules.

A description of more general classifier systems is given in (Geyer-Schulz, 1995), while (Holland, 1995) provides a survey of recent applications.

2.3 Creating new rules

If a message enters a house which does not fulfill any of the available rules, then a new rule has to be created. First we compare the incoming message with each rule. At those indices, where the rule entry does not match with the message we compute the difference and derive the sum of all the differences for each rule. Finally we choose one of those rules with the lowest sum of differences. The action part and the strength (see section 2.4.4) of this rule is inherited by the new rule. The incoming message itself is used as the new condition part.

2.4 Costs and Fitnesses

Each decision taken by any department is assumed to cause two different types of costs. The first cost term represents those costs that originate from implementing the chosen decision while, on the other hand, there are some opportunity costs that arise from the difference between the actually chosen output and the theoretical optimal output with respect to the input.

2.4.1 Implementation costs

The costs of implementation depend on the numerical values of each particular component of the output vector plus a term that represents the correlation or synergy among the components. For instance, if the department chooses the row vector $\vec{x}$, then with certain cost parameters the implementation costs may become

$$c_1 = \vec{x}\,\vec{b} + \vec{x}\,\mathbf{B}\,\vec{x}' \ .$$

In the above formula the matrix $\mathbf{B}$ represents the correlation among the components. Therefore, in the main diagonal and below it there are only zeros. The other entries may be negative, if there are synergies that reduce the costs. However, the inequality

$$B_{ij} \geq -\frac{b_i}{4\ (n-1)} \quad \forall i,j$$

must be fulfilled to avoid negative costs.

2.4.2 Opportunity costs

We assumed that there might exist an affine transformation that assigns an optimal output to each input,

$$x^* = A\ x^{input} + a\,. \tag{1}$$

In order to obtain a feasible solution we have to check if each component is in the intervall $[0, 4]$ and reduce or increase those components that are too big or too small. Finally, the distance between the chosen output and the optimal output determines the opportunity costs,

$$c_2 = \beta \|x^* - x\| \,. \tag{2}$$

Note, that in our model an overshoot is considered to be as bad as a shortfall. In most cases this might obviously make sence. However, if we think about excessive precision this statement is not so clear. In such a situation the costs might become too high and, therefore, it is feasible to assign high opportunity costs to such a solution.

2.4.3 Total costs and fitness

The total costs are the sum of implementation costs and opportunity costs,

$$c = c_1 + c_2 \ .$$

These costs are used to determine the fitness of the chosen rule. Higher costs lead to a lower fitness and vice versa.

Local optimization

In an organization consisting of locally optimizing agents only those costs arising from the particular department are considered to compute the fitness. For simplicity we want to achieve fitness values in the interval $[0, 1]$. Therefore, we first compute the highest total costs possible,

$$\bar{c} = \vec{x}\ \vec{b} + \vec{x}\ \hat{\mathbf{B}}\ \vec{x}' + \beta \cdot ||\vec{x}||,$$

with $\vec{x} = (4, 4, 4)'$, and $\hat{\mathbf{B}}$ containing the absolute values of the entries of $\mathbf{B}$. The parameter β determinines the weight of the opportunity costs. Thus, we are able to control the relative importance of c_1 and c_2. Then we obtain the fitness

$$f = \frac{\bar{c} - c}{\bar{c}} \ . \tag{3}$$

Global optimization

If the agents' aim is to minimize the companys total costs, then also the impacts of their decision on the other departments have to be taken into account. If we denote the highest possible cost in the i-th house with $\bar{c}^i$, and

the costs occuring in the i-th department with c^i, then the fitness value of the rule recently chosen in the j-th department becomes

$$f = \frac{\sum_{i=j}^{4} \bar{c}^i \cdot f_b^{i-j} - \sum_{i=j}^{4} c^i \cdot f_b^{i-j}}{\sum_{i=j}^{4} \bar{c}^i \cdot f_b^{i-j}} . \tag{4}$$

The only difference to formula (3) is that we consider a weighted sum of all the costs in the recent house plus all the following houses. The factors f_b in the above formula are used to adjust the trade-off which has to be made between local and global optimization. A selfish, suboptimizing decision maker will choose $f_b = 0$, while a decision maker only interested in the success of the company as a whole will choose $f_b = 1$. Nevertheless, since not all the costs occuring in subsequent steps are caused by the decision at step i. Thus, we assume that choosing a level of f_b somewhere between 0 and 1 will yield the best results. However, in this paper we only concentrate on the case $f_b = 0$, and leave out the other setups for further investigations.

2.4.4 Strength - apportionment of credit

While training the classifier system, all the fitnesses gained by a rule during a particular iteration are cumulated. Thus, rules which are chosen more frequently also gain a higher strength. After proceeding all the inputs of the training set through the system, the strength is updated according to

$$strength = (1 - \alpha)\ oldstrength + \alpha\ newstrength\ . \tag{5}$$

This is done several times. In our simulation we defined the variable r_p, determining how often the whole set of data has to be proceeded through the system within one generation.

2.5 The genetic algorithm - rule discovery system

After r_p iterations a genetic algorithm is used to adapt the rules in the classifier system. A detailed study about training a classifier system using genetic algorithms was done by (Goldberg, 1989). Our aim is to find rules that enable the departments to find a "good" response to every input.

2.5.1 Selection

In order to select those rules that we use to generate the next population we rank all the rules with respect to their strength received in the past. Then we

choose the best 50%. However, at the beginning we start with a very small population. Thus, during the start we choose all the rules in order to increase the number of individuals. A small example of the selection process is given in figure 3.

population

rules	actions	strength
# 4 1	0 2 4	0.75
3 4 #	2 3 1	0.67
1 3 0	4 1 3	0.84
2 4 1	1 0 2	0.34
# 0 1	3 4 0	0.82

selecting the 3 fittest rules

rules	actions	strength
# 4 1	0 2 4	0.75
1 3 0	4 1 3	0.84
# 0 1	3 4 0	0.82

Figure 3: Selecting the rules

The rules that are selected cannot be changed by the following crossover operation but only by the mutation.

2.5.2 Crossover

First we build pairs with the individuals (rules and actions) selected in section 2.5.1. Then a crossover mask is generated to create two new individuals. Finally, we assign the average fitness as starting fitness value to the new rule. The crossover probability χ is 1, i.e. the crossover process always takes place. The example in figure 4 illustrates the crossover operator.

2.5.3 Mutation

The last operator in our genetic algorithm is the mutation. The aim of this operator is to avoid converging towards a local minimum. Therefore, we first create mutation masks containing only zeros and ones. The probability of ones is the mutation probability μ, which is between 0.001 and 0.004 in our

crossover mask

1 1 0 1 0 1

new individuals build with # 1 and # 3

rules	actions
# 4 1	0 4 4
# 0 1	3 2 0

Figure 4: The crossover operator

simulation. Then we create the random strings containing only $0, 1, 2, 3$, and 4. Finally, we replace the old contents of our population with the random numbers with probability μ, as it is shown in figure 5.

mutation mask

0 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0

new population

rules	actions
# 4 1	0 2 2
1 3 0	4 1 3
# 0 1	3 4 0
# 4 1	0 4 4
# 0 1	3 2 0

Figure 5: The mutation operator

Afterwards the second phase of the mutation operator takes place. If we'd finish our genetic search as described until here, as a result of the mutation operator the number of don't care symbols would decrease from one generation to the next one. To avoid this we apply a special mutation only to write new don't care symbols into the existing rules. This works exactly like the general mutation described above. Another effect of having an extra muta-

tion operator only for don't care symbols lies in the fact that this enables us to control the strictivity of our rules. The parameter determining the probability of writing a don't care symbol into the population is called d_p. In our experiments we tried values between 0 and 0.003.

3 Results

The objective of the departments is to find replies that yield low costs. As a first approach the fitness of a rule is assumed to depend only on those costs that arise in that particular department. That means, all the departments try to find a strategy that minimizes their own costs without considering the indirect effects of their decision. However, it would also be interesting what happens if the fitness assigned to a certain rule is also influenced by the costs in the subsequent departments.

In the simulation we tried 432 different parameter settings. If we observe the development of the average costs in each department over time when the decision makers only get information about their own costs, the time series may look like in figure 6.

The first graph shows the actual costs and the minimum costs in house 1 as a solid line, and the actual and minimum costs of house 2 as a dashed line. The costs caused by the decision chosen by the CS are plotted as a thick line and the minimum costs (caused by the optimal decision) as a thin line. Minimum means that we assume, the decision makers know the cost function and can choose their respond such that the minimum costs arise. The second graph shows the same for the third and fourth house, and, finally, the third picture shows the cumulated actual and minimum costs of all the four houses together.

The generations from 1 to 40 were used to train the CS in all the four houses. Therefore the training set with 107 different input messages was used. From generation 41 to 5 we used the remaining 20 messages to check if the classifiers are capable to deal with new inputs which they did not experience during the training period.

4 Conclusions and extensions

If we look at figure 6, we can see easily, that the algorithm indeed is capable of finding better rules in terms of cost reduction. The minimum costs of the first house do not change, because the input remains the same. On the other hand, the minimum costs of the other three houses change, because they

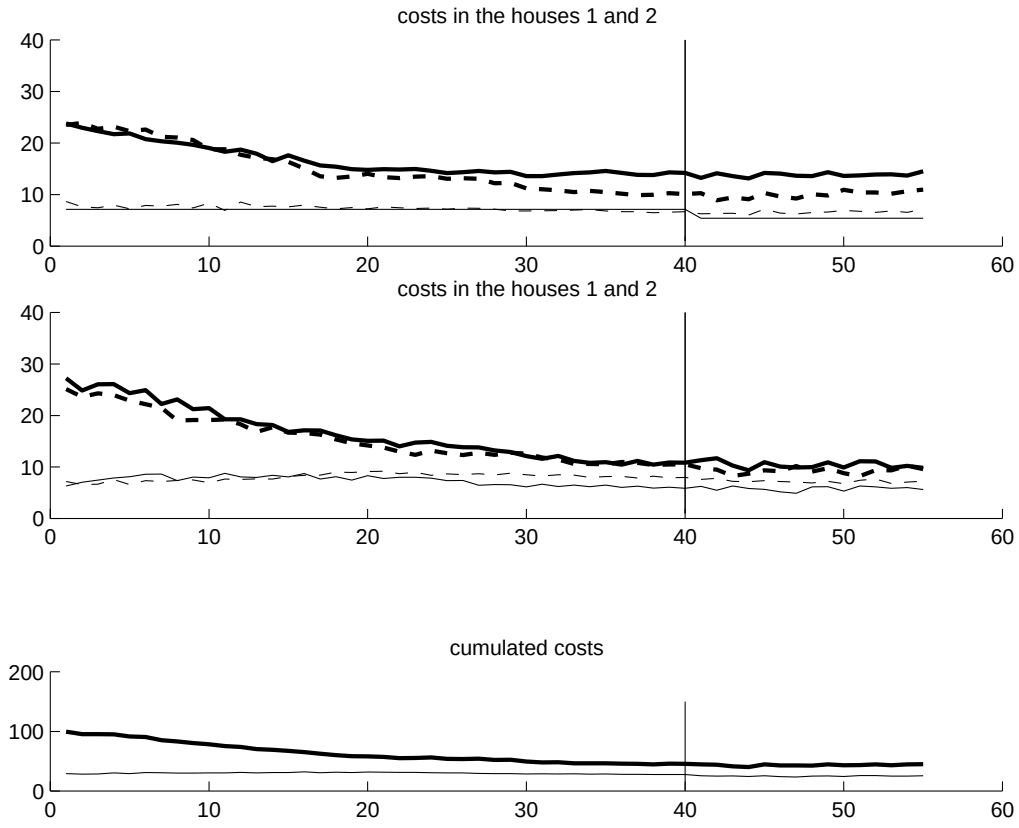

Figure 6: $\alpha = 0.1, r_p = 1, despop = 72, \mu = 0.001, d_p = 0.001$

depend on the output of the previous department as suggested in figure 1. In some situations the minimum costs even decrease as time elapses due to the suboptimizing behaviour of the preceding departments.

The reason why the cost reduction in the first house is smaller than in the other houses lies in the fact that the first house has to find a policy for 107 different inputs with a CS containing only 72 different rules. With this CS it can only produce 72 different outputs which facilitates the others' decision problems. To analyze this effect in more detail we ran the simulation with population sizes of 36, 72, and 108 individual rules in each CS. It turned out that bigger populations yield better results in terms of cost reduction. In the simulation the computing time certainly exceeds when increasing the population size.

Certainly, it would be possible to implement a CS which always finds the optimal solution by increasing the population size until one rule for each possible input is available. However, the purpose of a CS is finding a generalization among the encountered inputs, and establishing a simple policy with satisfying results.

If we analyse the behaviour after the training period - i.e. from generation 41 to 55 - we find out, that this reqirement is fulfilled. The departments get completely new inputs, and though the costs remain reasonable small, and only slight adaptions are made in the sequel.

By comparing the different results it turned out, that a high level of the strength update factor α (in particular in combination with a high rate of repeats r_p) results in very fissured cost courses. Thus, the genetic algorithm did not always move to lower costs as time elapsed. The same holds for high mutation probabilities μ and d_p.

As already indicated in section 3 we would like to observe what happens if the information set is enlarged or reduced. In economic terms this can be interpreted as a change in the firm's organization structure, or a change of the internal incentive system. Moreover, it would be interesting to observe the changes of the reactions to one particular input message and the related costs. Another interesting aspect could be to compare different learning algorithms or just have a look at what happens when some of the genetic operators are added, changed, or removed. Another possible extension would be to create a central CS responsible for all the decisions in the whole company.

References

Chase, R. B. and Aquilano, N. J. (1995). *Production and Operations Management, Manufacturing and Services*, chapter 5, pages 173–180. Richard

D. Irwin, Inc., 7 edition.

Dawid, H. (1996). *Adaptive Learning by Genetic Algorithms, Analytical Results and Applications to Economic Models*, volume 441 of *Lecture Notes in Economics and Mathematical Systems*. Springer.

Geyer-Schulz, A. (1995). Holland classifier sytems. *APL Quote Quad*, 25(4):43–55. ACM SIGAPL.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company.

Hauser, J. R. and Clausing, D. (1988). The house of quality. *Harvard Business Review*, pages 63–73.

Holland, J. H. (1976). Adaption. In *Progress in Theoretical Biology IV*. ed. Rosen, R. F. New York: Academic Press.

Holland, J. H. (1995). *Adaption in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press/Bradford Books edition, Ann Arbor. MI.