ELSEVIER

Expert Systems with Applications 31 (2006) 145–149

Expert Systems with Applications

www.elsevier.com/locate/eswa

An integrated approach to achieving optimal design of computer games

Shang Hwa Hsu *, Feng-Liang Lee, Muh-Cherng Wu

Department of Industrial Engineering and Management, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30010, Taiwan, ROC

Abstract

In a time-to-market environment, designers may not be able to incorporate all the design features in a computer game. For each feature, there are several levels of implementation, which is corresponded to different levels of benefit as well as cost. Therefore, a trade-off decision for determining appropriate levels of implementation is very important, yet has been rarely studied in literature. This paper presents an approach to solve the trade-off decision problem. This approach applies the neural network technique and develops a genetic algorithm to optimize the design of computer games. By this approach, a near-optimal design alternative can be identified in a timely fashion.

Keywords: Neural networks; Genetic algorithms; Optimization; Computer game

1. Introduction

Computer game is a promising industry (Bethke, 2003) and much research on the effective design of computer games has been published in the last two decades. Most of these studies focused on the identification of design factors to make a game fun. Malone and Lepper (1987) argued that a fun game has to involve four characteristics: challenge, fantasy, curiosity and control, which can intrinsically motivate game-players. Based on these characteristics, Fabricatore, Nussbaum, and Rosas (2002) developed a set of detailed design guidelines for action games. A recent study further identified 39 design features that contribute to the fun of an action game (Hsu, Lee, & Wu, 2004).

Attempting to incorporate all the design features in a computer game is infeasible for several reasons. First, time-to-market in this industry is a critical factor for product success because the life cycle of a computer game is quite short. Therefore, incorporating all the design features is generally impossible in a limited time frame. Second, the benefit function of a design feature has a marginal-diminishing effect. That is, the incremental benefit decreases as total investment on implementing a design feature increases. Therefore, it may not be cost-effective to implement every design feature up to its maximum level. Third, most companies in this industry tend to adopt a risk-pooling product strategy, which leads to the development of several games at the same time. The resources allocated to a particular game are rather limited. How to effectively design a computer game in limited time and budget is therefore a very important issue, yet has rarely been investigated in previous literature.

The issue of designing a computer game within time and budget constraint is essentially a selection problem; that is, selecting a design alternative out of a large solution space. For example, a game involves 39 design features. Each feature can be implemented in six levels; the higher the implementation level, the higher the performance as well as the cost. Consequently, the solution space of designing such a game can involve 6^{39} alternatives—a formidable task if an optimal design is to be identified by applying an exhaustive search.

This paper proposes an approach to efficiently identifying a near-optimal design alternative out of the enormous solution space. This approach involves the use of artificial neural network (ANN) technique (Rumelhart, Hinton, & Williams, 1986) and genetic algorithm (GA) (Michalewicz, 1992) to select the design alternative. Specifically, a neural network is established to evaluate the perceived fun of a design alternative. A genetic algorithm is developed to identify a near-optimal design alternative. The identified design alternative suggests the appropriate implementation level of each design feature and facilitates designers to properly allocate their efforts to various design features.

The technique of integrating ANN and GA has been successfully applied in some optimization problems such as engineering design (Marcelin, 2004; Mok, Kwong, & Lau, 2001; Qiu & Li, 2004; Shi, Lou, Lu, & Zhang, 2003; Su, Kao, & Tarng, 2004), management decision (Kuo & Chen, 2004;

* Corresponding author. Tel.: +886 3 5726731; fax: +886 3 5722392.
E-mail addresses: shhsu@mail.nctu.edu.tw (S.H. Hsu), fengl@cc.fec.edu.tw (F.-L. Lee).

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.09.012

Lee, Lee, Lee, Choi, & Lee, 2001; Li, Wu, & Pang, 2004; Yuan & Chen, 2001), and financial planning (Chen & Huang, 2003; Kim & Han, 2000, 2003; Shin & Han, 2000; Versace, Bhatt, Hinds, & Shiffer, 2004). In this study, we attempt to apply the techniques to tackle the problem of designing computer games.

The remainder of this paper is organized as follows. Section 2 describes the modeling of a design alternative, which involves its implementation levels of design features and perceived fun. Section 3 describes how to apply neural network technique to evaluate the perceived fun of a design alternative. Section 4 formulates the research problem. Section 5 presents the genetic algorithm to search for a near-optimal solution. A numerical example is illustrated in Section 6 and concluding remarks are given in Section 7.

2. Modeling of design alternatives

Hsu et al. (2004) conducted a study to identify the design features contributing to fun, which involves 28 action games and the overall perceived fun of each game was rated at one of five levels, ranging from 1 to 5.

The procedure for the identification of design features consisted of three steps. First, experienced game-players were asked to classify the 28 games into three groups according to their perceived fun. Second, each game in the 'fun' group was contrasted with a game in the 'no-fun' group in order to identify the design features that make a game fun. Third, each design feature of a game was rated in terms of its implementation level, ranging from 0 to 5. Their experiment results indicated that an action game is composed of 39 design features.

An action game is essentially a design alternative before completion of a design process. This study models ith design alternative by vector $\bar{x}_i = [x_{ij}]$, $1 \le j \le 39$, where $x_{ij} \in [0, 5]$ represents the implementation level of jth design feature in the design alternative. The overall perceived fun of ith design alternative is represented by $y_i \in [1,5]$.

According to the modeling, the solution space of the optimal design of interest can be represented by $S = \{\bar{x} | \bar{x} = [x_j], x_j \in [0,5], x_j \in Z^+, 1 \le j \le 39\}$, also called the *Design Space*. The space of the perceived fun can be represented by $Y = \{y | y \in R, 1 \le y \le 5\}$, briefly called *Fun Space*.

3. Evaluation of perceived fun for design alternatives

To find an optimal design of interest, a general mapping function between *Design Space S* and *Fun Space Y* has to be established. Such a mapping function is constructed by a neural network.

The architecture of the neural network is described below. The network is composed of three layers: an input layer, an intermediate hidden layer, and an output layer. Each node in the input layer represents the implementation level of a design feature and the single node in the output layer represents the overall perceived fun. Given a design alternative $\bar{x}$ in *Design Space S*, the neural network is intended to compute the overall perceived fun of the alternative.

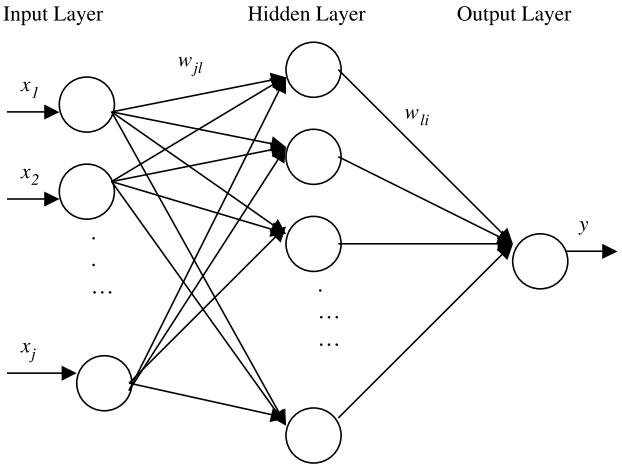

Fig. 1. Architecture of the neural network.

The construction of the neural network involves two stages: network training and verification. In Fig. 1, each of the lines connecting the nodes in distinct layers represents a weighting to be determined by the network training process. Of the sampled data sets, some are used to train the neural network and the others are used to verify the effectiveness of the network. A well-trained network has to ensure the validity of input/output mapping. That is, for an input vector $\bar{x}_h$ with a corresponding output y_h, the network will estimate an output O_h with tolerable error; namely, $(|O_h - y_h|/y_h) \le \varepsilon$ where ε is a very small value. The network if well trained can appropriately evaluate the perceived fun of a design alternative. Detailed procedure of the training algorithms can be referred to Rumelhart et al. (1986).

With the constructed network, we can quickly determine the perceived fun of any design alternative in *Design Space S*. To facilitate the following presentation, the mapping provided by the neural network is represented by $y = G(\bar{x})$.

4. Problem formulation

The research problem for selecting an optimal design alternative of a computer game can be formulated as follows:

$$\text{Minimize} \quad T(\bar{x}) = \sum_{j=1}^{N} f(x_j)$$

Subject to:

$$\bar{x} \in S \tag{1}$$

$$y = G(\bar{x}) \tag{2}$$

$$y \ge \mathrm{PF}_{\min} \tag{3}$$

The objective function indicates that the total design time is to be minimized, where $f(x_j)$ represents the design time required to achieve a particular implementation level x_j for jth design feature. Constraint (1) indicates that we need to

select a design alternative from *Design Space S*. Constraint (2) represents the neural network mapping between *Design Space S* and *Fun Space F*. Constraint (3) denotes that the perceived fun of the selected design alternative should meet the minimum requirement PF_{min}, which is demanded by users.

In the above formulation, *Design space S* consists of m^N design alternatives, where N denotes the number of design features and m denotes the number of implementation levels. The design space can easily become quite huge ($6^{39} \approx 2.23 \times 10^{30}$) when several tens of design features ($N=39$) and a few implementation levels ($m=6$) are involved. Applying an exhaustive search method is almost impossible. Therefore, this research develops a genetic algorithm to efficiently find a near-optimal solution from such a huge space.

5. Searching for optimal design alternative

We develop a GA to solve the formulated problem. Each aspect of the proposed GA is described below.

5.1. Chromosome representation

A solution (a design alternative) is modeled by vector $\bar{x}$ (called a chromosome), which is composed of N features (called genes). Each feature has m implementation levels.

5.2. Initial population

An initial population $P(0)$ is created by randomly generating M chromosomes. In generating a chromosome, the value of each gene x_j, an integer, is randomly chosen from the interval [0, 5].

5.3. Fitness function

By referring to Chen and Huang (2003) and Marcelin (2004) , the fitness function is defined below. The higher the $F(\bar{x})$, the lower is the fitness value of $\bar{x}$. The chromosome with a small fitness value is less likely to survive during the evolution of the population and tends to be finally excluded from the population.

$$F(\bar{x}) = -\left[T(\bar{x}) + Z\left(\frac{PF_{min}}{y} - 1\right)\right]$$

$$Z = \begin{cases} 0, & \text{if } PFmin \leq \bar{y}; \\ \text{a large positive number}, & \text{otherwise} \end{cases} \quad (4)$$

5.4. Crossover

The crossover operator is designed to create $M \times P_{cr}$ new chromosomes, where P_{cr} is the crossover probability. The creation procedure is as follows. Randomly sample $M \times P_{cr}$ chromosomes from $P(t)$ and group them into $(M \times P_{cr})/2$ pairs. Two new chromosomes are created by exchanging a portion of the two original chromosomes in a particular pair.

5.5. Mutation

The mutation operator is designed to create $M \times P_{mu}$ new chromosomes from $P(t)$, where P_{mu} is the mutation probability. This operator creates a new chromosome by randomly selecting a chromosome from $P(t)$ and replaces the value of one of its genes. The mutation procedure is repeated until $M \times P_{mu}$ has been created.

5.6. Evolution of population

Put the original chromosomes in $P(t)$ as well as the newly created ones into a set W, from which M chromosomes are to be selected to from the next population $P(t+1)$. The evolution procedure is as follows (Goldberg & Deb, 1991). Randomly sample $n \geq 2$ chromosomes from W and evaluate their fitness values. Select the one with maximum fitness value and put it into $P(t+1)$. Iterate the above steps until $P(t+1)$ is formed.

Table 1
Design features identified in example (Hsu et al., 2004)

DF1: Scenario is unpredictable
DF2: Scenario is varying
DF3: Scenario is dramatic
DF4: Scene is creative
DF5: Scene is looking-real
DF6: Scene is colorful
DF7: Scene is complex
DF8: Scene is varying
DF9: Scene transmits smoothly
DF10: Character is creative
DF11: Character looks like a real person
DF12: Character's style is similar to mine
DF13: More than one player can participate
DF14: Opponent is competitive
DF15: Opponent is unpredictable
DF16: Weapons are creative
DF17: Weapons are powerful
DF18: Background music suits the scene
DF19: Background music is varying
DF20: Music tempo suits the plot
DF21: Sound effect varies with events
DF22: Sound effect suits the event
DF23: Sound effect is loud enough
DF24: Sound effect is varying
DF25: Level difficulty is flexible to choose
DF26: New skills are acquired at every level
DF27: Level difficulty increases progressively
DF28: Levels can be skipped
DF29: Beginning levels are easy
DF30: Final levels are difficult
DF31: Pace is fast enough
DF32: Pace is varying
DF33: Input device is easy to control
DF34: Game can be saved for continuity
DF35: High score board can be viewed
DF36: Game scores can be accumulated
DF37: Virtual token can be won
DF38: Instruction is clear
DF39: Real-time information is updated

5.7. Termination condition

The evolution stops either when the number of evolution $t > T$ or when a particular chromosome keeps being the best solution in the population for over Q generations.

6. Numerical example

The proposed method of integrating ANN and GA to optimally design an action game is implemented by using the result of Hsu et al. (2004). The 39 design features identified are shown in Table 1. The implementation is coded with C++ program language.

In the research problem, we aim to identify a design alternative that minimizes total design time $T(\bar{x})$ and meets the minimum fun requirement $PF_{min} = 4.5$. Design times required to achieve various implementation levels for each design feature are shown in Table 2.

Table 2
Design time required to achieve various implementation levels for each design feature

Implementation level	0	1	2	3	4	5
DF1	0	10	30	50	70	90
DF2	0	10	20	40	80	160
DF3	0	10	30	90	190	330
DF4	0	10	20	40	80	160
DF5	0	10	20	30	40	50
DF6	0	5	6	10	12	15
DF7	0	5	7	8	9	10
DF8	0	5	10	15	18	20
DF9	0	5	7	10	12	20
DF10	0	10	20	30	40	50
DF11	0	10	30	60	90	120
DF12	0	10	20	40	80	160
DF13	0	10	12	20	22	24
DF14	0	10	20	40	80	160
DF15	0	10	20	30	40	50
DF16	0	10	20	40	80	120
DF17	0	10	20	40	80	100
DF18	0	10	20	40	60	80
DF19	0	5	7	19	20	25
DF20	0	5	7	8	9	10
DF21	0	5	7	10	15	20
DF22	0	10	20	30	40	50
DF23	0	5	7	10	12	14
DF24	0	5	8	10	12	20
DF25	0	5	6	10	12	15
DF26	0	10	20	40	60	80
DF27	0	10	20	30	40	50
DF28	0	10	12	15	18	20
DF29	0	5	8	10	15	20
DF30	0	5	8	10	12	15
DF31	0	10	12	15	18	20
DF32	0	10	12	15	18	20
DF33	0	10	20	30	40	50
DF34	0	5	15	25	35	45
DF35	0	5	10	15	20	25
DF36	0	5	10	15	20	25
DF37	0	10	20	40	60	80
DF38	0	5	10	15	20	25
DF39	0	10	20	30	40	50

Table 3
The best design alternative of the 100 replications

Total design time required $T(\bar{x})$	483				
Overall perceived fun (y)	4.5483				
Implementation level of each design feature	$x_1=0$	$x_2=1$	$x_3=1$	$x_4=2$	$x_5=2$
	$x_6=3$	$x_7=2$	$x_8=0$	$x_9=5$	$x_{10}=0$
	$x_{11}=1$	$x_{12}=0$	$x_{13}=0$	$x_{14}=0$	$x_{15}=1$
	$x_{16}=0$	$x_{17}=2$	$x_{18}=4$	$x_{19}=4$	$x_{20}=0$
	$x_{21}=4$	$x_{22}=0$	$x_{23}=5$	$x_{24}=3$	$x_{25}=4$
	$x_{26}=2$	$x_{27}=1$	$x_{28}=5$	$x_{29}=0$	$x_{30}=5$
	$x_{31}=5$	$x_{32}=1$	$x_{33}=1$	$x_{34}=5$	$x_{35}=3$
	$x_{36}=1$	$x_{37}=0$	$x_{38}=3$	$x_{39}=3$	

Of the 28 design alternatives in Hsu et al. (2004) study , the data sets of 21 alternatives are used to train a neural network and that of the other seven alternatives are used to evaluate the effectiveness of the neural network. The trained neural network seems an effective one because the output error of the network is less than 0.2%.

The proposed GA is then used to identify a near-optimal design alternative from *Design Space S*. The parameters of the GA are given as follows: $M = 100$, $P_{cr} = 0.80$, $P_{mu} = 0.05$, $T = 99{,}999$, $Q = 1000$. We run the GA program with 100 replications. For the 100 replications, the mean design time is 514.83 time units and its standard deviation is 19.396 time units. Of the 100 replications, the best design alternative is shown in Table 3. The small standard deviation indicates that a few replications will be sufficient to identify a near-the-best solution of the GA. The computation time required for a replication is about 5 min, by a personal computer with 1.8 GHz CPU. This approach can provide a timely solution and therefore serve as an effective aid for game design.

7. Conclusions

This study proposes an approach to solving a trade-off decision problem for game design. Making a trade-off among design features is very important in the design process, but can be very complex. A computer game involves a large number of design features, and each feature can be implemented in various levels. The number of design alternatives can therefore be quite huge due to the combinatorial explosion of implementation levels. Due to the pressure of time-to-market, the time available to designers is relatively limited. Therefore, developing a method that allows to efficiently identifying a design alternative is critical to game design.

This study uses ANN and GA to optimize the design of a computer game. This issue is resolved in two-fold. First, the constructed ANN, using a few expensive experiment data, can evaluate the overall perceived fun of a design alternative. Second, the proposed GA can efficiently identify a near-optimal design alternative out of the enormous solution space. The suggested design alternative facilitates game designers to properly allocate their efforts to various design features.

Although this approach has been successfully applied to action games, the applicability of other types of digital contents remains to be investigated. At the present time, we are

investigating the possibility of extending this approach to some other digital contents such as other types of computer games, multi-media, movies, e-learning, and man–machine interfaces of mobile devices.

References

Bethke, E. (2003). *Game development and production*. Plano, TX: Wordware Publishing.

Chen, M. C., & Huang, S. H. (2003). Credit scoring and rejected instances reassigning through evolutionary computation techniques. *Expert Systems with Applications*, *24*(4), 433–441.

Fabricatore, C., Nussbaum, M., & Rosas, R. (2002). Playability in action videogames: A qualitative design model. *Human–Computer Interaction*, *17*, 311–368.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 69–93). San Mateo, CA: Morgan Kaufmann.

Hsu, S. H., Lee, F. L., & Wu, M. C. (2004). *Design for appealing to game buyers*. Technical report, IEM, NCTU, Taiwan.

Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, *19*(2), 125–132.

Kim, K. J., & Han, I. (2003). Application of a hybrid genetic algorithm and neural network approach in activity-based costing. *Expert Systems with Applications*, *24*(1), 73–77.

Kuo, R. J., & Chen, J. A. (2004). A decision support system for order selection in electronic commerce based on fuzzy neural network supported by real-coded genetic algorithm. *Expert Systems with Applications*, *26*(2), 141–154.

Lee, S., Lee, H. S., Lee, K. C., Choi, J. W., & Lee, M. H. (2001). Performance management of communication networks for computer integrated manufacturing. *International Journal of Advanced Manufacturing Technology*, *18*(10), 764–770.

Li, S., Wu, Z., & Pang, X. (2004). Job shop scheduling in real-time cases. *International Journal of Advanced Manufacturing Technology* . doi10.1007/s00170-003-2051-x.

Malone, T. W., & Lepper, M. R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. In R. E. Snow, & M. J. Farr (Eds.), *Aptitude, learning, and instruction, III: Conative and affective process analysis* (pp. 223–253). Hillsale, NJ: Lawrence Erlbaum.

Marcelin, J. L. (2004). A metamodel using neural networks and genetic algorithms for an integrated optimal design of mechanisms. *International Journal of Advanced Manufacturing Technology*, *24*(9–10), 708–714.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs*. London: Springer.

Mok, S. L., Kwong, C. K., & Lau, W. S. (2001). A hybrid neural network and genetic algorithm approach to the determination of initial process parameters for injection moulding. *International Journal of Advanced Manufacturing Technology*, *18*(6), 404–409.

Qiu, H. B., & Li, C. X. (2004). Conceptual design support system in a collaborative environment for injection moulding. *International Journal of Advanced Manufacturing Technology*, *24*(1–2), 9–15.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representation by error propagation. *Parallel and Distributed Processing*, *1*, 318–362.

Shi, F., Lou, Z. L., Lu, J. G., & Zhang, Y. Q. (2003). Optimisation of plastic injection moulding process with soft computing. *International Journal of Advanced Manufacturing Technology*, *21*(9), 656–661.

Shin, T., & Han, I. (2000). Optimal signal multi-resolution by genetic algorithms to support artificial neural networks for exchange-rate forecasting. *Expert Systems with Applications*, *18*(4), 257–269.

Su, J. C., Kao, J. Y., & Tarng, Y. S. (2004). Optimization of the electrical discharge machining process using a GA-based neural network. *International Journal of Advanced Manufacturing Technology*, *24*(1–2), 81–90.

Versace, M., Bhatt, R., Hinds, O., & Shiffer, M. (2004). Predicting the exchange traded fund DIA with a combination of genetic algorithms and neural networks. *Expert Systems with Applications*, *27*(3), 417–425.

Yuan, S. T., & Chen, S. F. (2001). A learning-enabled infrastructure for electronic contracting agents. *Expert Systems with Applications*, *21*(4), 239–256.