Decision Making Based on Feed-Forward Neural Networks for Time-Constrained Sequential Decision Problems: Application to Tetris

Go Irie
School of Science for OPEN and
Environmental Systems
Keio University
3-14-1 Hiyoshi, Kohokuku
Yokohama 223-8522, Japan
email:irie@yoshida.sd.keio.ac.jp

Kazuo Yoshida
Department of System Design Engeneering
Keio University
3-14-1 Hiyoshi, Kohokuku
Yokohama 223-8522, Japan
email:yoshida@sd.keio.ac.jp

***Abstract*—In this paper, an approach using feed-forward neural networks for solving decision problems with time constraints is proposed. Especially, the approach focuses attention on easy construction and increase of efficiency of search by using two different decision mechanisms and a control mechanism designed with feed-forward neural networks which is able to construct nonlinear and multi-output function. To verify the validity of the technique, it is applied to a game playing program for Tetris which is a kind of puzzle game with strict time constraint. The technique copes with time constrained decision problem by considerably automatic construction.**

I. Introduction

Recently, many researches have been made about algorithms for intelligent systems performing autonomous problem-solving and decision-making. In order to apply them to real problems, it is necessary to consider some constraints in the task environment where the system is placed, such as observation impossibility, uncertainty and time constraints. Especially about time constraints, there may be nothing that does not contain it in real problems, and it is recognized as the important one which should be taken into consideration.

Many techniques for solving time-constrained problems have been proposed. Dean and Boddy advocated the framework of *anytime algorithm* [1] which is able to interrupt processing in arbitrary time such as Dynamic Programming (DP). They also proposed *deliberation scheduling* [2]: scheduling technique using *anytime algorithm*. Then, Zilberstein *et al.* have progressed *contract algorithm* as one kind of *anytime algorithm* which determinates computation time in advance in accordance with predicted deadline [3], [4]. Although it is thought that flexibility is spoiled a little, it becomes unnecessary to carry out continuous monitoring for determination of interrupting time, it may be interpreted as the form of considering the practical use side. Several techniques in its framework have also been proposed, for example in [5], a type of contract scheduling technique which uses the different solution methods depending on the situation has been proposed. As an another approach, there is *metareasoning* [6] which solves it by including terms representing time costs in a utility function.

The effectiveness of these techniques depends largely on the skill of designers for the point that designers have to specify the type of problems including time constraints, and design evaluation functions or utility functions united with it in design processes. Usually these processes become a cause of trouble to designers. Therefore, techniques which give efficient solutions by easy and inexpensive design processes should be explored.

On the other hand, it has been verified that neural networks provide easy solutions with automated construction for many difficult problems in the various fields such as control engineering or robotics. For time-constrained scheduling problems, a technique using Hopfield neural networks as an approximate method for optimization has been proposed in [7]. Among various types of neural networks, the feed-forward neural networks have the several distinguishing abilities such as nonlinear mapping, learning, and generalization; there have been especially many applications. Also for solving the problem described above, these abilities may provide good advantages. Therefore, it is significant to explore feed-forward neural network-based approaches for time-constrained problems.

The purpose of this paper is a proposal of an efficient technique which can be designed automatically for solving time-constrained decision problems. The technique is a contract type algorithm, and solves time-constrained decision problems by simultaneous use of two types of decision-making mechanisms and a controller designed with feed-forward neural networks. Each component has a special role independently, and it means a separation of roles aiming at easy construction and high efficiency. The basic procedure of the technique is search and run-time allocation. About run-time allocation, it is performed by the controller which is the unique component for considering time constraints, depending on current state of task environment, to only a task which occurs in the nearest future. The type of problem treated in this paper is a sequential decision problem. Although the typical benchmark of this type of problems is grid world, Tetris is adopted—a larger scale decision problem which is including time constraints more explicitly.

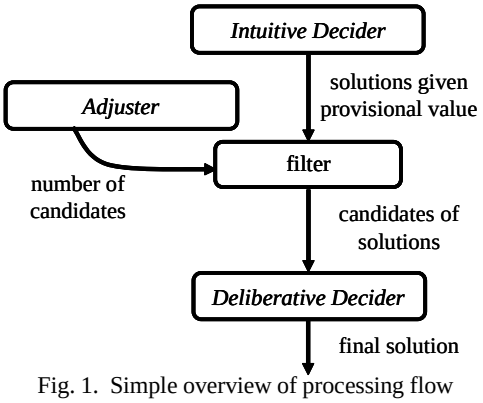

Fig. 1. Simple overview of processing flow

II. Proposal Technique

A.Outline

The technique is constructed on the assumption that a deadline of a nearest future task is predictable before starting processing. In view of this, it lies in the framework of *contract algorithm*. The basic architecture of the algorithm proposed in this paper is based on [8]. It consists of three main components including feed-forward neural networks: two decision-making mechanisms which have mutually different performances, and a controller which takes a balance. The difference of performances between two decision-making mechanisms is in quality of results and computation time. One of the two performs processing emphasizing quality of results than computation time, and the other does computation time than quality of results. As a matter of convenience, the former is called *Deliberative Decider* and the latter is called *Intuitive Decider*. Each decision-making mechanism can make its decision independently. However, that is difficult to solve time-constrained problems. Therefore, these are used together through a controller called *Adjuster* which performs run-time allocation.

Simple overview of the processing flow is shown in Fig. 1. First, *Intuitive Decider* gives provisional value for all solutions. Second, *Adjuster* receives some features which control the level of time constraints as inputs, and predicts deadline implicitly. The concrete method of the prediction is to determine the number of candidates which is the number of solutions going to be reevaluated by *Deliberative Decider* whose computation time increases in proportion to the number of solutions to evaluate. Third, only the number determined by *Adjuster* is selected from in high quality order from all the solutions given provisional value by *Intuitive Decider*. Finally, *Deliberative Decider* reevaluates the candidates which are selected, and outputs the final solution which obtained the highest value as the decision.

In design process, what designers have to do for constructing the system are to determine models of feed-forward neural networks of each component, and training methods of them.

B.Deliberative Decider

The most important purpose of *Deliberative Decider* is to select a solution with highest quality of results precisely from given candidates. For its achievement, *Deliberative Decider* evaluates each candidate individually, and then selects the final solution. The processing flow of *Deliberative Decider* can be summarized as follows.

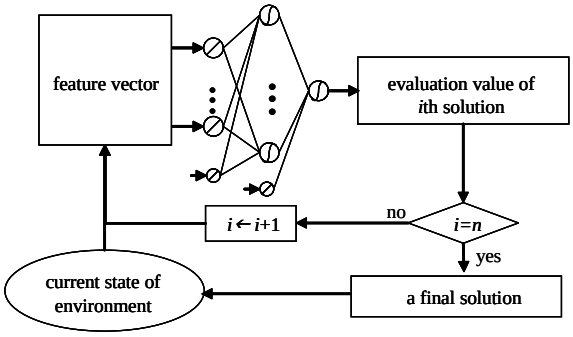

Fig. 2. *Deliberative Decider*

Step 1. Observe current state of task environment.
Step 2. Set the number of candidates to *n*.
Step 3. Set *i*=1.
Step 4. About the *i*th candidate, compute features used as inputs.
Step 5. Evaluate the *i*th candidate using the computed features with an evaluation function constructed as a feed-forward neural network.
Step 6. If *i*=*n*, go to Step 7. Otherwise do *i*f *i*+1, then return to Step 4.
Step 7. Select the solution which has the highest evaluation among *n* candidates, and output it as a final solution.

The difficulty of designing an evaluation function is to select the model and to adjust the parameters. The reason for using a feed-forward neural network as an evaluation function is that the process is automated by using feed-forward neural networks which have advantage of high performance of learning and nonlinear characteristics. In the learning process, to avoid the difficulty of collecting training data sets and to acquire high automaticity, it is necessary to use methods which do not require training data sets, such as Evolutionary Computation (EC).

C.Intuitive Decider

The most important purpose of *Intuitive Decider* is quick processing, in other words, evaluating all solutions as fast as possible without losing the required minimum performance for estimating quality of solutions. Unlike *Deliberative Decider*, each of solutions is not evaluated individually, but all of them are evaluated at a time by using multi-output feed-forward neural networks. For this reason, different from *Deliberative Decider*, *Intuitive Decider* cannot use the features which correspond to variations of state. *Intuitive Decider* performs pattern recognition for selecting a solution from the features representing current state. The processing flow can be summarized as follows.

Step 1. Observe current state of task environment.

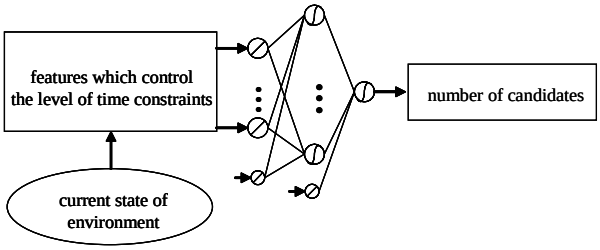

Fig. 3. *Adjuster*

Step 2. Compute features used as inputs from the observed state.

Step 3. Evaluate all solutions with an evaluation function constructed as a multi-output feed-forward neural network.

Step 4. Select the solution which has the highest value among them, and output it as a final solution.

Feed-forward neural network is used by the same reason as *Deliberative Decider*, in addition, it is also a reason that multi-output functions can be obtained easily. Learning is carried out by Back-Propagation (BP) algorithm. In this process, the current decision of *Deliberative Decider* is treated as the desired output. Because *Deliberative Decider* constructed in advance provides desired outputs, learning with BP is performed automatically as if it is an unsupervised learning.

D.Method of simultaneous use

Using two decision-making mechanisms independently does not lead to a good trade-off. Therefore, they are used together in the technique. The method of simultaneous use is that *Intuitive Decider* evaluates all solutions previously, and only some prospective solutions which obtain good evaluation are reevaluated by *Deliberative Decider*. *Intuitive Decider* does not draw final conclusions, but gives provisional value to narrow solutions, and final decision is made by *Deliberative Decider*. This method enables the algorithm to increase efficiency of search, since it becomes unnecessary to apply *Deliberative Decider* which requires much time for evaluating all solutions including hopeless ones. When the number of solutions which *Deliberative Decider* reevaluates increases, the more priority is indeed given to the quality of results and when it decreases, the more priority is given to processing time. In this method, the only factor that controls trade-off is the number of candidates narrowed down, and in order to control it automatically depending on the degrees of time constraints, we adopt *Adjuster*.

E.Adjuster

Adjuster performs run-time allocation according to the level of time constraints by setting the number of candidates, and plays a role which determines the balance competing characters for quality of results and computation time. It enables the proposal technique to respond time constraints.

Adjuster receives some features which dominate the degrees of time constraints as inputs, and outputs the number of solutions reevaluated by *Deliberative Decider*. It is assumed that the features used as inputs are observable. The function is obtained by a feed-forward neural network, and in learning, for the same reason in *Deliberative Decider*, it is necessary to use methods which are not required training data sets, such as EC.

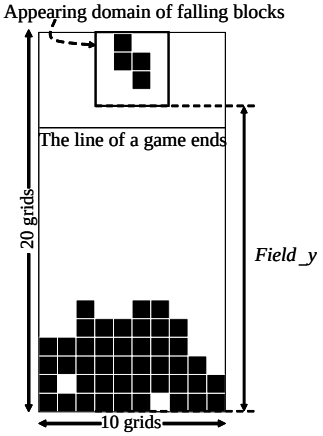

Fig. 4. Game screen of Tetris

III. Tetris

Tetris is a kind of the puzzle game developed by Alexey Pajitnov *et al.* in 1985, and has been studied from various perspectives. For example, in [9], it was treated as a large-scale Markov decision problem, and was played by DP using some features. In this paper, although the general rule is used in most part, some specific points are shown below.

a)The game field consists of grid of height 20 and width 10.

b)In order to adjust velocity of falling blocks to computer spec (733-MHz Pentium III), it is set quite quicker than when man plays as a game, specifically in the range of 1.0 - 40.0 grid/msec.

c)When 16th row from the bottom of the field is crossed with the height of the wall, a game ends.

d)Next falling blocks are unobservable.

e)Result is not any score, but total deleted lines.

In the simulation, the deadline is approximately calculated with the following equation:

$$Free\_y = Field\_y - \overline{Height} \quad (1)$$

$$Time_l = Free\_y / Fall\_Velocity \quad (2)$$

where, *Field_y* is the maximum range of grids for controlling falling blocks. In the simulation, *Field_y*=16. The reason is that the height of the field and the appearing domain of falling blocks are 20 and 4 respectively, so, it is calculated as 20 - 4 = 16 (showed in Fig. 4). $\overline{Height}$ is the average height of columns in field. *Free_y* is the range of grids which are able to control falling blocks. *Fall_velocity* is the velocity of current falling blocks. $Time_l$ is time to deadline, in other words, time from appearance of current falling block to placement on the wall. Success and failure of a decision branches off in two results by comparing calculated time to deadline $Time_l$ with real processing time $Time_p$.

a) $Time_p \leq Time_l$

The decision is success, and the action according to it is carried out.

b)*otherwise*

The decision is failure, and the action which is randomly shortened is carried out. For example, if the series of operations of the action according to the decision are [move right, move right, move right, rotate, drop], then [move right, move right, move right, drop], or [move right, move right, drop] may replace them.

Tetris has the following features as a benchmark of decision problems.

a)The current state of the field is dependent on the history of actions taken until then.

b)The task is deletion of lines, and for the execution of it, a series of action sequences of some size is required.

c)Next falling block is determined at random, therefore it cannot be known in advance.

d)It is NP-complete that optimizing various objectives in the offline version of Tetris such as maximizing the number of deleted lines [10].

e)The velocity of current falling block and the current state of walls control time for current decision.

e) is the feature which is closely related to the problem set on this paper, and should be explained in detail. In Tetris we play, the velocity of falling blocks is not always constant, but increasing with the gradual growth of total deleted lines. It is greatly related to the deadline of time for decision, and as the velocity of fall blocks becomes quicker, the time for decision becomes shorter. In addition, the more there are many blocks piled on field, the domain which can handle falling blocks freely is reduced, and it also causes reduction of time until deadline. That is, the deadline is changed depending on both current state of the wall and the velocity of falling blocks. If a processing is not in time, even if it selects an excellent decision which can get 4 deleted lines, the action must become impossible to execute and the value of the decision decreases remarkably. It is exactly the problem of the trade-off between quality of results and computation time in time-constrained problems, therefore Tetris is a good benchmark of that problems.

IV. Construction of Game Playing Program

In the technique, it is three main components that have to be constructed, but most parts of construction processes are performed by the automated learning. What we have to do is to determine the models of feed-forward neural networks of them, and the methods of training. The process is described in further detail below.

A.Deliberative Decider

The feed-forward neural network in *Deliberative Decider* comprise 14 input nodes, 23 hidden nodes, and an output node (including a bias node in each of input layer and hidden layer). Input nodes use a linear activation function, and hidden and output nodes use a sigmoid activation function given by the following equation

$$f(x) = \frac{1}{1+\exp(-x)}. \quad (3)$$

The bias nodes use a linear activation function. The features chosen for inputs are prepared in Table 1 (some variations of states are treated as features). In fact, these features are inputted after carrying out scaling adjustment suitably.

Table 1. Features chosen as inputs of the feed-forward neural network of *Deliberative Decider*

1) $Height_{max}$	maximum height among columns
2) $Space$	the number of spaces on the wall
3) $Height_{sum}$	summation of height
4) $Hole_{sum}$	summation of depth of holes on the wall
5) $DeleteLine$	deleted lines which obtained when a decision is carried out
6) $\Delta Height_{max}$	difference of maximum height when a decision is carried out
7) $\Delta Space$	difference of the number of spaces when a decision is carried out
8) $\Delta Height_{sum}$	difference of summation of height when a decision is carried out
9) $\Delta Hole_{sum}$	difference of summation of depth of holes when a decision is carried out
10) $JointSurface$	the number of surfaces where a falling block touches the wall when a decision is carried out
11) $Height_{drop}$	height where a falling block dropped when a decision is carried out
12) P_1 13) P_2	difference of summation of height in a special domain when a decision is carried out

Table 2. Operators

	operators	
selection	MGG [11]	
crossover/ production	for weights	LUNDX-m(m=15)+EDX [12] and mutation (perturbation according to normal distribution)
	for feature selection	IUMDA [13]

In the learning, it is used that EC with the offline version of Tetris. The average of deleted lines of five games is given as fitness, and process of learning is iterated for 1000 generations. In this regard, since all the features shown above are not necessary, coincidentally they are selected in the process of learning. In coding, the weights are encoded with real number, and with regard to feature selection, the coding is performed by binary coding, 1 corresponds to the case where the feature is used, and 0 corresponds to the case where it is not used. As the operators, by considering that it is the mixed problem of real number and binary number, what maintain diversity well and have high performance to search solution are selected (Table 2). The sampling for update of probability vector in IUMDA is performed by roulette wheel selection with the following equation:

$$fitness_i' = fitness_i / c_i \quad (4)$$

$$E_i = fitness_i' \Big/ \sum_i^{pop} fitness_i' \quad (5)$$

(it is simplified version of equation for calculation of expectation value used in immune algorithm (IA) [14]) where, E_i is the parameter of probability for survival of *i*th individual, $fitness_i$ is the fitness value of *i*th individual, *pop* is the number of individuals, c_i is the parameter about similarity of *i*th individual (about c_i, it is described in [14] in detail). Consequently, the selected features were seven: 2), 6), 7), 8), 9), 10), and 11) in Table1.

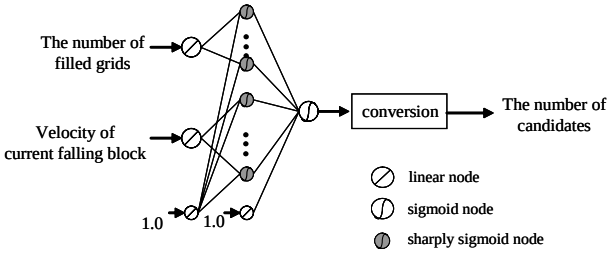

Fig. 5. The model of feed-forward neural network of *Adjuster*

B.Intuitive Decider

In *Intuitive Decider*, the feed-forward neural networks are separately prepared to 19 kinds of each falling block. The reasons are to reduce burden of each feed-forward neural network, and to solve the problem that the number of available solutions changes from 9 to 34 according to the kind of falling blocks. Each of networks comprises 41 input nodes, 35 hidden nodes, and from 9 to 34 output nodes (including a bias node in each of input nodes and hidden nodes). Input nodes use a linear activation function, hidden and output nodes use a sigmoid activation function given by Eq. (3). Input vector is formed by the information on grids of 4×10 which is in upwards from the row of the minimum height at the range of four lines. The components in the input vector are elements from {1, 0}, where 1 corresponds to a filled grid, 0 is an empty grid. The learning is executed by BP with the offline version of Tetris. The desired output vector comprises only an element corresponding to the decision of *Deliberative Decider* as 1.0, and the other element as 0.0.

C.Adjuster

The model of feed-forward neural network is shown in Fig. 5. The feature of this model is to implement pruning partially between input layer and hidden layer for two purposes. One is reduction of the processing time, the other is to guarantee nonlinearity and monotonically decreasing by constraining the weights of input-hidden layers at 1.0, hidden-output layers at negative value and the biases at positive value. The input layer consisted of 2 linear nodes (and a bias node), the number of the grids currently filed all over the field and the velocity of current falling block are inputted as the factor which governs time constraint (these are inputted after carrying out linear scaling adjustment suitably in fact). In the activation function of hidden layer, to attain sharp change of the output value easily, it was adopted that represented by the following equation

$$f(x) = \frac{1}{1 + \exp(-100 \cdot x)}. \quad (6)$$

The number of nodes in hidden layer was 15 (except a bias node): 10 of them are connected to "The number of filled grid" input node, and the other 5 are connected to "Velocity of current falling block" input node. Output node use a sigmoid activation function given as (3), to output the number of candidates which is reevaluated in *Deliberative Decider*, the conversion is performed by the following equation:

$$CandidateDecisionNum = ceil(34.0 \times y) \quad (7)$$

where, y is the output of the feed-forward neural network,

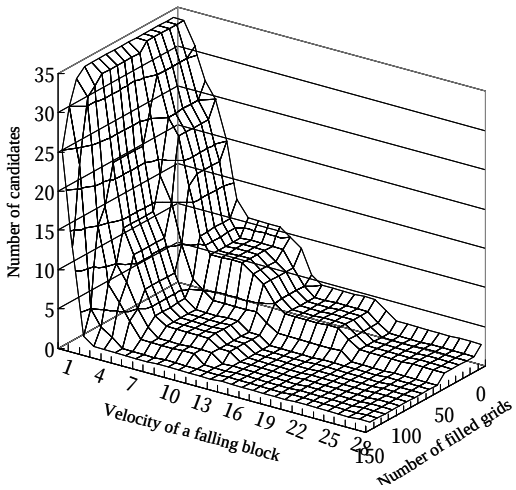

Fig. 6. Relation between the number of candidates and the number of filled grids or velocity of a falling block

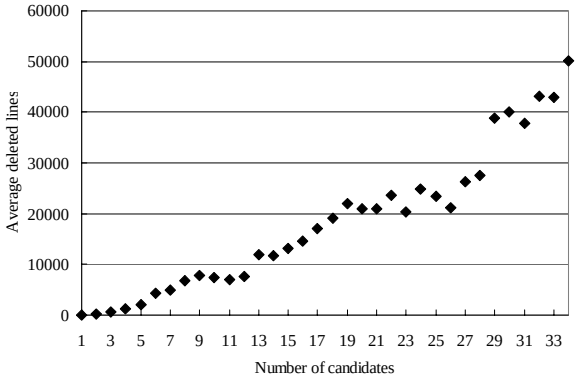

Fig. 7. Performance profile

$CandidateDecisionNum$ is the number of candidates, $ceil(x)$ is the function which returns more than x minimum integer values. Learning is performed using EC with regarding average deleted lines of 5 games play as fitness, in the environment that velocity of falling block started from 1.0 grid/msec. and it increases by 1.0 grid/msec. whenever 5 deleted lines are obtained. The process of learning is iterated for 300 generations. MGG as the selection operator, LUNDX-m(m=20)+EDX as the crossover operator, and as the mutation operator, the completely same thing as the case of *Deliberative Decider* is used in this process.

V. Simulation Results

Before the simulation result with Tetris, the input-output relation of *Adjuster*, in other words, the relation between the number of candidates which are reevaluated and the number of grids or velocity of a falling block is shown in Fig. 6. Since time-constraints become severe along with the number of the grids filled in the field and velocity of a falling block increase, the number of solutions reevaluated is decreased, and vice versa.

Next, the simulation result using Tetris is shown. First, it is shown in Fig. 7 that the difference of the results of plays when the number of solutions reevaluated was changed. This figure has the meaning of the *performance profile* of *anytime algorithm* [1], [2]. The curve is monotonically increasing in most part, although there seems to be slight irregular part because of randomness included in Tetris, and it

Table 3. Simulation Result

Velocity	Deliberative Decider		Intuitive Decider		Proposal method	
[grid/msec.]	ave.	max.	ave.	max.	ave.	max.
1.0	37950.98	218711	30.16	69	40183.38	265404
3.0	7419.50	29471	33.44	130	23507.72	91167
5.0	267.64	1120	27.78	94	10570.40	32545
7.0	1.84	5	29.30	83	2966.88	11644
10.0	0.62	3	30.00	139	1072.08	6993
15.0	0.02	1	22.34	76	204.14	1020
20.0	0.00	0	12.18	67	53.26	215
30.0	0.00	0	2.38	8	2.56	11
40.0	0.00	0	1.62	8	1.54	8
grad.	41.26	46	27.20	122	145.22	231

is the character of *performance profile* of *anytime algorithm*. Second, the simulation result in Tetris in which time constraints are included is shown. In Table 3 the results of the number of deleted lines when playing 50 games under each falling block speed is shown (grad. represents the case where it starts from 1.0 grid/msec., and it increases by 1.0 grid/msec. whenever 5 deleted lines are obtained). This result shows that the technique has found out a good trade-off which may be associated with the level of time constraints. Third, Figure 8 describes an example of transition of the number of candidates during a game play is shown. The number of candidates is decreased in synchronization with gradual increment of the velocity of current falling block, and is changed accurately by fluctuation of the number of filled grids.

VI. Conclusion

In this paper, time constrained sequential decision problems were considered, and a technique to solve it was proposed. The technique was based on feed-forward neural networks for considering user-friendliness of construction and efficiency of search. To verify that the technique can solve these problems actually, it is applied to a game playing program for Tetris.

The technique was applied to the game playing program for Tetris including time constraints explicitly. In construction process, the abilities of feed-forward neural networks and the structure of the model proposed were harnessed, and most part of the process were performed automatically. It is considered as an advantageous point for easy use because it lessens designers' burden.

In the simulation with Tetris, the following results were obtained. First, the proposal technique which uses them together through Adjuster showed quite good results compared with the cases where two decision-making mechanisms are used independently. Second, it turns out that the number of candidates was greatly fluctuated through a game according to the change of the features which govern time constraints. From these results, it was shown that present technique is able to find out good trade-off in time-constrained decision problems, and to solve these problems efficiently.

It is just conceivable that the single task run-time allocation worked well, because the type of time constraint of Tetris was simple. To apply the technique to another type of problems which are more complex, some types of extensions may be needed such as preparing hierarchical model for larger scale scheduling or deepening the depth of search.

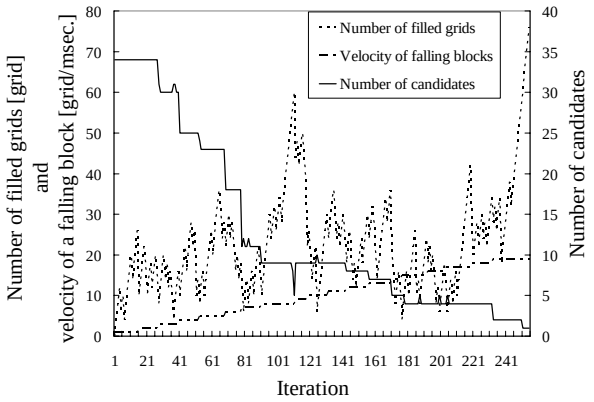

Fig. 8. An example of transition of the number of candidates through a game play

References

[1]M.Boddy, and T.L.Dean. An Analysis of Time-Dependent Planning, *Proc. AAAI-88*, pp.49-54, 1988.

[2]M.Boddy, and T.L.Dean. Deliberation Scheduling for Problem Solving in Time-Constrained Environments, *Artificial Intelligence*, 67, pp.245-285, 1994.

[3]S.Zilberstein, and S.J.Russell. Efficient Resource-Bounded Reasoning in AT-RALPH, Proc. *First International Conference on AIPS*, pp.260-266, 1992.

[4]S.Zilberstein *et al.* Real-Time Problem-Solving with Contract Algorithms, *Proc. IJCAI-99*, pp.1008-1013, 1999.

[5]A.J.Garvey, and V.R.Lesser. Design-to-Time Real-Time Scheduling, *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), pp.1491-1502, 1993.

[6]S.Russell, and E.Wefald. Principles of Metareasoning, *Artificial Intelligence*, 49, pp.361-395, 1991.

[7]F.Charpillet *et al.* Stochastic and Distributed Anytime Task Scheduling, *Proc. 10th International Conference on Tools with Artificial Intelligence*, pp.280-287, 1998.

[8]T.Hashieda, and K.Yoshida. Online learning system with logical and intuitive processings using fuzzy Q-learning and neural network, *Proc. IEEE International Symposium on CIRA*, 1, pp.13-18, 2003.

[9]J.N.Tsitsiklis, and B.V.Roy. Feature-Based Methods for Large Scale Dynamic Programming, *Machine Learning*, 22, pp.59-94, 1996.

[10]E.D.Demaine *et al.* Tetris is Hard, Even to Approximate, *Proc. COCOON-03*, pp.351-363, 2003.

[11]H.Sato *et al.* A New Generation Alternation Model of Genetic Algorithms and Its Assessment, *Journal of Japanese Society for Artificial Intelligence*, 12(5), pp.734-744, 1997.

[12]J.Sakuma, and S.Kobayashi. *k*-tablet Structures and Crossover on Latent Variables for Real-coded GA, *Proc. GECCO Late-breaking papers*, pp.404-411, 2002.

[13]H.Muhlenbein. The Equation for Response to Selection and Its Use for Prediction, *Evolutionary Computation*, 5(3), pp.303-346, 1998.

[14]K.Mori *et al.* Application of an Immune Algorithm to Multi-Optimization Problems, *Transactions of the Institute of Electrical Engeneeres of Japan part C*, pp.593-598, 1997.