

## MULTIPLE OBJECTIVE SOLUTIONS FOR TETRIS\*

MICHAEL M. KOSTREVA

REBECCA HARTMAN

*Department of Mathematical Sciences  
Clemson University  
Clemson, SC 29634-1907  
e-mail: FLSTGLA@CLEMSON.EDU*

### ABSTRACT

The electronic game *Tetris* is among the most popular games in history, with over 40 million copies sold. Multiple objective dynamic programming is extended to show how to obtain a Pareto optimal strategy within a discrete optimal control formulation. Four objective functions derived from players' experiences are scalarized for practical implementation.

### Introduction

The objective of this article is to develop a theory to explain the electronic game of *Tetris*. *Tetris* is an extremely popular "game" that is available on most types of personal computers, on Nintendo "Game Boy" devices, and even a Nintendo wristwatch version exists. The word "game" is not as accurate a description as one might expect. *Tetris* is more of a puzzle than a game because the electronic machine supporting the play is not actively replying to the human. For our

\*The authors wish to express their gratitude for support from the National NSF Multidisciplinary Design and Analysis Fellowship NGEP0008, NASA's Space Grant Program, and South Carolina PSCOR 1996 Undergraduate Research program.

purposes, we focus on the personal computer versions. Geometric shapes composed of four squares are presented to the human player by the electronic machine according to a uniform distribution. The human player controls these geometric shapes by giving input commands by touching keys with the fingers. We propose that the theory may ultimately lead to determining whether or not *Tetris* can be "won." That is, can *Tetris* be played indefinitely (continuously accumulating points), or does the theory require a disruption of play under certain conditions? The nature of the decision-making process of *Tetris* suggests a dynamic programming approach.

Speaking in general, the goals of the human *Tetris* player include maximization of the duration of play and the accumulation of points. The game begins with the appearance of one of the seven *Tetris* pieces at the top of a rectangular playing board. The board is a 20 by 10 matrix of square cells. The player then translates the piece either right or left one cell at a time, rotates the piece counterclockwise by 90 degrees, and/or drops the piece. Once the piece is at rest, a new piece begins its descent onto either the base or the previously placed piece, and so on. We call the shape of the accumulated pieces *the contour*. When a horizontal row of cells is completely filled by portions of various *Tetris* pieces, with no empty spaces, that row is deleted from the board. The filled cells above it drop one row onto those below this filled row, points are awarded and displayed, and play continues. The game is lost when the accumulated pieces reach the top of the board, thereby preventing further positioning.

As the game proceeds and points are accumulated, a threshold is triggered, the level changes, and the speed of the falling *Tetris* pieces increases. Our mathematical model of *Tetris* makes time discrete and does not account for these changes in speed. Although time is an important aspect of the actual game and may be the cause of a human player's loss, it is removed as a factor because it does not directly affect our development of the theory. We assume that the steps of our decision-making algorithm proceed at a speed sufficient to remain ahead of the falling *Tetris* pieces, regardless of the level changes.

Our research started with a thorough literature and Internet search. We found that the mathematical journals were devoid of contributions which discussed *Tetris*. We gathered what we could find apart from journals and then began to study the main factors of the play of *Tetris*. The first factor considered was the character and distribution of the falling *Tetris* pieces. Existing research projects in this area failed to explicitly consider the distribution and simply assumed "randomness" [1-3]. The data we collected was the total number of appearances of each of the seven *Tetris* pieces for a number of different plays of *Tetris*. Chi-squared tests were performed on the individual collections of data as well as the sum of all of the games. Our calculations showed that these tests cannot reject the hypothesis that the distribution of the falling tetrominoes is uniform. Heidi Burgiel describes the distribution of the *Tetris* pieces as "randomly selected from

the set of seven possible tetrominoes" [2]. The results of these tests permit us to concur with this statement, but to say more. The seven pieces are randomly distributed with equal probability of any given piece.

The second factor, which certainly is dominant, is the control aspect of *Tetris*. However, uniform distributions are not routinely applied in optimal control theory. In general, a Gaussian model of random noise is used. This implies that there is a need for new mathematics to model *Tetris* and to optimize the model. In other words, the well-known methods of [4] seem to be inadequate for the task at hand. The more recent multiple objective approach of [5] will be adapted to the solution by the additional consideration of the random arrival of uniformly distributed pieces.

The dynamic programming method is described in [4] as a procedure based on a simple "optimality principle." This principle states the following:

Let  $(\{u(k)^*\}, \{x(k)^*\})$  be a pair of optimal control and state sequences, starting with the initial state  $x(0)$  at  $k=0$ , for the control problem. Then, for the same control problem starting with  $x(h)$  at  $k=h$ , where  $0 < h < N$ , the part of the above  $(\{u(k)^*\}, \{x(k)^*\})$ , with  $k=h, \dots, N$ , is also a pair of optimal solutions.

When we consider the control problem as a path problem, this principle states that an optimal path within a network from the origin node to the destination node is also optimal for all other originating nodes contained within that path. In its discrete interpretation, applying dynamic programming to solving *Tetris* involves finding a path through a large network while optimizing several objectives. Therefore, we model the game of *Tetris* as a control problem, and apply the optimality principle to find a nondominated solution sequence.

### Formulation

We consider *Tetris* as a control system of the form:

$$\begin{aligned} x(n+1, N) &= f(x(n, N), u(n, N)) \text{ where} \\ x(n, N) &= [a(n, N) \ b(n, N) \ c(n, N) \ k(n, N)]^T \\ u(n, N) &= [r(n, N) \ l(n, N) \ t(n, N) \ d(n, N)]^T \end{aligned}$$

where  $x$  and  $u$  are the state the control (column) vectors, respectively. The subscript,  $n$ , counts the cell level (height) of the playing board at which the piece's center of rotation is located. Thus,  $1 \leq n \leq 20$ . The superscript,  $N$ , counts the stage of the game and is therefore incremented after each piece falls onto the contour. The state vector is 4-dimensional, with components including a quantity for the number of the piece ( $a$ ), the orientation ( $b$ ), the horizontal center of rotation ( $c$ ), and the vertical center of rotation ( $k$ ). The control vector is also 4-dimensional with variables describing the number of movements right ( $r$ ), the number of movements left ( $l$ ), and the number of rotations performed ( $t$ ). The fourth component is a dummy variable indicating whether or not the drop control is

implemented (d). The assignment of the control as a vector rather than a single variable allows for simultaneous, multiple controls at each cell level. Each state and control variable possesses a subscript and superscript which corresponds to their associated vectors.

The vector state equation determines the state of the piece at level  $n+1$ , dependent on the state and control at level  $n$ . Specifically, we suggest the following formulation, which represents in a straightforward way what happens in *Tetris*:

$$\mathbf{X}(n+1, N) = \begin{bmatrix} a(n, N) \\ b(n, N) + t(n, N) \pmod{S} \\ c(n, N) + r(n, N) - 1(n, N) \\ \text{if } d(n, N) = 0: k(n, N) + 1 \\ \text{if } d(n, N) = 1: k(\max, N) = g(x(n, N)) \end{bmatrix}$$

This first component of the vector equation reflects the fact that the *Tetris* piece is constant within each stage and not dependent on the control. The variable  $S = S(a)$  is the number of unique orientations of the piece. When the drop command has been implemented,  $d(n, N) = 1$ , the vertical level of the center of rotation is maximized; therefore, the piece is at rest upon the contour at level  $n+1$ . Otherwise,  $k(n+1, N) = k(n, N) + 1$ . This corresponds the moving down one cell, which is what happens if "drop" is not used. Figure 1 shows that  $k(\max, N)$  is not a constant but rather a function of the state vector.

Once these state variables are defined, one needs to observe that there are still some difficulties. There exist both translation and rotation constraints in this model. The walls and the contour of the base restrict the state variables. For example, the center of rotation of the linear  $(4 \times 1)$  *Tetris* piece in its vertical configuration ( $a = 2$ ,  $b = 1$ ) must be more than two cells out from the left wall and more than one cell out from the right wall in order for rotation to occur (see Table 1 for further similar constraints). These state constraints can be handled explicitly by defining only the controls which do not violate them. The state constraints related to the contour need not be explicitly considered, since the control may be applied at the  $n - 1$  level and the piece dropped to the optimal placement.

After taking these constraints into account, the *Tetris* player's goals are to optimize certain functions. These goals are achieved by determining a sequence of optimal control inputs. Since there is normally more than one possible placement for each falling *Tetris* piece, we suggest multiple objectives be evaluated to determine the "best fit." We do not suggest the algorithms of [5], which compute all nondominated solutions. Our approach is to use the weighting method to form an appropriate linear combination of the objective functions. This approach will find a single nondominated solution, which is sufficient in the *Tetris* setting.

Now we give details about the choice of objective functions. One objective is to minimize the number of empty cells created between the piece and the contour

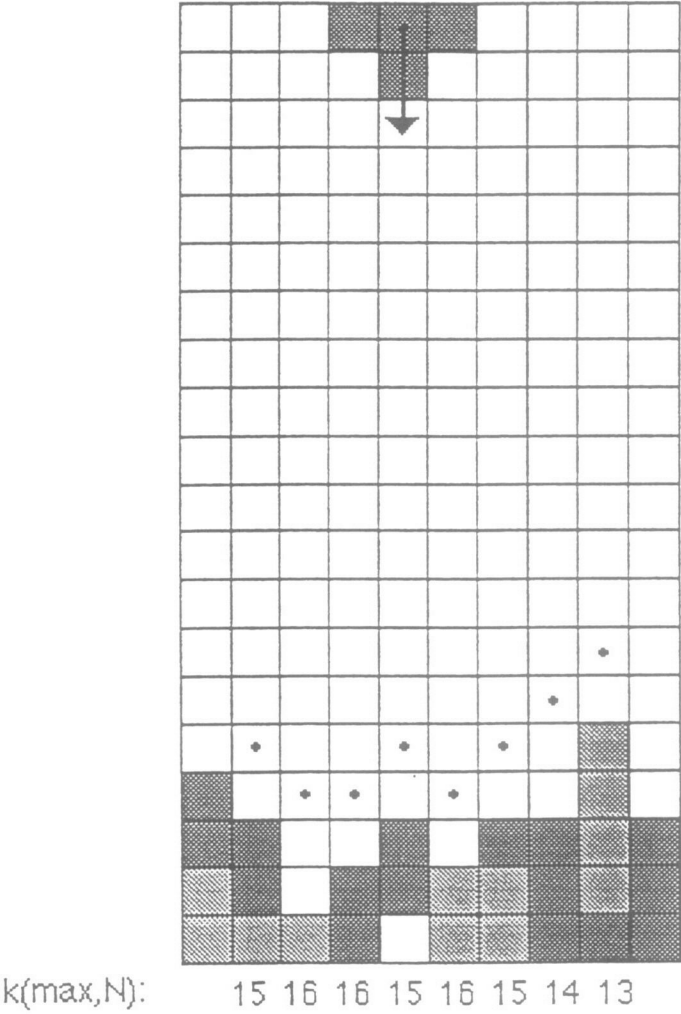






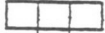



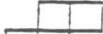



Figure 1.



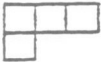

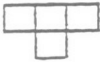
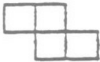
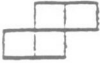
Table 1.

a	b <sup>a</sup>		Translation Domain	Rotation Domain
1	0		{1-9}	No restrictions
2	0		{1-7}	No restrictions
	1		{1-10}	{3-9}
3	0		{1-8}	No restrictions
	1		{1-9}	{2-10}
	2		{1-8}	No restrictions
	3		{1-9}	{1-9}
4	0		{1-8}	No restrictions
	1		{1-9}	{2-10}
	2		{1-8}	No restrictions
	3		{1-9}	{1-9}
5	0		{1-8}	No restrictions
	1		{1-9}	{2-10}
	2		{1-8}	No restrictions
	3		{1-9}	{1-9}
6	0		{1-8}	No restrictions
	1		{1-9}	{1-9}
7	0		{1-8}	No restrictions
	1		{1-9}	{2-10}

<sup>a</sup> b = 1,2,3 corresponds to the number of counterclockwise rotations performed on the piece.

associated with the placement. This one is well motivated by observing good human players. A second objective is to *maximize the number of contacting cell walls* resulting from the placement. Paul Maglio describes a related objective which he calls "snugness" [3]. A third objective is to *minimize the amount of additional height* that the placement adds to the contour. The fourth objective is to *maximize the number of rows cleared* from the board as a result of the placement. These objectives are evaluated for all orientations of the piece and for each possible placement along the contour. As indicated above, we combine the four objectives described here to create a single score for each possible placement.

Table 2.

a	b <sup>a</sup>		# Placements	Total
1	0		9	9
2	0		7	17
	1		10	
3	0		8	34
	1		9	
	2		8	
	3		9	
4	0		8	34
	1		9	
	2		8	
	3		9	
5	0		8	34
	1		9	
	2		8	
	3		9	
6	0		8	17
	1		9	
7	0		8	17
	1		9	

<sup>a</sup>b = 1,2,3 corresponds to the number of counterclockwise rotations performed on the piece.

Since the objectives are not measured using the same scale, weighting factors are implemented. That is, consider maximizing

$$\text{.Score}(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\}) = k_1*j_1(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\}) + k_2*j_2(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\}) \\ + k_3*j_3(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\}) + k_4*j_4(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\}).$$

In the above equation, the placement being scored is a function of the state and control vectors (see Table 2). The functions  $j_1(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\})$ ,  $j_2(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\})$ ,  $j_3(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\})$ , and  $j_4(\{\mathbf{x}(n,N),\mathbf{u}(n,N)\})$  represent the four objectives described above, respectively. Since the objectives do not agree in direction (max vs. min), the signs of the weighting factors are adjusted so that

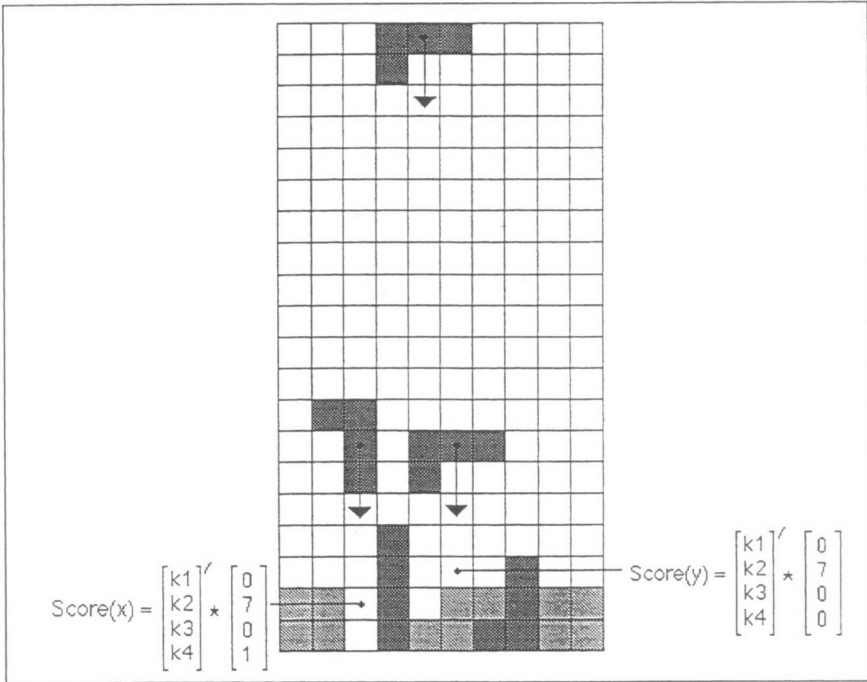


Figure 2.

maximizing score will result in an optimal placement. We therefore associate positive weighting factors with the objectives to be maximized,  $j_2$  and  $j_4$ , and negative weighting factors with those to be minimized,  $j_1$  and  $j_3$ . In the case of a tie in score between some placements, we arbitrarily choose between these best placements.

In some cases, one objective may become more important than the others. For example, if the height of the contour restricts the player's ability to rotate and translate effectively in a certain stage, he may put more weight on objective three. However, our current basic model assumes constant weighting factors from stage to stage. We therefore keep a vector of the individual objectives and use it to form the score function. Figure 2 shows a stage in which there are apparently two optimal placements. However, the associated objective vectors show that placement  $x$  is optimal with respect to the scoring function because it causes a line to be deleted, and thus has a higher score by  $k_4$  units.

To complete the model, we now consider the treatment of the uniformly distributed pieces. There are seven distinct *Tetris* pieces, which we view as distinct starting states for the development of *Tetris* play. For definiteness, consider a set of



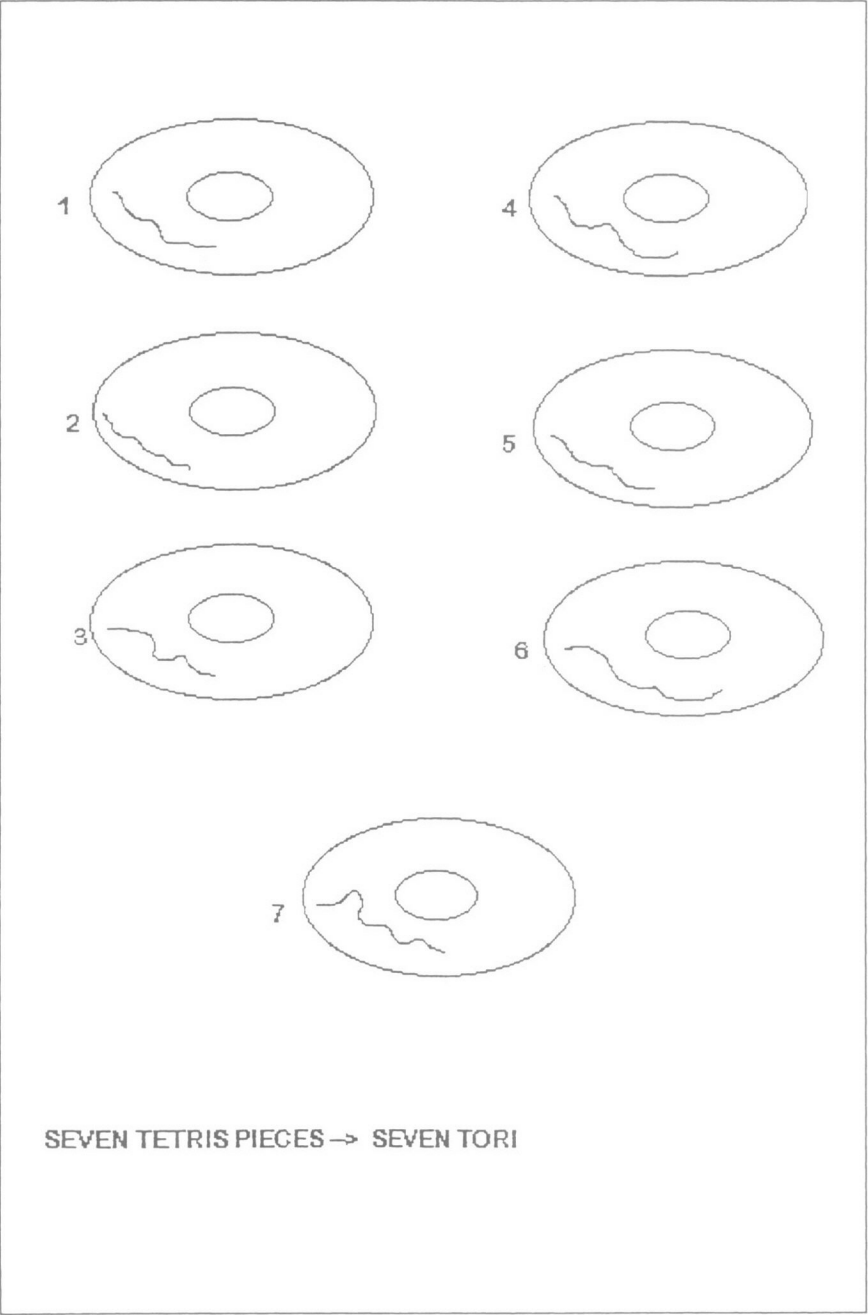


Figure 3.

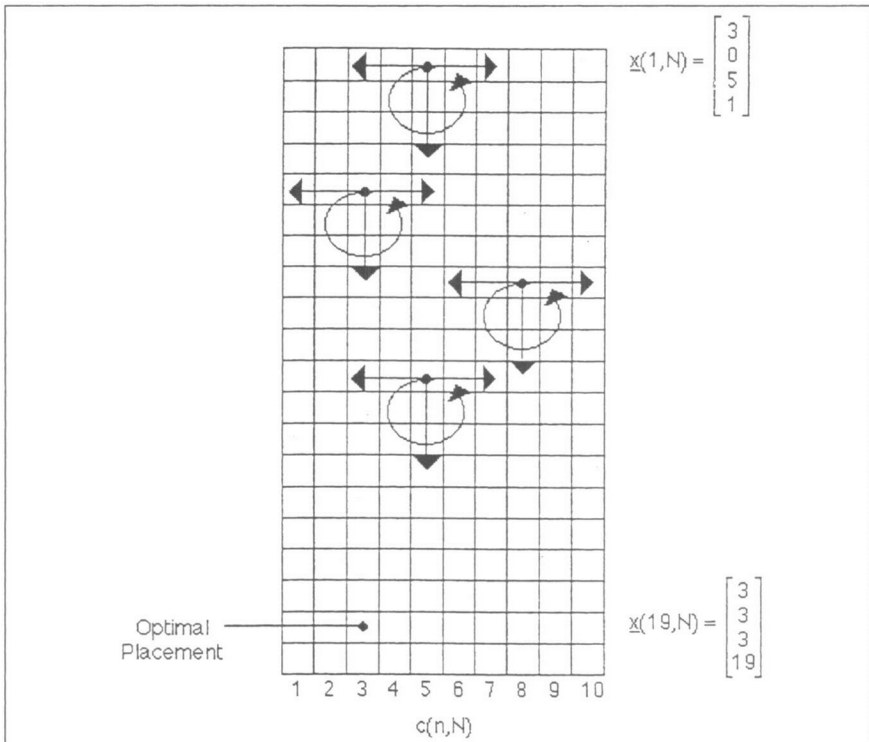


Figure 4.

seven tori, the surfaces of which are to contain the trajectories of the seven *Tetris* pieces ( $a=1,2,3,\dots,7$ ). Once a *Tetris* piece has been finally placed on the contour, its trajectory is temporarily halted on its associated torus. Then another piece (maybe the same type) is available for control, according to a uniform distribution. The trajectory is either started from the starting state or appended onto the existing trajectory from the last time this state was available (see Figure 3). For the total objective function calculation, a double summation including a summation over seven values of  $a$ , and a summation over all stages ( $N$ ) is required. This summation is to be understood in the sense that at most one value of  $a$  per stage is active.

The dynamic programming approach finds the optimal placement and corresponding path for each stage of Tetris. Figure 4 depicts the same *Tetris* stage shown in Figure 2, with corresponding points indicating the center of rotation of the *Tetris* piece. The arrows leaving these points correspond to the four possible controls. This figure shows the state vectors associated with the origin at level 1 and the optimal placement determined in Figure 2. The vector state equation may

be solved for the optimal control vector given these two state vectors. Multiple and simultaneous controls can then be applied to the piece while it is  $\text{kinmax}$  as long as the final sum of these applied control vectors is equal to the optimal control vector, determined by the vector state equation.

At this point, the dynamic programming theory had moved to rejecting the idea that the game can be played indefinitely. Rather, it finds a strategy for optimal play. As stated earlier, existing research in this area assumed that the distribution of the *Tetris* pieces is random. John Brzustowski proved that if the machine were programmed to play competitively rather than randomly, then there would exist no winning strategy for humans playing *Tetris*. He discussed a certain sequence of pieces which will ultimately lead to the player's loss and used the assumption of randomness to conclude that this sequence must eventually occur in an infinite string of *Tetris* pieces [1]. Burgiel modifies the proof of Brzustowski to present a proof that eventually all *Tetris* games must terminate [2]. The time for this to occur could be longer than a human lifetime, so it is mostly of the academic interest. Paul Maglio of the University of California, San Diego, presented much research on the game of *Tetris*, but his main focus was cognitive science. His program, *RoboTetris*, combined multiple objectives to produce "expert-level performance" [3]. Maglio did not claim to use dynamic programming in another context. Kostreva and Wiecek presented two algorithms for solving multiple objective dynamic programming [5]. Their *Algorithm 1* is based on forward dynamic programming to find all nondominated paths from the origin node to every node in the network. For *Tetris*, the selection of one nondominated strategy is sufficient, and is accomplished with the weighting factors.

The mathematical model and its solution described above have been applied *manually* to illustrate the performance of optimal play. This tedious process, which is likely to be automated by a computer program, has resulted in two sets of *Tetris* games with very high scores. In one set, the average number of lines removed was 11.845 while on another set of games, the average number of lines removed was 14.084. The manual application of dynamic programming was accomplished by a feature of the game in which the player is able to pause *Tetris*, decide what move is optimal, and then restart *Tetris* and quickly make the move.

In conclusion, a form of dynamic programming which is standard has been applied to get optimal solutions for mathematical models of the computer game *Tetris*. This form of dynamic programming is able to handle the particular type of random phenomena (i.e., uniformly distributed state perturbations) present in *Tetris*. A non-negative combination of four objectives (line loss) seems to be adequate to represent the goals of human players. There have been studies and presented in the literature, but more or fewer objectives can also be considered. Future research will tackle the implementation issues of this algorithm and may remove some of the simplifying assumptions.

**References:**

1. J. Brzustowski, *Can You Win at Tetris?* Master's Thesis, Department of Mathematics, University of British Columbia, Vancouver, March 1992.
2. H. Burgiel, *How to Lose at Tetris*, Manuscript, Geometry Center, University of Minnesota, Minneapolis, 1996.
3. P. Maglio, *The Computational Basis of Interactive Skill*, Ph.D. Dissertation, Cognitive Science Department, University of California, San Diego, 1995.
4. G. Chen, G. Chen, and S. Hsu, *Linear Stochastic Control Systems*, CRC Press, Boca Raton, 1995.
5. M. Kostreva and M. Wiecek, Time Dependency in Multiple Objective Dynamic Programming, *Journal of Mathematical Analytic Applications*, 173, pp. 289-307, 1993.