1 Nash Equilibrium for mixed strategies

Nash shown that every non-cooperative game with finite sets of pure strategies has at least one mixed strategy equilibrium pair. We define such pair as a *Nash equilibrium.* For a two-player game, where the matrices A and B define the cost for players 1 and 2 respectively, the strategy (y^*, z^*) is a Nash equilibrium if:

$$\begin{aligned} {y^*}^T A z^* &\leq y^T A z^* \quad \forall y \in Y \\ {y^*}^T B z^* &\leq {y^*}^T B z \quad \forall z \in Z \end{aligned}$$

in which Y and Z are the sets of possible mixed strategies for players 1 and 2 respectively. Remember that the elements of Y and Z are vectors defining the probability of choosing different strategies. For example, for a $y \in Y$, $y = [y_1, y_2, ..., y_m]^T$, we have $\sum_{i=1}^{m} y_i = 1$, in which $y_i \geq 0$ defines the probability of choosing the strategy i.

If a player plays the game according with the strategy defined by the Nash equilibrium, then we say that the player is using a *Nash strategy.* The Nash strategy safeguards each player against attempts by any one player to further improve on his individual performance criterion. Moreover, each player knows the expected cost for the game solution (y^*, z^*). For player 1 the expected cost is ${y^*}^T A z^*$, and for player 2 is ${y^*}^T B z^*$. For the case of two players, the Nash equilibrium can be found using quadratic programming. In general, for multi-player games, the Nash equilibrium is found using non-linear programming.

This solution generally assumes that the player know each other's cost matrices and that, when the strategies have been calculated, they are announced at the same instant of time.

Note that the Nash strategy does not correspond in general with the security strategy. When the game has a unique Nash equilibrium for pure strategies, then the Nash equilibrium maximizes the security strategy.

2 Sequential games

Until now we have used matrices to describe the games. This representation is called *normal form.* For sequential games (i.e., parlor games), in which a player take a decision based on the outcome of previous decisions of all the players, we can use the *extensive form* to describe the game.

The rules of a sequential game specify a series of well defined *moves*, where each move is a point of decision for a given player from among a set of alternatives. The particular alternative chosen by a player in a given decision point is called *choice*, and the totality of choices available to him at the decision point is defined as the *move.* A sequence of choices, one following another until the game is terminated is called a *play.* The extensive form description of a sequential game consist of the following:

- A finite tree that describes the relation of each move to all other moves. The root of the tree is the first move of the game.
- A partition of the nodes of the tree that indicates which of the players takes each move.
- A refinement of the previous partition into information sets. Nodes that belong to the same information set are indistinguishable to the player.
- A set of outcomes to each of the plays in the game.

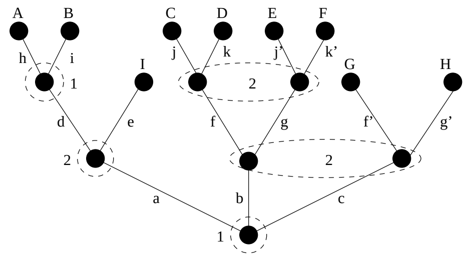

Figure 1: A tree for the extensive form.

Figure 1 shows an example of a tree for a sequential game. The numbers beside the nodes indicates which player takes the corresponding move. The edges are labeled by the corresponding choice selected. The leaves indicate the outcome of the play selected (a root-leaf path in the tree). The information sets are shown with dashed ellipses around the nodes. Nodes inside the same ellipse are indistinguishable for the players, but the players can differentiate nodes from one information set to another. If every ellipse enclose only one node, then we say that the players have *perfect information* of the game, which leads to a "feedback strategy".

In the extensive form all games are described with a tree. For games like chess this may not seem reasonable, since the same arrangement of pieces on the board can be generated by several different routes. However, for the extensive form, two moves are different if they have different past histories, even if they have exactly the same possible future moves and outcomes.

3 Derivation of the Dynamic Programming equation

Using the cost-to-go (backward dynamic programming), we define the cost in the final state as:

$$L^*_{F,F}(x_F) = \ell_F(x_F)$$

in which $\ell(x_F)$ denotes the value of the loss function at the final state x_F and $L^*_{k,F}(x_k)$ is the optimal cost of going from state x_k to state x_F. We define $L^*_{k,F}$ as follows:

$$L^*_{k,F}(x_k) = \min_{u_k;u_K} \left\{ \sum_{i=k}^{K} \ell(x_i, u_i) + \ell_{k+1}(x_{k+1}) \right\}$$

Decomposing the sequence of "min" of the last equation:

$$L^*_{k,F}(x_k) = \min_{u_k} \min_{u_{k+1};u_K} \left\{ \ell(x_k, u_k) + \sum_{i=k+1}^{K} \ell(x_i, u_i) + \ell_F(x_F) \right\}$$
$$L^*_{k,F}(x_k) = \min_{u_k} \left\{ \ell(x_k, u_k) + \min_{u_{k+1};u_K} \left[\sum_{i=k+1}^{K} \ell(x_i, u_i) + \ell_F(x_F) \right] \right\}$$

but we know that:

$$L^*_{k+1,F}(x_{k+1}) = \min_{u_{k+1};u_K} \left[\sum_{i=k+1}^{K} \ell(x_i, u_i) + \ell_F(x_F) \right]$$

and we can write the BDP equation as:

$$L^*_{k,F}(x_k) = \min_{u_k} \left\{ \ell(x_k, u_k) + L^*_{k+1,F}(x_{k+1}) \right\}$$

4 Sequential Decision Making with continuous X and U

Here we consider the case when the state space and the space of control inputs are continuous, namely $X \subset \Re^n$ and $U \subset \Re^m$. We consider X and U to be closed and bounded. One approach to apply the concepts we saw during the class to this case is to sample the continuous spaces. One desirable characteristic for subsequent samples of a function is that the value of the function should not be very different. Note that this is not solved by choosing samples arbitrarily close. For example, if $f = tan(u)$, the function has a finite time escape and for samples close to $pi/2$, the value of f has very big changes. To formalize the requirements for X and U, the *Lipschitz condition* is introduced.

We say that the function $f : \Re \to \Re$ is Lipschitz if:

$$\frac{\|f(x_1) - f(x_2)\|}{\|x_1 - x_2\|} \leq L \qquad \forall x_1, x_2 \in \Re,\ L \in (0, \infty)$$

in which $\|x\|$ denotes any p-norm, and L is known as the Lipschitz constant. The Lipschitz condition allow us to state the requirement described before:

$$\frac{|\mathrm{d}f(x)|}{|\mathrm{d}x|} \leq c \qquad c \in (0, \infty)$$

That is, the absolute value of the derivative is bounded by the Lipschitz constant c. For our problem, we apply the Lipschitz condition to the state transition equation and the loss function as:

$$\begin{aligned} |f(x,u) - f(x',u')| &\leq \alpha_1 |x - x'| \\ |l(x,u) - l(x',u')| &\leq \alpha_2 |x - x'| \\ |l_F(x,u) - l_F(x',u')| &\leq \alpha_3 |x - x'| \end{aligned}$$

Using the samples, for example, in the dynamic programming computations we are left with one problem:

$$L^*_{k,F} = \min_{u_k} \left\{ l(x_k, u_k) + L^*_{k+1,F}(x_{k+1}) \right\} \tag{1}$$

What value should be use for $L^*_{k+1,F}(x_{k+1})$? Remember that X is continuous, but now we only have samples of X. One way to solve this is to use interpolation. For example, assume that x_{k+1}

is between the samples x and x', and the distance between adjacent samples is normalized (i.e., $|x - x'| = 1$). If $|x - x_{k+1}| = \beta$, we can compute the value as:

$$L^*_{k+1,F} = \beta L^*_{k+1,F}(x) + (1-\beta)L^*_{k+1,F}(x')$$

We expect some errors because of this discretization. One them, as we can easily see, is the error in the estimation of $L^*(x)$. Another error is related to the cost by executing the strategy using samples instead of the continuous space. Even if the Lipschitz condition is satisfied, we may end with very different strategies.

One important question is: how to sample X and U? We want samples through all the space in order to reduce the estimation errors, but if the dimension of X, for example, is high, the number of samples needed will be also high. An easy way to sample will be to take random samples, with the hope that some of them will be useful, or we can also sample using a grid. We can establish some measures to evaluate the quality of different sampling schemes. Two of such measures are the *discrepancy* and the *dispersion*.

The discrepancy of a set of samples $P \subset X$ is defined as:

$$D(P) = \max_{r \in R} \left| \frac{|P \cap r|}{|P|} - \frac{\mu(r)}{\mu(X)} \right| \tag{2}$$

in which R usually is the set of all axis aligned rectangular boxes, and $\mu(x)$ is a measure, like area, volume, etc. The discrepancy measures "how well" the proportion of the volume of the boxes is approximated by the proportion of the samples in the space. A high discrepancy indicates that we found a box whose size is very big of very small for the number of samples inside it. Since we use axis align boxes, grids may have very high discrepancy, since it is possible to find big boxes including only one sample. But if we increase the box by a small epsilon it may include many other samples. For example, in 2D, we can find a box including one sample, and if we increase the volume by a small epsilon, the number of samples inside the box will be 9.

The dispersion is a measure of how far the nearest neighbor of a sample could be. It is defined as:

$$\delta(P) = \max_{x \in X} \min_{p \in P} \|x - p\|$$

The dispersion is telling us the radius of the biggest ball centered at one sample and that only contains that sample. Usually we want this radius to be small. For a fixed number of samples n in a space of dimension d it has been proved that:

$$\delta(P) \geq \frac{1}{2\lfloor n^{1/d} \rfloor} \tag{3}$$

This bound is achieved sampling with grids. A sampling strategy to keep the dispersion low is to divide the space into cubes, and to generate a sample inside of each cube.

Another consideration that is taken into account in the sampling strategies is if the number of samples is fixed, or we are allowed to increase the number of samples as they are needed. For the first case we deal with a finite point set, while for the second we may use infinite sequences to generate the samples.

As an example of how an infinite sequence works, we sample the interval $[0, 1]$ using the *van Der Corput sequence*. This sequence has the advantage that generates very low dispersion. To form this sequence, the bits representing decimal numbers in base two inside the interval are reversed, as is shown for 3 bits in the next table:

dec	original	reversed	sample
.000	.000	.000	.000
.125	.001	.100	.500
.250	.010	.010	.250
.375	.011	.110	.750
.500	.100	.001	.125
.625	.101	.101	.625
.750	.110	.011	.375
.875	.111	.111	.875

If only two samples are generated, these will be 0 and 0.5. As more samples are needed, bits are added to the binary representation. If we want to apply this construction to sample in higher dimensions, for every dimension we add, the new dimension is sampled with the next prime number available. This construction is called *Halton sequence.*

References

[1] Steve LaValle's classes

[2] G. Leitmann (editor). Multicriteria decision making and differential games. Plenum Press, 1976.

[3] D. Luce and H. Raiffa. Games and Decisions. Dover Publications, 1989.

[4] H. Khalil. Nonlinear systems. Prentice Hall, 2002.