Computational Intelligence in Games - Cheat Sheet -

Alexander Dockhorn

June 20, 2018

This short overview tries to provide you a guide for all equations and algorithms mentioned in the lecture Computational Intelligence in Games at the Otto von Guericke University Magdeburg. It can be quite hard to understand all the little details involved without being sure about the symbols used. We hope this guide helps you during your review of the course topics.

I currently do not recommend printing this guide, because it is the first version of this kind of overview. Even if I reviewed the contents multiple times, errors are still likely. Please always refer to the actual version in the Git-Repository.

In case you have any ideas how to improve this guide please let us know by writing an e-mail to: dockhorn@ovgu.de.

Contents

List of Equations

List of Algorithms

1 (Evolutionary) Game Theory

1.1 Basics in Game Theory

Symbol	Name	Description
N	number of agents	the number of agents in an N-player game
S_i	strategies of player i	each player has a set of strategies, which he can choose to play
s_i	chosen strategy of player i	
m_i	number of strategies in S_i	
π_i	payoff function of player i	provides a reward to agent i after each agent chose his action
A	payoff matrix player 1 (or both)	represents the payoff matrix of the first player, or of both players in case the payoff is symetric
B	payoff matrix player 2	

1.2 General 2-Player Games

Symbol	Name	Description
C	strategy "Cooperate"	often used in standard literature to describe cooperative behavior, which comes with additional cost for the playing agent
D	strategy "Defect"	often used in standard literature to describe cooperative behavior, which comes without a cost for the playing agent
T	temptation	Reward for defecting, when the other player is Cooperating
R	reward for mutual cooperation	reward in case both players are cooperating
S	suckers payoff	reward for the player who cooperated against a player who defected him
P	punishment	reward for the player when both players chose defect

1.3 Nash Equilibria

Symbol	Name	Description
$\mathcal{S}$	strategy profile	the chosen strategies per player
x	mixed strategy	probability distribution on all possible strategie in S_i
$E(s_i, y)$	expected payoff of a pure strategy	expected payoff of pure strategy s_i against mixed strategy y. See Equation (1)
$E(x, y)$	expected payoff of a mixed strategy	expected payoff of mixed strategy x against mixed strategy y. See Equation (2)

Expected payoff of pure strategy s_i, $E(s_i, y)$

$$A = \begin{bmatrix} \pi_{s_1,s'_1} & \pi_{s_1,s'_2} \\ \pi_{s_2,s'_1} & \pi_{s_2,s'_2} \end{bmatrix}, \qquad E(s_i, y) = \sum_{j=1}^{|S_j|} \pi_{s_i,s'_j} y_j \tag{1}$$

Expected payoff of mixed strategy x, $E(x, y)$

$$A = \begin{bmatrix} \pi_{s_1,s'_1} & \pi_{s_1,s'_2} \\ \pi_{s_2,s'_1} & \pi_{s_2,s'_2} \end{bmatrix}, \qquad E(x, y) = \begin{bmatrix} x_1, & x_2 \end{bmatrix} \begin{bmatrix} \pi_{s_1,s'_1} & \pi_{s_1,s'_2} \\ \pi_{s_2,s'_1} & \pi_{s_2,s'_2} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{2}$$

1.4 Evolutionary Game Theory

Symbol	Name	Description
x	frequency of strategies	similar to the mixed strategy, but here the population frequency is given by x
s_i	strategy with index i	index corresponds to the frequency vector x
$f_i(x)$	fitness of strategy s_i	See Equation (3)
$\Phi(x)$	average population fitness	See Equation (4)
$\dot{x}$	replicator equation of x	shows the time and fitness dependent development of each strategies frequency, is computed for every s_i separately, See Equation (5)

Evolutionary Stable Strategy: A strategy σ is an evolutionary stable strategy if:

- $\pi_{\sigma,\sigma} > \pi_{\tau,\sigma}$ for all possible mutant strategies τ
- or, $\pi_{\sigma,\sigma} = \pi_{\tau,\sigma}$ and $\pi_{\sigma,\tau} > \pi_{\tau,\tau}$

Fitness of strategy s_i, $f_i(x)$

$$fitness(s_i) = f_i(x) = payoff(s_i) = \sum_{j=1}^{n} x_j \pi_{ij} \tag{3}$$

Average population fitness, $\Phi(x)$

$$\Phi(x) = \sum_{j=1}^{n} x_j f_j(x) \tag{4}$$

Replicator Equation for infinite populations, $\dot{x}$

$$\dot{x} = x_i[f_i(x) - \Phi(x)] \tag{5}$$

There seems to be something wrong with our current definition of replicator equations for finite populations. Please ignore this part for now! We will update the slides and this overview after a careful revision of this part.

1.5 Additional Information on Evolutionary Game Theory

Symbol	Name	Description
β	selection intensity	used to apply weak selection, See Equation (6)
ρ	fixation probability	probability that a mutant strategy can take over the whole population, See Equation (7)
Δf	fitness difference of two strategies	
b	benefit	benefit for playing with a cooperating player, See Equation (8)
c	cost	cost for playing cooperation

Weak Selection, $\dot{x}$

$$f_A(i) = 1 - \beta + \beta\pi_A(i)$$

$$\begin{aligned} \beta = 0 &\Rightarrow \text{neutral selection} \\ \beta \ll 1 &\Rightarrow \text{weak selection} \\ \beta = 1 &\Rightarrow \text{selection by payoff} \end{aligned} \tag{6}$$

Probability for replacing a strategy using weak selection ρ

$$\rho = \frac{1}{1 + exp^{-\beta\Delta f}} \tag{7}$$

Comparison of using T,R,S,P or b,c for describing a payoff matrix

$$\begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C' & D' \end{array} \\ \begin{bmatrix} b-c & -c \\ b & 0 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cc} C' & D' \end{array} \\ \begin{bmatrix} R & S \\ T & P \end{bmatrix} \end{array} \tag{8}$$

2 Reinforcement Learning

2.1 General Notation

Symbol	Name	Description
t	timestep $0 \dots T$	
T	final timestep T	
$\mathcal{S}$	set of possible states	
s_t	state at time t	$s_t \in \mathcal{S}$, the t is dropped in case the equation if indepentent of the actual time
$\mathcal{A}(s_t)$	available actions in s_t	
a_t	action at time t	$a_t \in \mathcal{A}(s_t)$, the t is dropped in case the equation if indepentent of the actual time
r_t or R_t	reward at time t	after the agent left state s_t by executing action a_t, he receives the reward r_{t+1}, where $r_{t+1} \in \mathbb{R}$, depending on the context we use upper-case R to not confuse the single reward with the probability distribution at the bottom of this table
π	policy function	a probability distribution on the actions depending on the state
$\pi(a\|s)$	policy	probability of choosing action a if in state s
$G_t,\ G_s$	(expected) return	sum of rewards from s or timestep t till the end T (undiscounted return, See Equation (9)) or a non-ending episode (discounted return, See Equation (10))
γ	discount rate	effects the influence of future rewards on the return calculation (discounted return, See Equation (10))
$p(s', r\|s, a)$	state and reward probability depending on a and s	also known as the markov property, probability of transitioning to state s' and receiving reward r is only dependent on choosing action a in state s,
$p(s'\|s, a)$	transition probability	probability of transitioning into s' after the agent executed its action a in state s
$r(s, a)$	expected reward	valid for all Markov Decision Processes, See Equation (11)
$r(s, a, s')$	expected reward after the follow-up state is known	valid for all Markov Decision Processes, See Equation (12)

Simple (undiscounted) Return, G_t

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{9}$$

Discounted Return, G_t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \quad \gamma \in [0, 1] \tag{10}$$

Expected Reward, $r(s, a)$

$$r(s, a) = \mathbb{E}[R_{t+1} | s_t = s, a_t = a] = \sum_{r \in \mathbb{R}} \sum_{s' \in \mathbb{S}} r(s, a, s') \cdot p(s'|s, a) \tag{11}$$

Expected Reward, $r(s, a)$

$$r(s, a, s') = \mathbb{E}[R_{t+1} | s_t = s, a_t = a, s_{t+1} = s'] = \frac{\sum_{r \in \mathbb{R}} r \ p(s', r|s, a)}{p(s'|s, a)} \tag{12}$$

2.2 Value and Action-Value Functions

Symbol	Name	Description
$v_\pi(s)$	value function	rates the value/expected return of a state following policy π, See Equation (13) and Equation (14)
π^*	optimal policy	policy with the best possible expected return for all states
v^*	optimal value function	value function of the optimal policy π^*
$q_\pi(s,a)$	action-value function	rates the value/expected return of choosing an action in a given state following policy π, See Equation (15) and Equation (16)
$V(s)$	approximation of $v(s)$	used in multiple iterative algorithms
$Q(s,a)$	approximation of $q(s,a)$	used in multiple iterative algorithms

Value Function, $v(s)$

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle|s_t = s\right] \tag{13}$$

Consistency Condition of the Value Function, $v(s)$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|s_t = s] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right] \end{aligned} \tag{14}$$

Action-Value Function, $q(s,a)$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k t_{t+k+1}\middle|s_t = s, a_t = a\right] \tag{15}$$

Consistency Condition of the Action-Value Function, $q(s,a)$

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \end{aligned} \tag{16}$$

2.3 Dynamic Programming

Algorithm 1: Iterative Policy Evaluation

Input: policy π to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S$

repeat
  $\Delta \leftarrow 0$
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s,r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ *(a small positive number)*
Output: $V \approx v_\pi$

Algorithm 2: Policy Iteration

Input: policy π to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S$

repeat
  apply Iterative Policy Evaluation given π
  policy-stable $\leftarrow$ *true*
  foreach $s \in S$ **do**
    $a \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s'.r|s,a)]r + \gamma V(s')]$
    If $a \neq \pi(s)$ **then** *policy-stable* $\leftarrow$ *false*
until *policy-stable*
Output: $V \approx v^*,\ \pi \approx \pi^*$

Algorithm 3: Value Iteration

Initialize an array $V(s) = 0$, for all $s \in S$

repeat
  $\Delta \leftarrow 0$
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta \leftarrow 0$ *(a small positive number)*
Output: a deterministic policy π such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

2.4 Monte Carlo Method

Algorithm 4: Monte Carlo Method $v(s, a)$

Input: policy π to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S$
Initialize an empty list *Returns*(s), for all $s \in S$

while *true* **do**
  Generate an episode using π
  foreach *state s appearing in the the episode* **do**
    $G \leftarrow$ collected return after the first occurrence of s
    Returns(s)*.append*(G)
    $V(s) \leftarrow$ *average(Returns(s))*

Output: $V \approx v_\pi$

Algorithm 5: Monte Carlo Method $q(s, a)$

Input: policy π to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S$
Initialize an empty list *Returns*(s, a), for all $s \in S$ and $a \in A$

while *true* **do**
  Choose $s_0 \in S$ and $a_0 \in A(s_0)$
  Generate an episode starting from s_0, a_0 following π
  foreach (s, a) *in the episode* **do**
    $G \leftarrow$ return following the first occurence of (s, a)
    Returns(s, a)*.append*(G)
    $Q(s, a) \leftarrow \arg\max_a Q(s, a)$
  foreach s *in the episode* **do**
    $\pi(s) \leftarrow \arg\max_a Q(s, a)$

Output: $Q \approx q_\pi, \quad \pi \approx \pi^*$

Derivation of the Incremental Mean and Incremental Monte Carlo, $\mu_k, V(s)$

$$\begin{aligned} \mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\ &= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\ &= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1}\frac{1}{k}(x_k - \mu_{k-1}) \end{aligned} \tag{17}$$

$$\begin{aligned} V(s) &\leftarrow \frac{1}{k}\sum_{i=1}^{k} G_s(i) \\ &\leftarrow \frac{1}{k}\left(G_s(k) + \sum_{i=1}^{k-1} G_s(i)\right) \\ &\leftarrow \frac{1}{k}(G_s(k) + (k-1)\ G_s(k-1)) \\ &\leftarrow G_s(k-1) + \frac{1}{k}(G_s(k) - G_s(k-1)) \\ &\leftarrow V(s) + \frac{1}{k}(G_s(k) - V(s)) \end{aligned} \tag{18}$$

Algorithm 6: Constant-α Monte Carlo Method $v(s, a)$

Input: policy π to be evaluated
Initialize an array $V(s) = 0$, for all $s \in S$
Initialize an empty list *Returns*(s), for all $s \in S$

while *true* **do**
  Generate an episode using π
  foreach *state s appearing in the the episode* **do**
    $G_s \leftarrow$ collected return after the first occurrence of s
    $V(s) \leftarrow V(s) + \alpha[G_s - V(s)]$

Output: $V \approx v_\pi$

2.5 Temporal Difference Learning

Symbol	Name	Description
G_t^{t+n}	n-step return	See Equation (19)
h	horizon $= t + n$	number of future reward steps used for return calculation, if $h > T$ the return is the sum of rewards till the end of the episode
V_t	approximation of v at timestep t	See Equation (19)
$\Delta_t(s_t)$	error based adaptation	Weighted error used in temporal difference learning, See Equation (19)

n-Step Return, $G_t^{t+n}(s)$

$$\begin{aligned} G_t^{t+n}(s) &= R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_h + \gamma^n V_t(s_{t+n}) \\ \Delta_t(s_t) &= \alpha[G_t^{t+n}(V_t(s_{t+n})) - V_t(s_t)] \\ V_{t+1}(s) &= V_t(s) + \Delta_t(s), \qquad \forall s \in \mathcal{S} \end{aligned} \tag{19}$$

Algorithm 7: Temporal Difference Learning

Input: policy π to be evaluated
Initialize an array $V(s)$ arbitrarily, for all $s \in S$

foreach *for each episode* **do**
    Initialize S
    foreach $s \in S$ **do**
        $A \leftarrow$ action given by π for S
        Take action A; observe reward R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$

Output: $V \approx v_\pi$

Algorithm 8: One Step Q-Learning

Initialize a matrix $Q(s, a)$ arbitrarily, for all $s \in S$, and $a \in A$
foreach *terminal state s* **do** $Q(s, \cdot) = 0$
foreach *episode* **do**
    foreach *state* s_t *in episode* **do**
        Choose a_t from s_t using policy derived from Q
        take action a_t, observe reward r, and follow-up state s_{t+1}
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
        $s \leftarrow s_{t+1}$

Output: $Q(s_t, a_t)$

3 Simulation Based Search Algorithms

3.1 Flat Monte Carlo

Algorithm 9: Flat Monte Carlo

Input: Initial state s_t and MDP of the environment
Initialize an array $Q(s_t, a) = 0$, for all $a \in A$

foreach $a \in A$ **do**
  Simulate k Episodes using random policy
  $Q(s_t, a) \leftarrow$ mean/expected return of simulated episodes

Output: $\arg\max_a Q(s_t, a)$

3.2 Flat Upper Confidence Bounds (UCB)

Algorithm 10: Flat UCB

Input: any UCB variant
Initialize arrays $Q(s_t, a) = 0,\ N(s) = 0, N(s, a) = 0, \forall s \in S, \forall a \in A$
Initialize an empty list of *Returns(s)* $\forall s \in S$

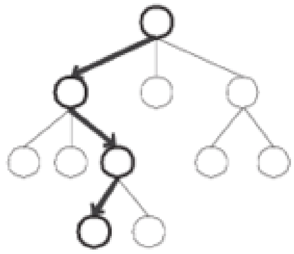
for k *iterations* **do**
  Select first action using UCB
  Simulate episode using random policy
  Update visit counts $N(s), N(s, a)$
  Returns(s).append(return of episode k)
  $Q(s_t, a) \leftarrow mean(Returns(s))$;

Output: $\arg\max_a Q(s_t, a)$

3.3 Monte Carlo Tree Search (MCTS)

- **1. Tree Selection:**

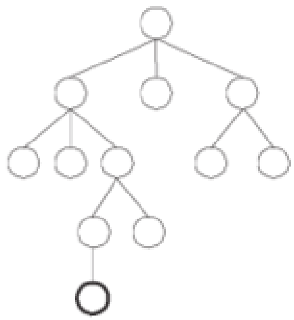
  Following the **tree policy** (i.e. UCB1), navigate the tree until reaching a node with at least one child state that was not explored yet.



- **2. Expansion:**

  Add a new node in the tree, as a child of the node reached in the tree selection step.

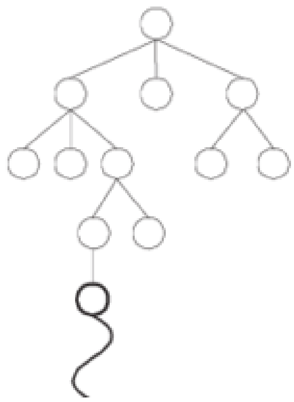
  - this node resembles an action



- **3. Monte Carlo Simulation/Rollout:**

  Following the **default policy**, advance the state until a terminal state or a pre-defined maximum number of steps.

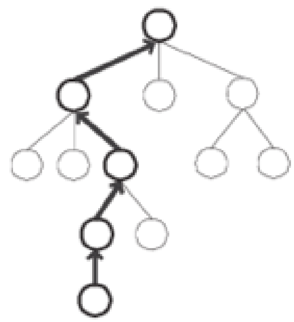
  - calculate the return of this episode.



- **4. Back-propagation:**

  Update the values of $Q(s, a)$, $N(s)$ and $N(s, a)$ of the nodes visited in the tree during steps 1 and 2.

  - Values of nodes visited during the simulation will not be updated.



3.4 Evolutionary Algorithms (EA)

Symbol	Name	Description
i	individual/solution	a possible solution for the problem at hand
P	population of individuals	set of possible solutions
t	time-step or generation	
$P_t, P(t)$	population at time t	
μ, N	(number of) individuals in P	equal to $\|P\|$
M	mating pool	individuals selected for genetic operators
λ	(number of) inidividuals in M	set of children
f_i	fitness of individual i	scoring the fitness of an individual based on an objective function
p_i	selection probability of individual i	probability of an individual being selected for the mating pool

Algorithm 11: General Evolutionary Algorithm

Input: Initial Population $P(t)$
$t \leftarrow 0$

repeat
  Select individuals for genetic operators: $M(t) \leftarrow \textit{selection}(P(t))$
  Apply Crossover: $M'(t) \leftarrow \textit{crossover}(M(t))$
  Apply Mutation: $M''(t) \leftarrow \textit{mutation}(M'(t))$
  Update $P(t)$: $P(t+1) \leftarrow \textit{reproduction\_scheme}(P(t) \cup M''(t))$
  $t \leftarrow t+1$
until *termination condition*
Output: Final Population $P(t)$

3.5 Rolling Horizon (RH)

Algorithm 12: Rolling Horizon Random Search

Input: current state s_t, horizon h

$E \leftarrow$ simulate a random episode starting at s_t till timestep $t + h$

repeat
    $E' \leftarrow$ simulate a random episode starting at s_t till timestep $t + h$
    if *Return of* $E' >$ *Return of* E **then**
        $E \leftarrow E'$
until *time is over*

Output: First action of E

Algorithm 13: Rolling Horizon Evolutionary Algorithm (RHEA)

Input: current state s_t, horizon h

$P \leftarrow$ population of random episodes starting at s_t till timestep $t + h$

repeat
    $M \leftarrow$ select individuals from P for genetic operators
    $M' \leftarrow$ apply crossover to M
    $M'' \leftarrow$ apply mutation to M'
    $P \leftarrow$ apply reproduction scheme to $(P \cup M'')$
until *time is over*

Output: First action of the episode with the highest return in P

4 Multi Objective Optimization

4.1 General Notation

Symbol	Name	Description
S	search space	the space of all possible solutions
O	objective space	$O = \{\vec{f}(x) \in \mathbb{R}^m \vert x \in S\}$
$\vec{f}(x)$	objective vector	all objective values for individual x
f_i	objective function	assigns an objective value to each solution
$f_i(x)$	i-th objective value of x	
r_i	rank of solution i	see the lecture for ranking methods
d_i	(crowding) distance of i	specifically used in NSGA-II
$\Omega_i(x)$	cone domination	cone-dominating version of objective function f_i
$C(A, B)$	convergence metric	compare the convergence between two non-dominated set of points A and B
$HV(A)$	hyper-volume of A	calculate the hyper-volume of a non-dominated set of solutions A
$MHV(i)$	marginal hyper volume of individual i	$MHV(i) = HV(entire\ Set) - HV(set\ without\ i)$

4.2 NSGA-II Algorithm

Algorithm 14: NSGA-II Algorithm

Input: Individuals of Population $P(t)$ and the Mating Pool $M(t)$
Fronts $(F_1, F_2, \cdots) \leftarrow$ non-dominated sorting of $P(t) \cap M(t)$
Set $P(t+1) = \emptyset, \quad i = 1, \quad N = P(t)$

repeat
  $P(t+1) \leftarrow P(t+1) \cap F_i$
  $i \leftarrow i + 1$
until $|P(t+1)| + |F_i| \leq N$

Sort all individuals in F_i by their crowding distance
$P(t+1) \leftarrow P(t+1) \cap \{\text{first } N - |P(t+1)| \text{ elements in } F_i\}$

Output: Next Population $P(t+1)$