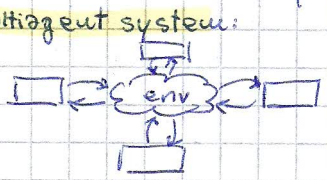
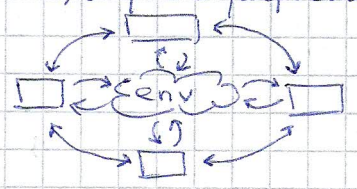
SUMMARY OF AAMAS

Autonomous agent: how can it decide his actions? → to maximize an utility function
- PREPROGRAM: decided by designer, environment very controlled
- ONLINE DECISION: equip the agent with the ability of decide what to do
- LEARNING AGENTS: equip with the ability of learn, improve performances over time.

Multiagent system:



interaction between agents by means of a wall



agents exchange messages

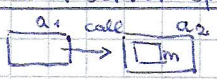
Interaction
- cooperative: to reach the global goal of the system, interaction to do sth together
- competitive: agents try to maximize proper utility (no general goal)

Regulate behavior
- pre-programmed decisions: efficient but doesn't cope with dynamic environm. or errors
- autonomous behavior: you tell the goal and leave freedom but you have to find a way to regulate the behavior of the single component

Human Organization
- entities: eterogeneous, cognitive (humans) or autonomous (agents)
- relationships: create an order of behaviors (es: A boss of B, negotiation)
- norms: constraints and regulation that cannot be violated (oth. you have a cost)
- environment: the more complex it is, the more complex the system will be.

Agent properties:
- autonomy: operate without human supervision
- social ability: interaction with other agents, humans and environment
- reactivity: respond in a fast way to changes in the environment
- proactivity: goal directed behavior. composition of reactive and proactive behavior

Client-server approach



m accepts the call
Caller has the power

Agent-based approach

a_2 decides if satisfy the request
Provider of the service has the power

- **reasoning agents (AI)**: agent able to perceive sth, decide and execute (planning, STRIPS, PDDL, A*)
- **reactive agents**: global behavior obtained by composing simple behaviors (subsumption architecture), when several behaviors are applicable define an order of priority.
- **BDI agents**: believes (what you know), desires (goals), intention (plan to execute)
- **Layered agents**: usually reactive in low, reasoning going higher.
  Three levels: reactive - planning (pro-active) - communication (social)

MARKOV DECISION PROCESSES (MDPs)

S: states that the agent can distinguish
A: action " " is able to perform
p: $S \times A \to \Delta S$ set of all prob distribution over all elements of S. $P(s'|s,a)$ (transition function) $S \times A \times S \to [0,1]$
r: $S \times A \to \mathbb{R}$ reward function (gain)

Policy $d: S \to A$ decision, not sure the effect.
If I knew the next state: $s \xrightarrow{d(s)=a} s' \xrightarrow{d(s')=a'} s'' \dots$ $\max r(s,a) + r(s',a') + r(s'',a'') + \dots$
problems: unknown next state AND absence of terminal state (∞ reward with any policy)

→ **Discount factor** λ $(0 < \lambda < 1)$: $r(s,a) + \lambda r(s',a') + \lambda^2 r(s'',a'') + \dots$
→ **Value function** $v^d(s)$: expected reward starting from s and following policy d

Optimal policy d^* (you don't know which one) has $v^*(s) = \max_{a \in A} \left[r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a)\, v^*(s') \right]$ (Bellman equation)

$d^*(s) = \arg\max_{a \in A} \left[r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a)\, v^*(s') \right]$ ↳ system of non linear equations for $v^*(s)$

Value iteration algorithm:
$v^* \leftarrow$ rdm initialization // always converge
repeat $v \leftarrow v^*$, for each $s \in S$
$v^*(s) \leftarrow \max_{a \in A} \left[r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a)\, v(s') \right]$
until $\max_{s \in S} |v(s) - v^*(s)| < \varepsilon$, return v^*

NON-COOPERATIVE GAMES

Mechanism (rules): NORMAL-FORM-GAMES → players play simultaneously

N: set of players
$A=\{A_1, A_2 \dots A_n\}$ A_i: set of actions of pl i
$U=\{U_1, U_2 \dots U_n\}$ $U_i: A_1 \times A_2 \times \dots A_n \to R$ utility function

Ex: Rock-Paper-Scissors
$N=\{1,2\}$
$A_i=\{R,P,S\}$

$U_{1,2}$:

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Strategies:

Behavior of players. In principle, a f that returns an action but we cannot bluff: we need distribution

$S_1 = \begin{cases} 1/3\ R \\ 1/3\ P \\ 1/3\ S \end{cases}$ $S_2 = \begin{cases} 1/2\ R \\ 1/2\ P \\ \varnothing\ S \end{cases}$ → Lottery probability distribution over outcomes (expected utility)

$E[U_1(S_1, S_2)] = 0 + \frac{1}{6} - \frac{1}{6} - \frac{1}{6} + 0 + \frac{1}{6} = \varnothing$

A strategy is not good in all the possible situations. Properties of the strategies: (<: strictly dom. ≤: weakly dom)

→ DOMINANCE: given a player i and $a \in A_i$, a is dominated IF $\exists a' \in A_i : U_i(a_i, a_{-i}) \leq U_i(a', a_{-i})\ \forall a_{-i}$
(a_i always worse, for each possible actions made by the opponents).
IF after discarding dominated a it remains one action per player, dominant strategy equilibrium.
Assumption: I know utility of the opponent → Iterated dominance (I can discard further actions assuming what a player will play). Not always dominance can be applied.

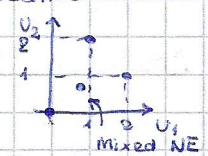
	T	C
T	1,0	2,0
C	2,1	3,4

→ NASH EQUILIBRIUM: (S_1, S_2) is a NE IF $\forall i \in N,\ E[U_i(S_i, S_{-i}) \geq E[U_i(S_i', S_{-i})]\ \forall S_i$
Check that players don't deviate. In mixed strategy players are randomized, otherwise it could be No equilibrium → the opponent understand if you always play the same.

	B	S
B	(2,1)	0,0
S	0,0	(1,2)

Nash proved that in every finite game there is at least a Nash Equilibrium (guaranteed in mixed strategies) and the # of equilibres is odd.

Bach or Stravinsky game



Mixed NE

Player is rational: rdm-zation introduces a constraint

$S_1(B) > 0$, $S_1(S) > 0$ $\Rightarrow E[U_1(B, S_2)] = E[U_1(S, S_2)] \Rightarrow 2 \cdot S_2(B) = 1 \cdot (1 - S_2(B)) \Rightarrow S_2(B) = 1/3$

Nash Equilibrium (and also use dominance) can be inefficient → rational players are selfish.

→ SUBGAME PERFECT EQUILIBRIUM: it is a Nash Equilibrium of every subgame of the original game.
Every finite extensive game has a subgame perfect eq.
↳ it allows to represent the sequence of players' moves and the payoffs, we have:
- perfect info game: every player observes all the actions of the other players;
- imperfect info game: player cannot observe action of another player, we link possible resulting nodes into an information set

Propagate backward in the tree the decisions of the players, substitute an incomplete tree with his Nash equilibrium to find the subgame perfect equilibrium.

NEGOTIATION: agents with diff interests try to reach an agreement without supervisors (peers)

Agents
Issues: objects under negotiation (es: car)
Outcomes: results of negotiation (es: price)
O: set of outcomes (es: all the possible #s)
$U^i: O \to R$: utility of agent i
protocol → rules defining the negotiation (es: low the price if there are more convenient offers)
strategy → individual, how agents act in the protocol (es: how to low it)

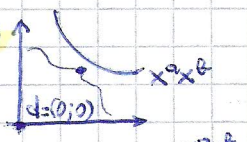
- cooperative (axiomatic) approach: fix some axioms to predict the outcome;
- non-cooperative (algorithm) appr: agents are free, run an algorithm to discover the outcome.

→ Axiomatic approach: agents perfectly rationals (always prefer an outcome that gives higher reward).
$O = A \cup \{D\}$ A: set of agreements D: disagreement $i \in \{A, B\}$ → the two agents
bargaining set $S = \{(U^a(x), U^b(x)) \mid x \in A\}$
$d = (d^a, d^b) = (U^a(D), U^b(D))$ (one element from the domain)
bargaining problem (S, d), bargaining solution $f: (S, d) \mapsto (x^a, x^b) \in S$
S closed and convex: $(x^a, x^b) \in S$ $(y^a, y^b) \in S \to \theta x + (1-\theta) y \in S$ $\theta \in [0,1]$
$S \cap \{(x^a, x^b) \mid x^a \geq d^a,\ x^b \geq d^b\} \neq \{\}$
Properties of f: - individual rationality: $f(S, d) \geq d$
- symmetry: if I swap A and B, I obtain the same point (just swapped)
- pareto optimality: impossible to improve it without decreasing at least one agent

no $y \in S$ s.t. $y \geq f(S,d)$ $\forall i$

- invariance: $U^a = \alpha U^a + \beta$, solution invariant to transformations (just stretched)
- independence of irrelevant alternatives: $\forall S' \subseteq S$ (S' convex and containing the solution) $f(S,d) \in S'$
  $f(S',d) = f(S,d)$

Given this, Nash bargaining solution: $f(S,d) \in \arg\max_{x \in S,\, x \geq d} (x^a - d^a)(x^b - d^b)$ (x^a, x^b)



→ Rubinstein's alternating offer protocol agents $\{a, b\}$ $x^a + x^b = 1$ $\arg\max x^a x^b$ $(x^a, x^b) \in S$

$O = \{(x^a, x^b) \mid 0 \leq x^a, x^b \leq 1,\ x^a + x^b = 1\}$ $t = 1, 2 \ldots$ at each time agent i does an offer

$U^a(x^a, t) = x^a \delta_a^{t-1}$ $0 \leq \delta_a \leq 1$ → discount factor $U^b(x^b, t) = x^b \delta_b^{t-1}$

$\delta_a = 1$: agent A patient
$\delta_a \approx 0$: impatient, after some point utility is $\emptyset$.

We can have deadline or not, without agreement → utility = $\emptyset$.

If negotiation can last forever: $x^a = \frac{1-\delta_b}{1-\delta_a\delta_b}$; $x^b = \frac{\delta_b - \delta_a\delta_b}{1-\delta_a\delta_b}$ $\begin{cases} \delta_a = 1 \rightarrow x^a = 1, x^b = \emptyset \\ \delta_b \approx 0 \end{cases}$

If $\delta_a = \delta_b = 1/2$, $x^a \neq x^b$

n=1 AgA propose (1;0), AgB accepts (oth no agreement)

n=2 AgA propose oth that AgB can accept (≥ to the MAX that agB can obtain at $t=2 \rightarrow \delta$)
↳ $(1-\delta, \delta)$. Following this strategy the first offer is accepted.

For $t < N$ ag i OFFER $(1 - \delta x^j(t+1), \delta x^j(t+1))$ at $t=N$ ag i OFFER $(1;0)$ if it's his turn otherwise accepts the OFFER from j.

The protocol holds when $U^i = x^i \delta_i^{t-1}$ and the resource is divided between i and j.

→ Monotonic concession protocol: the two agents make an offer at the same time

(1) $x^{(a)} \leftarrow \arg\max_{x \in O} U^a(x)$
(2) propose $x^{(a)}$ to agent B
(3) receive proposal $x^{(b)}$ from agB
(4) if $U^a(x^{(b)}) \geq U^a(x^{(a)})$ then accept $x^{(b)}$ and return
else $x^{(a)} \leftarrow x \in O$ s.t. $U^{b_i}(x) \geq U^{b_i}(x^{(a)})$ (and $U^{a_i}(x) \geq 0$)
↓
$U^b(x) = U^b(x^{(a)}) + \varepsilon$
(5) go to (2)

1) easy to verify if the other agent follows the protocol (concedes sth)
2) the protocol converges to a solution but it can be very slow

Not always you know U of the other (to make a concession).

Variant (concede in proportion to how much others are conceding): Zeuthen strategy
(in the else branch) we compute the willingness to risk a breakdown:

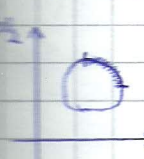
$risk_a = \frac{U^a(x^{(a)}) - U^a(x^{(b)})}{U^a(x^{(a)})}$ $risk_b = \frac{U^b(x^{(b)}) - U^b(x^{(a)})}{U^b(x^{(b)})}$

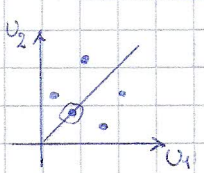
if $risk_a < risk_b$ then $x^{(a)} \leftarrow x \in O$ s.t. $risk_a(x) > risk_b(x)$ and goto(2)
else goto(3)

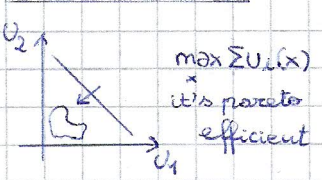
It converges quickly. The solution is a Nash bargaining solution.

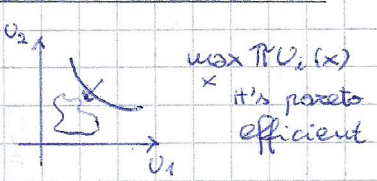
Solution concepts:

- PARETO EFFICIENCY
- EGALITARIAN SOL
- UTILITARIAN SOL
- NASH BARGAINING SOL









COMPUTATIONAL SOCIAL CHOICE

agents $N=\{1,2\dots n\}$ → each one is a voter
alternatives U $|U|\geq 2$
pref relation $\succsim_i$: $a \succsim_i b$ → a is preferred than b by agent i
- transitive: $a\succsim_i b, b\succsim_i c \rightarrow a\succsim_i c$ $a,b,c \in U$
- complete: $a\succsim_i b$ or $b\succsim_i a$ or both

pref profile $R=[\succsim_1, \succsim_2, \dots \succsim_n]$
$\mathcal{R}(U)$: set of all possible pref relation $\#=|U|!$
$\mathcal{R}(U)^n$: " " vectors of pref profile

Vote means aggregate preferences and come up with an order

Aggregation function: social welfare function $f: \mathcal{R}(U)^n \rightarrow \mathcal{R}(U)$ $f([\succsim_1, \succsim_2, \dots \succsim_n]) = \succsim$

1) Pareto optimality: if $a \succ_i b$ $\forall i \in N$ then $a \succ b$ (strict preference)
2) independence of irrelevant alternatives (IIA): changing positions in pref relations, but NOT the order for a and b to obtain a diff final order BUT the same relation btw a, b
   $R|\{a,b\}$: restriction of pref profile applied to a and b, $f(R)=\succsim$
   if $R|\{a,b\} = R'|\{a,b\}$ then $\succsim|\{a,b\} = \succsim'|\{a,b\}$
3) non-dictatorship: $\nexists\, i \in N$ s.t. $a\succsim_i b \rightarrow a \succsim b$ for any R and $a,b \in U$.

Arrow's theorem: there is no f s.t. it satisfies these three properties (IIA usually drops). $|U|\geq 3$

Social choice function $f: \mathcal{R}(U)^n \times \mathcal{F}(U) \rightarrow \mathcal{F}(U)$ → set of all possible subsets of feasible alternatives.
$A \in \mathcal{F}(U)$ $f([\succsim_1, \succsim_2, \dots \succsim_n], A) \rightarrow A'$ $A' \in A$. How to implement it?
→ Voting rule $f: \mathcal{R}(U)^n \rightarrow \mathcal{F}(U)$ f is resolute if $|f(R)|=1$
→ Scoring systems: family of voting rules, score every alternative.
$\langle s_1, s_2 \dots s_m \rangle$ $m=|U|$ $s_1 \geq s_2 \geq \dots \geq s_m$, $s_1 > s_m$
- plurality voting: $\langle 1,0,\dots 0\rangle$
- Borda count: $\langle n-1, n-2, \dots, 1, 0\rangle$

Alternative preferred by most of the agents

Condorcet winner: alternative that wins against each other alternative with pairwise majority
→ Condorcet extension: voting system that selects the condorcet winner (if it exists)
- Copeland's rule: an alt gets one point every time it wins (½ in case of tie) → cond extension

→ Multistep voting systems:
- Plurality with runoff: take the two most voted alternatives from a plurality voting and run a pairwise majority. Choosing how to vote has a large influence on the outcome
- Single Transferable Vote (STV): run n-1 plurality voting, excluding the last alt every time
- Sequential Majority voting: sequence of pairwise majority voting following an Agenda (order) → pairwise maj between the first two, the winner goes against the third alt. ↳ #: $\frac{n!}{2}$

cond Extension ↩

Every agent has the model of preferences of the others: Strategic manipulation of voting to avoid dissatisfaction
Gibbard-Satterthwaite Th: every possible reasoning voting system is prone to manipulation: it's possible to represent a not-true order of preferences in order to have a winner that is better for the voting agent. (easy to estimate pref relation of the others).
→ Voting systems for which manipulation is computationally high (cryptography)
→ Probabilistic voting systems: introduce rdm elements to force the agents to tell the truth.
TR: no voting system that ① avoid manipulation, ② is pareto efficient and ③ not probabilistic dictatorship

AUCTIONS: different agents make a bid, an auctioneer will take a decision according to a mechanism design
Single-good auctions → one single item is auctioned
Multiple-good auction → more tricky, you can submit bids for multiple goods (combinatorial auctions)

Scarce resource to allocate on only one of the agents:
- English: auctioneer fixes an initial price, agents increase it until there won't be an higher offer, then the product will be sold. Problems: winner determination problem and payment.
- Japanese: auctioneer sets an initial price and increase it, meanwhile agents can leave the room. The remaining agent will be the winner and it will pay the actual value.
- Dutch: dual of Japanese → auctioneer starts from a very high price and decreases it until an agent accepts the offer. control of time: you know the maxtime. Only one signal to stop.
- Sealed-bid: every agent makes a bid in a closed envelope. It's a single decision (diff from previous)
  - first price: the auctioneer will choose the agent who submitted the higher bid
  - second price: winner is the larger bid, the agent will pay the 2nd bid. (Vickey)

Assumption: agents have independent private values (evaluation indep. drawn from a distribution, false in real world). Dominant strategy (in second price situation): bid exactly v_i.
You reveal your v_i, this problem is not present in first price.
In English and Japanese auctions the dominant strategy is the same, not true for Dutch or first price auctions → no dominant strategy but equilibrium.

Th: n agents risk-neutral (evaluation without consider risk), independent v_i for each agent, then the equilibrium strategy is: every agent bids $\frac{n-1}{n} v_i$ (you use it if you know the others use it).

If n agents risk-neutral with independent v_i, interval of values $[\underline{v}, \bar{v}]$ and:
① assign the good to the agent who offers more ② never assign to the agent with $v_i = \underline{v}$
then, every mechanism brings to the auctioneer the same expected revenue.

COMBINATORIAL AUCTIONS

agents $N = \{1, 2 \dots n\}$
goods $G = \{1, 2 \dots m\}$ ← possible subsets of G: 2^m Impossible to represent.
valuation function $v_i: 2^G \to \mathbb{R}$ $i \in N$ $\hat{v}_i$: bid

- Additive: $v_i(S \cup S') = v_i(S) + v_i(S')$ assuming $S \cap S' = \emptyset$. Specify v of single elements
- Complementary: $v_i(S \cup S') > v_i(S) + v_i(S')$ additional reward (es: shoes)
- Substitutive: $v_i(S \cup S') < v_i(S) + v_i(S')$ dual case of the prev

Auctioneer allocates goods to the agents in order to maximize the revenue.
Allocation x: specify which subset of G goes to every agent (partition of G)
X: set of possible allocations ($x \in X$).
Goal of auctioneer: $\max_{x_{S,i}} \sum_{i \in N} \sum_{S \subseteq G} \hat{v}_i(S)\, x_{S,i}$ $x_{S,i}$: binary, if set S is allocated to i

Imposition: $\sum_{S | j \in S} \sum_{i \in N} x_{S,i} \leq 1 \ \forall j \in G$; $\sum_{S \subseteq G} x_{S,i} \leq 1 \ \forall i \in N$ (technical constr); $x_{S,i} \in \{0,1\} \ \forall S \subseteq G \ \forall i \in N$

Source of complexity: n×m variables → for large instances no optimal solution
For every set keep track only of $\bar{v}(S)$: max offer received for S. We add $\bar{v}(S) = \emptyset$ for single items if they are missing. For not single items apply additivity in case of missing.
Branch on items: build a tree of possible allocations, starting from the empty one. At each step I add the bids containing the i-th good but not goods already in the path. You can calculate the utility of each path and take the higher revenue, (keep track of the agent that does each offer $\bar{v}$). To speed up:

$x^* \leftarrow \{\}$
$g^* \leftarrow \emptyset$
branch-on-items(1, {})

branch-on-items(j, x)
if sets S in x contain all goods in G
then if $\sum_{S \in x} \bar{v}(S) > g^*$
then $g^* \leftarrow \sum_{S \in x} \bar{v}(S)$
$x^* \leftarrow x$
return

for sets S s.t. $j \in S$ and $S \cap (\cup_{S' \in x} S') = \{\}$
$x' \leftarrow x + S$
if $\sum_{S \in x'} \bar{v}(S) + \underline{h(x')} > g^*$
then branch-on-items(j', x')

Tree built in a depth-first fashion.
h(x'): estimate of the value I can get from allocating objects not yet in x' (upper bound)
↳ $= \sum_{j \in \text{item not in } x'} \max_{S | j \in S} \frac{\bar{v}(S)}{|S|}$ optimistic estimate

... again on auctions
Utility: $U(x) = RP_i - x$ where RP = reservation price, x: what you pay ($x \leq RP_i$)
Allocation function: f that selects the winner
On first price $U_i = 0$, in second price the winner has $U_i > 0$.
(1stp) ↳ winner decreases the price until the second bid to obtain a NE, all the players should know params on the others → NE is not used to prescribe a strategy.

(2ndp) ↳ Weakly dominant strategy in telling the truth (best you can do independently from the others, you don't need other info)

COALITION FORMATION: cooperative games, each agent has benefit in cooperating and can be bounded in an agreement (constraint)

Transferable Utility Games (class of cooperative games): utility/payoff is given to the coalition, then the team decides how to divide the amount between the members. Utility given to a team is independent from the utility given to other teams (not in football)

$A = \{a_1 \dots a_n\}$ set of agents

$V: 2^A \to R$ characteristic function. The returned value represents how well a subset of agents is performing a task. $V(\{\}) = 0$ $V(C) \geq 0$ $\forall C \in A$ A: largest coalition, grand coalition

V is the value of coalition, not how the agents will divide it.

Outcome $(CS, \vec{x})$ CS: coalition structure → partition of the set of agents (set of coalitions)

$CS = \{C_1 \dots C_n\}$ s.t. $\bigcup_i C_i = A$ $C_i \cap C_j = \{\}$ $i \neq j$

$\vec{x}$: payoff vector → $\vec{x} = (x_1 \dots x_n)$, $x_i \geq 0$ $\forall i$; x_i is the payment for ag i

we assume that $\sum_{i \text{ ag } \in C} x_i = V(C)$

On the family of transferable utility games:

- **super additive game**: $V(C_1 \cup C_2) \geq V(C_1) + V(C_2)$ $\forall C_1, C_2 \in A$ and disjointed. Best CS: gran coalition
- **convex game**: $V(C_1 \cup C_2) + V(C_1 \cap C_2) \geq V(C_1) + V(C_2)$ $\forall C_1, C_2 \in A$

  they are super additive (taken into account for disjointed subsets), more special
- **simple game**: $V(C) \in \{0, 1\}$ $\forall C \in A$

How to find an interesting outcome? Two steps: finding a good coalition structure and a good payoff vector.

Payoff vector is NOT stable if a subset of a coalition has a sum of payoffs that is lower than the payoff of their coalition (unhappiness)

CORE: set of all stable outcomes. $Core(G) = \{(CS, \vec{x}) \mid \sum_{i \text{ ag } \in C} x_i \geq V(C) \ \forall C \subseteq A\}$

Problems: several outcomes in the core seem to be unfair

- $A = \{1,2,3\}$ $V(C) = \begin{cases} 1 & \text{if } |C| > 1 \\ \emptyset & \text{otherwise} \end{cases}$ No possibility to distribute 1 keeping stability. → empty core (not in convex games)

Shapley values: idea is to produce a fair outcome, based on marginal contribution (in super additive games)

permutation $\{a_1, a_2 \dots a_n\} = A$. $n!$ possible permutations $\Pi \in \Pi_A$ — set of all possible permutations

$C_\Pi(a_i)$ → set of agents that appear before a_i in permutation Π

$\Delta^G_\Pi(a_i) = v(C_\Pi(a_i) \cup \{a_i\}) - v(C_\Pi(a_i))$ → marginal contribution

$\varphi_i(G) = \frac{1}{n!} \sum_{\Pi \in \Pi_A} \Delta^G_\Pi(a_i)$ → Shapley value. Payoff vector composed by Shapley values.

CS = A (super additive game)

Properties:

1) **Efficiency**: $\varphi_1(G) + \varphi_2(G) + \dots + \varphi_n(G) = v(A)$, $\vec{x} = (\varphi_1(G), \varphi_2(G), \dots, \varphi_n(G))$
2) **Dummy player**: $v(C \cup \{a\}) = v(C)$ $\forall C \subseteq A$, $\varphi_a(G) = \emptyset$
3) **Symmetric**: $v(C \cup \{a_i\}) = v(C \cup \{a_j\})$ $\forall C \subseteq A - \{a_i, a_j\}$ $\varphi_i(G) = \varphi_j(G)$
4) **Additive**: given $G_1 = (A, v_1)$ and $G_2 = (A, v_2)$, $G = (A, v_1 + v_2)$

$\varphi_i(G_1)$ $\varphi_i(G_2)$ $\varphi_i(G) = \varphi_i(G_1) + \varphi_i(G_2)$ $\vec{x} = (x_1, x_2 \dots x_n)$ $x_1 = \varphi_1(G)$ $x_2 = \varphi_2(G)$

In gran coalition, the only way to satisfying these properties is assign to x_i, $\varphi_i(G)$

In convex games this payoff vector $\vec{x}$ is in the core.

Goodness of CS: $v(CS) = \sum_{C \in CS} v(C)$, $CS \in \mathcal{P}^A$ — set of all possible partitions

In super additive games no CS is strictly better than gran coalition, with high # of agents we have a representation problem (2^n subsets, we cannot sum single values).

Dynamic Programming Algorithm: $\varphi(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\{v(C), \max_{\{C', C''\} \in P^C} \varphi(C') + \varphi(C'')\} & \text{otherwise} \end{cases}$

Looking at the best way of splitting a coalition in exactly two sets, compare the MAX value I obtain with the value of that coalition (remember how I obtained it). Still exponential but ensure to obtain the best solution with reduction of complexity.

Distributed coalition formation - Shehory-Kraus

1. $C_i \leftarrow$ all coalitions including a_i
2. $C_i^* \leftarrow \arg\max_{c \in C_i} \frac{v(c)}{|c|}$ ← same knowledge for each agent
3. broadcast (a_i, C_i^*), add received (a_j, C_j^*) to C^*
4. $C_{max} \leftarrow$ largest C s.t. $\forall a_j \in C, (a_j, C) \in C^*$ ← all the agents in C would like to be part of.
5. if $a_i \in C_{max}$ then join C_{max} and return
6. delete from C_i all the coalitions including agents in C_{max}
7. go to 2.

Not ensured to find the best solution.

MULTIAGENT PLANNING

Calculate a sequence of actions to maximize a global function. In MDP the sequence is implicitly encoded in policy, in STRIPS it is explicitly encoded.
Take in account the long term (a greedy solution could not reach a solution).
Build sequences for each agent, you also have to know what the other agents plan → two sources of complexity: length (temporal) and # of agents.
The state is factored: the effect of an action involves a subset of agents ← Exploited in the algorithms
Ways to intend multiagents:
- centralized algorithm that calculate (multiagent) plan;
- distributed construction of the plan (parallelize it).

Coordination before planning: I decide rules, then every agent builds his plan. The initial coordination ensures that plans are not conflicting (es: driving)
Use social laws: you assume that every agent knows the rules.
Planning before coordination: two diff plans, then solve conflicts before to start the execution (merge of plans). (es: rearrange a schedule to do the same things of a friend in the same time).
Decision-theoretic m.a. planning: plan implicitly represented in the policy
Dynamic m.a. planning: conflict ignored before to start the execution, solved at the moment (when discovered)

Assumptions: n agents in a stochastic environment, they are synchronous in performing action until a finite - infinite horizon. Don't know the state of the environment, reward for the system.

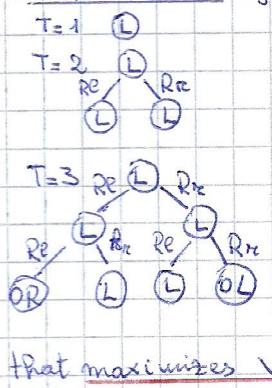
DEC-POMDP (Partially observable MDP, decentralized)

$I = \{1, 2 \dots n\}$ agents
S: states of the world
A_i: action that agent i can perform. $\vec{A} = \bigotimes_{i \in I} A_i$: set of all possible jointed actions (cartesian product)
$P: S \times \vec{A} \to \Delta S$ transition function
Ω_i: observations (set of perceptions of agent i) $\vec{\Omega} = \bigotimes_{i \in I} \Omega_i$

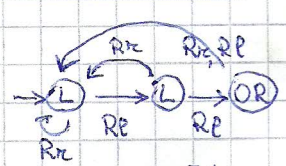
$O: \vec{A} \times S \to \Delta\vec{\Omega}$ observation function: a set of actions brings in a certain state, it gives a probability of making a jointed observation. Indipendent observations: $O(\vec{o} \mid \vec{a}, s) = O_1(o_1 \mid \vec{a}, s) \cdot O_2(o_2 \mid \vec{a}, s)$.
$R: S \times \vec{A} \to \mathbb{R}$ reward function, single function for the system.
Definition of solution is missing (it is in the form of a policy for each agent, s.t. we'd like to maximize sth)
Since the state of the world is NOT observable, the policy is more complex. Map a sequence of observations to actions. How?

policy δ_i

finite horizon T (game lasting T steps)



infinite horizon → FSM with a fixed # of states (3 is the best)



Policies $\vec{q} = \langle q_1, q_2 \rangle$ $V(s_0, \vec{q}) = E\left[\sum_{t=0}^{T-1} R(\vec{a}_t, s_t)\right]$

$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s', \vec{o}} P(s' \mid s, \vec{a}) \cdot O(\vec{o} \mid s', \vec{a}) \cdot V(s', \vec{q}_{\vec{o}})$ (root of $\vec{q}$)

Find a policy that maximizes $V(s_0, \vec{q})$ is NEXP-hard. $V(s', \vec{q}_{\vec{o}})$ is used to calculate the result

DCOP (distributed constraint optimization problem)
Every agents decides independently but the goal is to max a global function (exchange info to obtain it)
Constraints network
$X = \{X_1, X_2, \dots\}$ variables
$D = \{D_1, D_2, \dots\}$ domains (finite)
$C = \{C_1, C_2, \dots\}$ constraints: hard → $C_i \subseteq D_{X_j} \times D_{X_k}$ (a combination is admissible or not)
soft → pay a cost if you choose a combination — utility $F_i : D_{X_j} \times D_{X_k} \to R$
It is possible to write all the constraints as binary constraints, $F_i = -\infty$ simulates an hard constr.

COP: you have a constraint network (represented with constraint graph → nodes are the variables, edges are the constraints) and you find an assignment to the variables in the network s.t. it optimizes the sum of soft constraints: $\sum_{F_i \in C} F_i(X_i, X_j)$

DCOP: every agent controls a subset of variables (only one variable). The agent knows the constraints in which the variable is involved and decides the value to assign. NP-hard
Why use DCOP? - scalability: worst-case complexity is the same (in COP big istances are a problem);
- privacy: an agent needs knowledge on domain and constraints about his variable;
- robustness: it works even without an agent

DPOP (dynamic programming optimization ~~optimization~~ problem → solving algorithm)
Phase 1 - DFS arrangement (preparation, Depth First Search) → build a pseudo-tree starting from a rdm node and visiting depth first (diff trees for diff choices). I'll add pseudo-arcs to represent all the constraints. Induced width: max # of parents and pseudo-parents a node can have the smaller it is, the most efficient the algorithm is.

Phase 2 - Util propagation → flow of the msgs from the leaf, going up. $U_{i \to j}(sep_i) = \max_{x_i} \left(\bigotimes_{x_k \in C_i} U_{k \to i} \otimes \bigotimes_{x_p \in P_i \cup PP_i} F_{i,p} \right)$

$U_{i \to j}$ sent from a child to his real parent.
Sep_i: set of all the agents preceding i in the pseudo-tree who are connected with i or a descendent
C_i: children of ag i P_i PP_i: parents, pseudo-parents of ag i

Phase 3 - Value propagation → flow from the top to the bottom $x_i^* = \arg\max_{x_i} \left(\sum_{A_k \in C_i} U_{k \to i}(x_i, x_j^*) + \sum_{A_j \in P \cup PP_i} F_{i,j}(x_i, x_j^*) \right)$

Max size sent is the max size of a separator.
Algorithm optimal but exponential (in the msg size).

$V_{i \to j} = \{x_i = x_i^*\} \cup \bigcup_{x_s \in sep_i \cap sep_j} \{x_s = x_s^*\}$

MAX-SUM → not guaranteed to provide the optimal solution (could also not converge). Every agent is exchanging iteratively msgs with neighbours. We assume synchronous execution.
At every step each agent sends to all the neighbours:

$$m_{i \to j}^{(x_j)} = \alpha_{ij} + \max_{x_i} \left(F_{ij}(x_i, x_j) + \sum_{A_k \in N_i \setminus \{j\}} m_{k \to i}(x_i) \right)$$

α_{ij}: constant for normalization
F_{ij}: constraint with receiving agent
$\sum m_{k \to i}(x_i)$: sum of msgs from neighb $\neq j$

↳ $\sum_{x_j} m_{i \to j}(x_j) = \emptyset$ for convergence

At the beginning $m_{i \to j}(x_j) = \emptyset$
If all the agents receive the same msgs two times in a row, the algorithm ends.

$z_i(x_i) = \sum_{k \in N_i} m_{k \to i}(x_i)$

$\hat{x}_i = \arg\max_{x_i} z_i(x_i)$

If the starting network is acyclic, max-sum converges to the optimal solution. Otherwise, if converges we have a local maximum.

Size of msgs = $|D|$ but you exchange a lot of msgs.

MULTIAGENT LEARNING

Advantages:
- Robustness: if a single agent fails, the others continue to work;
- Efficiency: divide work between agents, the extra cost in coordinate agents is more than compensated;
- Reconfigurative: local changes in the system in dynamic environments;
- Scalable: easy to add new agents;
- Adaptable: you can use it in diff situations → mechanism of learning.

Challenges in learning:
1) the space of actions is very large: you have to learn how to connect states in every possible combination of actions;
2) Orthogonal problem: when the system receives the credits: how to divide credits among the sequence of actions? how to divide among the single agents?

Schema to solve the problems: objective function for the system and local objective functions for the agents.
Factoredness → alignment between local and global objectives (es: if I max locals, I'm max global. Not always);
Learnability → how much my local objective depends on the action of that agent?
Multiagent Reinforcement Learning: max rewards obtained by some actions. Agents don't know the transition function. Learning with trial and error processes: exploration and exploitation.

MARKOV STOCHASTIC GAMES

S: states
$A = A^1 A^2 \dots A^n$ actions ∀ agent
$r = r^1 r^2 \dots r^n$ rewards function $r^i : S \times A^1 \times A^2 \times \dots \times A^n \to \mathbb{R}$
p = transition function $p : S \times A^1 \times A^2 \times \dots \times A^n \to \Delta S$
} the agents don't know r, p (they learn them)

In a synchronous way, the agents perform an action and receive a reward. The world will go to s'.

$\sum_{s' \in S} p(s' \mid s, a^1, \dots, a^n) = 1$

Strategy $\pi^i : S \to A^i$ Interacting with the world we want to learn sth we don't know (to find π^i).
1. the state of the world encloses the knowledge of all the agents.
2. the policy of the single agent is a path if p is deterministic (in general you have to define an action for every state).

Each agent will max his value function: $V^i(s, \pi^1 \dots \pi^n) = \sum_{t=0}^{\infty} \beta^t E\left[r_t^i \mid \pi^1, \pi^2 \dots \pi^n, s_0 = s\right]$ ($0 \le \beta \le 1$) ← expected because p is probabilistic

↓

Problem: find his own strategy to maximize it. Agent-i has no knowledge about the other strategies, it modifies his strategy in order to max v, same for the others.

Soluzione: NASH EQUILIBRIUM $(\pi^1_*, \pi^2_* \dots \pi^n_*)$ s.t. $V^i(s, \pi^1_* \dots \pi^n_*) \ge V^i(s, \pi^1_* \dots \pi^{i-1}_*, \pi^i_* \dots \pi^n_*)$
$\forall s, \forall i, \forall \pi \in \Pi^i$

Nash-Q value: $Q^i_*(s, a^1, a^2 \dots a^n) = r^i(s, a^1, a^2 \dots a^n) + \beta \sum_{s' \in S} p(s' \mid s, a^1, a^2 \dots a^n) \cdot v^i(s', \pi^1_*, \dots \pi^n_*)$
The first action is arbitrary, from the next one NE is followed

Stage game: normal-form game played by two agents.
Once you have it, you calculate the NE of the game.

	a^2_1	a^2_2
a^1_1	$Q^*_1(s, a^1_1, a^2_1)$ $Q^*_2(s, a^1_1, a^2_1)$	$Q^*_1(s, a^1_1, a^2_2)$ $Q^*_2(s, a^1_1, a^2_2)$
a^1_2	$Q^*_1(s, a^1_2, a^2_1)$ $Q^*_2(s, a^1_2, a^2_1)$	$Q^*_1(s, a^1_2, a^2_2)$ $Q^*_2(s, a^1_2, a^2_2)$

What if agents don't know r, p? They try, knowledge from what happens. Agent perceives the state of the environment, take an action and see:
1) reward I get 2) reward others get 3) actions that other agents took 4) new state

$Q^i_{t+1}(s, a^1 \dots a^n) = (1 - \alpha_t) Q^i_t(s, a^1 \dots a^n) + \alpha_t \left[r^i_t + \beta \, \text{Nash} Q^i_t(s')\right]$ Matrix with $|S| \times |A^1| \times \dots \times |A^n|$ cells

$\alpha_t = 0$ → new value = old value $\alpha_t = 1$ → completely forget the old value. Every agent keeps updated all Q^i.

$\text{Nash} Q^i_t(s')$: payoff that ag i gets in the stage game of s' playing a NE. All the agents choose the same NE between multiples. ↳ I have to build the stage game. $Q^i_{t=0}(\dots) = \varnothing$

$\alpha_t = \frac{1}{n(s, a^1, \dots a^n)}$ → # of times I saw this combination: exploration at the beginning, exploitation later.

Idea: continuing to update, $Q^i_t \to Q^i_*$ (build what you don't know: r, p).
↓ called

Nash-Q learning: it brings me a NE exploring a huge space.