Reactive agents

Assume all states reachable (e.g. network routing) **Purely Reactive Architecture**: Stimulus-response table. No memory, planning over time or model of environment. Conflicting behaviors, dependence on history. Real time **Subsumption architecture** Behaviors may conflict ($\Rightarrow$ turn on/off). Priorities decides who wins. **Decision process**: describes knowledge of rewards & transitions. **MDP**: non-deterministic transitions (may only depend on last state) Discrete space of states, small #actions, **POMDP**: state is uncertain (can be converted to MDP by replacing state by belief function), Need to discretize probabilities, otherwise belief is a vector of continuous probabilities (infinite number of possible beliefs) In such case, value function is piecewise linear ($\max_a \rho(b,a)$, the reward). Init with sample points to construct best policy at extreme points to see if it holds generally. If another policy is better, add it as a new segment, Drop segments that are never the best. Use intersections as discretization intervals. Value function can have unbounded number of segments. Policy is a directed graph (FSC) with nodes a subspace of beliefs associated with action $\alpha(n)$ and arcs associated with observation z and define pmf of successor states. Evaluate policy by constructing value function. Improve by adding nodes for new segments of value function and removing dominated nodes. **Q-learning:** transitions and rewards are not known Simpler than MDP (transitions not needed). **Value:** Potential for rewards from this state onwards. $V(s_i) = R(s_i, a^*) + \gamma \sum_{s' \in S} T(s_i, a^*, s')V(s')$. Alternative: Maximize avg reward $\lim_{h\to\infty} \mathbb{E}[\frac{1}{h}\sum_{t=0}^{h} R(s(t))]$. Note that for MDP, $V(s') = \sum_{s'' \in S} T(s,a,s'')V(s'')$. The **value iteration** algorithm initializes V_i at random, pick best action by trying all, update V_i, stop when $\max_{s\in S} |V'(s) - V(s)| \le \epsilon$ for two consecutive iterations. Converges to the true value: $\max_{s \in S} |V(s) - V*(s)| \le \frac{2\epsilon\gamma}{1-\gamma}$. Simple Slow. The **policy iteration** optimizes over the policy directly by solving for the linear equations of values for the current policy. Faster requires equation solving.

Learning Agents

Q-Learning: *Q-values:* Expected cumulative future reward**Bayesian update(state-less):** Converges to correct values $Q_{t+1}(a) = \alpha r(a) + (1-\alpha)Q_t(a)$ for played a (other Q-vals untouched). $a^* = \arg\max_a Q(s,a)$. For MAvg, $\alpha = \frac{1}{N_t(a)}$.
Cum-Regret: $L_T = \sum_{t=1}^{T} l_t = \max_a \sum_{t=1}^{T} r_t(a) - r_t(\pi)$. For Q values with states, observe (s,a,r,s'): $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$. Strictly improve if $\alpha, \gamma < 1$. α is learning factor, which usually decreases over time. Converges to correct values if sample is representative (all states visited sufficiently often). Not all states easily reachable. Transition cause dependency between successive states. Init rewards to very high values so action selection will pick those not tried a lot. If space is large, slow convergence.
Multi-Armed Bandit: Assume independence (not true for transitions). Objective is to maximize sum of rewards. Naive $\epsilon-$greedy gives $L_T = \mathcal{O}(\epsilon T \max_{a,a'} r(a) - r(a'))$. With $\epsilon = 1/t$, $\mathcal{O}(\log T)$.
Confidence bounds: Set bounds s.t. $\mathbb{P}[LCB(a) < r(a) < UCB(a)] \ge 1-\delta$. *Successive elimination:* Delete action a s.t. $\exists LCB_{a'} > UCB_a$ (delete optimal w.p. $< 2\delta$). *Optimistic:* Pick action with best UCB, i.e. play $\arg\max_a Q_t(a) + u(a)$, where $u(a) = \sqrt{\frac{2\log t}{N_t(a)}}$. For UCB, $L_T = \mathcal{O}(\log T)$. **Regret Matching** Randomness helps against adversarial bandits or collisions Play action w.p. $\propto$ Cum-Regret (the one that we lost more for not playing it in the past) Full observability of rewards. Multiplicative weight update ($P(a) = \frac{(a)}{\sum_{a'} w(a')}$, $w(a)_{t+1} = w(a)_t(1 + \epsilon r_t(a)/B)$) and EXP3 ($w(a)_{t+1} = w(a)_t e^{\epsilon r_t(a)/p_t(a)}$, stronger changes for unlikely actions Only observe played action, partial observability) have $L_T = \mathcal{O}(\sqrt{T})$. For contextual bandits (learning for groups of states) use EXP4.
Experience replay: Randomize order of observations to avoid time dependencies (would destroy SGD).

Deliberative agents

Sets of states and actions can vary and be infinite specially for problems solved once (only small subset visited) **Deliberative architecture**: explicit representation of goal states and plans.
Search algorithms: # states at level $l = \mathcal{O}(\text{branching}^{l+1})$. **BFS:** Always find shortest plan. Not penalized by bad initial desicions Large amounts of memory. **DFS:** Memory grow linearly with depth of search (logarithmic in search space) May be stuck in unpromising branch (can impose depth limit to tackle. Iterative deepening with $\le$ double complexity). Plan found may not be the shortest. *Branch-and-bound:* Keep track of best goal node seen so far in DFS. Ignore nodes with higher cost that the solution (cost of successor can never be lower). **Heuristic search:** First explore most promising solutions, those with $\min\{f(n) := g(n) + h(n)\}$, $g(n)$ cost of best path to node n, $h(n)$ estimate of min cost from node n to a goal node. A* always finds optimal solution if $h(n)$ under-estimates true cost. **Beam search (width n):** Limit queue to only keep first n nodes. With adversary: **minimax**, (we maximize our payoff, adversary minimize it) $\alpha - \beta$ pruning (abandon branch s.t. $\max > \beta, \min < \alpha$, e.g. chess). **MC:** Randomly roll-out sequences of legal moves to end of game. Pick best move with statistics on n roll-outs. **UCT:** Play $\arg\max_i \frac{\text{wins}_i}{N_i} + c\sqrt{\frac{\log N}{N_i}}$ c: exploration parameter. Use it along MC (gives MCTS, used in e.g. Go together with roll-out policy). For games with chance level, play action w.p. $\propto$ counterfactual regret (regret of having played another action instead).

Factored representation

Model each feature by different factors. Represent only those important for current goal. Factor value function into basis $b_i(u)$ (should reflect what are the factors the value depend on): $V(X) = \sum_i b_i w_i$ (find w_i to minimize MSE) Often just few basis functions. Approximation depends on quality of basis function, transitions, policy and reward. Value iteration gives over-determined system. Can solve with least-squares approximation of b_i. A *situation* is a finite sequence of actions. **Situation calculus (STRIPS):** Logic formalism for representing and reasoning, operators that transform situation $S_i \to S_{i+1}$ (preconditions, add-list, delete-list), e.g. predicates AT, LOC, CONN operations CARRY, MOVE. **Least-commitment principle**: If actions can be carried in any order (don't explore $n!$ plans), do parallel actions. **Partial-order planning**: discover which actions can be carried out in parallel. **GraphPlan** builds layers (time) with nodes (proposition and actions) and directed edges (in: preconditions, out: add-list). Solved with backward search. After n levels, all levels of graph will become identical. If goal state not containet or ruled out by mutual-exclusion constraints, problem unsolvable. Built in polynomial time, complete (if no sol in graphplan, no plan exists), but not exhaustive (not all solutions). **Fast Forward:** Ignore delete lists and use graphplan as heuristics for A*. **SAT**: goal state search can be encoded as a CSP. State = vector of state variables (each with a domain), add constraints for: pre and post-conditions for operators, incompatible propositions, exclusion for operators using the same resource. Time is broken up into $2n$ discrete points (states, actions). No solution with neural nets so far. Graphplan as SAT: $S_1 \wedge OP \implies S_2$ translated into $\neg S_1 \vee \neg OP \vee S_2$. **Frame axioms:** Ensure that propositions not affected by actions remain true. Complexity low if many solutions or any, very high if 1 solution.

Multiagent systems

Even if planning centralized (common platform, same data structure), mediated (interact with well-defined messages through central coordinator) or distributed (interact through message exchange) need plans represented in same factorized representation. **Multiagent modalities**: centralized with shared memory, centralized mediator or distributed with message passing, decentralized (no communication, but observe common signals). **Optimization goal**: egalitarian cares about min reward for any agent (all work); social welfare cares about sum of rewards for all agents (can happen that one agent does everything). **Centralized planning**: can explicitly detect conflicts and synergies (plans achieve similar goals). **e.g. blackboard systems** current goals/reward structure, current state, agent's plans. Central point of failure, no concurrency, **vs publish-subscribe system** craete object for potential conflicts and notifies on resource usage, p2p negotiations. Eliminates blackboard systems weaknesses. **Partial-global planning (PGP)**: goal tree in which each agent inserts its partial plans (expressed with predicates). Can discover joint goals and then reorder actions, exchange tasks. **Ontologies:** Shared vocabulary. **Contract nets**: cooperation protocol. Managers divide tasks, contractors place bids, contract is made for lowest bid (no negotiation). Tasks can be subcontracted. Market-based variation: managers increase prices until they obtain a solution. Can handle simultaneous requests (naive version cannot) and solve conflicts. First come first served. Impossible to resolve conflicts. To tackle this, design biddding behaviors and increment until no change (increment is critical, too small means slow, too large can lose solution).

Distributed multiagent systems

Social laws: Common rules followed by agents to avoid conflicts. Must allow to reach goal. **Learning algorithm no-regret** if strategy it learns will eventually have the same performance as the best possible strategy. **Theorem**: under mild conditions (smoothness), agents will converge to equilibrium using no-regret learning. **UCB**: Poor convergence since opponent play is not stationary. **Anti-coordination**: Difficult to learn complementary strategies because many different optima (e.g. resource allocation). **Courteous learning**: Full observability of other's actions. Agents with converged strategy no longer change. Agents without converged will not use conflicting actions. Fast convergence in $\mathcal{O}(n \log n)$. **Distributed contract nets**: make contracts asynchronously and contact agents directly. Agents try to sub-contract tasks with high **marginal cost**, i.e. find tasks that have a high marginal cost and look for other agents with lower marginal cost: for task t that is $c_{add}(A_i, t) = cost(A_i, T \cup t) - cost(A_i, T)$. Pay their cost and assign them the task. Requires knowledge of other agents' capabilities. Incremental bidding can't work, impossible to resolve conflict, bidders must speculate on future tasks. **Distributed CSP**: Constraints are relevant to all agents with variables involved. Solved using **Sync backtracking:** Sequentially generate partial solutions. If it cannot be extended, $k = k-1$, otherwise $k = k+1$. If $k < 1$ unsolvable if $k > n$ solution is current assignment. Improve with *Forward checking:* Send partial solutions to future agents, which initiate backtrack if domain becomes empty. *Dynamic variable ordering:* Forward checking but next variable is the one with smallest domain size. **Async backtracking:** MC search, deliberative agent (estimate cost using random samplng and pick values that seem best) or pseudotrees (build using DFS and back edges from ancestors. No edges between nodes in different branches). For latter, *cost estimation is:* Receive context from ancestors, sample different values for its own variable and forward to descendants. Leaf nodes compute cost (generalization of conflict to cost) and send to direct ancestor. Ancestors form averages of samples and send to its own ancestor. *Value assignment:* root picks optimal value and sends to descendants as value contexts. Descendants pick optimal values depending on context received. **Distributed UCT:** MCTS. Random sampling with UCB in trees. Much faster than systematic search. In backtracking, exponential growth messages (at least one per step), slow message delivery). Better having few large messages. **Dynamic programming** Eliminate variables by replacing them with constraints. Send utilities $u(x = v) = u(x = v, y = (y * | x = v))$ from below. Send value from root afterwards. Linear number of messages (utility and value for each variable). Maximum message size grows exponentially with tree width (product of dimensions of domain of ancestors). **Distributed local search:** Start randomly, make local changes that reduce the number of constraint violations Simple, low complexity. Incomplete. **Min-conflicts:** At each step, find the change in variable assignment that most reduces number of conflicts. Can change values async (accept one at a time inside neighborhood). **Break-out:** As min-conflict but dynamic priority to every conflict (init 1). Can escape local minima.
Sync backtracking: each agent extends the partial solution of the previous, allows use of heuristics forward checking (partial instantiations extended to all future agents) and dynamic variable ordering (select next variable according to domain size - used along forward). **Async backtracking with DUCT**: Represent as pseudo-tree where different branches can be done in parallel. Use distributed Monte Carlo sampling to estimate value assignment's cost. Choose next variable according to Distributed UCT. Orders of magnitude faster than systematic search, all in parallel. exponential number of messages needed. **Distributed DP**: organize agents in a rooted tree + backedges. Send `util` (constraint) messages up, parent decides the best value locally, get `value` messages down. **Distributed local search**: start with random assignment, make local change which most reduces the number of conflicts. Neighborhood = variables connected through constraints. Changes can be async as long as there is only 1 change per neighborhood. **Breakout algorithm**: extension of min-conflict, assigning **dynamic priority** to each constraint. Pick change that reduces most the sum of priorities. All remaining conflicts have priority increased when reaching local minimum, then restart. Termination: when time count is larger than the distance to the furthest agent.

Game theory

Representation of game: Extensive (tree, dotted line if no knowledge about each other choices), normal (matrix). **Pure strategy:** For each state, action is chosen deterministically. **Minimax strategy:** Strategy maximize gains supposing opponent will minimize its losses. Equilibrium. **Minimax Thm:** In zero-sum 2-player games, avg game of A = avg loss of B = value of game, using their best (mixed) minimax strategy resp. Mixed equilibrium (minimax) strategies always exist. **Computing minimax for B:** 1. *Linear Programming:* Find set of p_i^B such that expected gain v of A is minimized and $\forall a_i^A, \sum_{a_j^B} p_j^B R_A(a_i^A, a_j^B) \le v$. 2. *Fictitious play:* Start random, A increases prob. of best response to B's strategy and vice-versa. Converges to optimal. **Preference order axioms:** (i) Completeness: For any pair of outcomes, (ii) Transitivity: $a \succ b, b \succ c \Rightarrow a \succ c$, (iii) Substitutability: $a \equiv b \Rightarrow$ any lottery equally preferred substituting a for b, (iv) Decomposability: If two lotteries assign same prob, equally preferred, (v) Monotonicity: $o_1 \succ o_2, p > q \Rightarrow [p : o_1; (1-p) : o_2] \succ [q : o_1; (1-q) : o_2]$, (vi) Continuity: $o_1 \succ o_2 \succ o_3$, $\exists p$ s.t. $o_2 \equiv [p : o_1; (1-p) : o_3]$. If (i)-(vi) satisfied, every outcome can be associated with utility $u(o)$ s.t. $u(o_1) > u(o_2) \Leftrightarrow o_1 \succ o_2$ and $u([p_1 : o_1; \ldots, p_k : o_k]) = \sum_{i=1}^{k} p_i u(o_i)$. **Eliciting utility fn:** Use axiom (vi): $p_j u(o_j) + (1-p_j)u(o_{j+2}) = u(o_{j+1})$. Usually set $u(o_1) = 1, u(o_k) = 0$. **General sum games:** B not necessarily play as to min $v_i(A)$, so cannot minimize latter with Linear Program: Linear Complementarity Problem (use Lemke's method). **Nash equilibria:** Eliminating weakly dominated strategies can reduce set of NE but never eliminates all NE. For n players, search combinations of n support sets. NE exists but not necessarily unique or minimax. **Stackelberg games:** Leader and follower. If leader informs follower, SE (Stack. equilibrium) very different from NE. Leader can usually force higher payoff (follower has no choice) Convergence only guaranteed for zero-sum games.

Real-world games

Uncertain utilities: Probability distribution of all agents is known. *Ex-ante:* No knowledge about any agent types (expected utilities) - Bayes-Nash equilibrium, **Ex-interim:** Own agent type is known, *Ex-post:* Strongest, knowledge of all types (ex-post NE not always exists - one action that is best for any opponents' type). **Mediated equilibrium:** Let mediator choose the best action combination for all agents that act through him (Prisoners Dilemma). **Correlated equilibrium:** Common coordinator (coin flip). Set of strategies, that is optimal for A if B does not deviate from it (Battle of the sexes). **Latent coordinator:** Agent's action is best response to opponents' observed play. There's no coordinator but agents learn. Easier to reach than NE. **No-regret play:** A sequence of plays $\{s^0, \ldots, s^T\}$ is no regret for i if $\sum_{t=0}^{T} u_i(s^t) \ge \max_{x \in s_i} u_i(x, s_{-i}^t)$. **Coarse Correlated eq.:** For all agents induces sequence of no regret plays, i.e. gives p s.t. $\sum_s p(s)u_i(s) \ge \max_{x\in s_i} \sum_s u_i(x, s_{-i})$ (better than best fixed strategy). **PNE⊂MNE⊂CE⊂CCE**. **Price of Anarchy:** Worst-case efficiency loss $\frac{\max_{s\in S} R(s)}{\min_{s \in E} R(s)}$ for equilibria E. **Price of Stability:** Best-case efficiency loss $\frac{\max_{s\in S} R(s)}{\max_{s\in E} R(s)}$. s^* **is** $(\lambda, \mu) - smooth$: $\sum_{i\in\mathcal{A}} r_i(s_i^*, s_{-i}) \ge \lambda R(s^*) - \mu R(s)\ \forall s$. If this holds, price of anarchy $\le \frac{\lambda}{1+\mu}$.
Strategic negotiation: Agent make and accept/reject offers in an unconstrained self-interested manner. *Alternating offers*: Sequence of rounds. Ends when an offer is accepted. Without time-constraints, the agent A that makes the last round's offer can propose anything and B will accept. With time-constraints, utilities reduce by a factor $\delta_{A/B}$. Agent A should bid $\alpha \le 1 - \delta_2$ choice of protocol is not equilibrium, first offer more power.
Axiomatic negotiation: Agents agree on axioms that outcome should satisfy (constrained opt.). *Criteria*: feasibility, Pareto-optimality, individual rationality, independence of sub-optimal alternatives, independence of linear transformations, symmetry. *Nash Bargaining Solution*: Joint plan where agents can get higher payoff than in conflict plan. Maximize product of gain utilities. Agents have interest in proposing best plan for everyone. Without mediator solve with Zeuthen protocol (alternating offers, concessions by agent with most to lose) or Rosenschein protocol (both calculate risk and agent with smallest risk makes concession, otherwise fail). If $risk_j > 1$ then A_j should not make concession. If $risk_j < 1$ then A_j should make concession $risk_j = \frac{u_j(D_j) - u_j(D_i)}{u_j(D_j) - u_j(D_c)}$. Converges to Nash bargaining for 2 players Hard to generalize.

Auctions

Individual rationality (IR): non-negative utility for participants. **Protocols** manipulability due to collusion: open-cry (**Dutch** speculation efficient, reveals only winner info, **English** vulerable to collusion no speculation); sealed-bids (**Discriminatory** speculation one round only, secret info, **Vickrey** vulnerable to collusion no speculation, one round only, secret info). **Revenue equivalence theorem**: all protocols yield equivalent revenue to auctioneer if private values. For risk-averse bidders due to speculation revenue follows Dutch, Discriminatory $\ge$ English, Vickrey. For non-private values auctioneer revenue English $\ge$ Vickrey $\ge$ Dutch $\ge$ Discriminatory. For non-private winner's curse means winner is most optimistic and tends to overestimate and overpay.
McAfee and McMillan: Bid for bidder with valuation t $b(t) = t - \frac{\int_A^t F(x)^{n-1} dx}{F(t)^{n-1}}$ for n bidders with valuations $\sim F(x)$, A lowest valuation gives a NE. For F uniform, $b(t) = \frac{n-1}{n}t$. Valid only if all bidders use this formula.
Multiple units: Unfair. If n units for sale, each agent pays $(n+1)-$th highest bid. Lower revenue but with higher price, some bidders wouldn't bid.
Multi-unit Vickrey: each agent pays the price of the bid it displaced from the set of winning bids. **Double auctions**: sort buy & sell bids in opposing orders, price is last match. M^{th} highest price is incentive-compatible for sellers, if M^{th} is buy bid then not IC for buyers; $(M+1)^{st}$ for the buyers. **Impossibility result:** No mechanism satisfies IC, IR, efficiency and budget balance. **McAffee**: price is average of last (sell, buy) bid that can still be matched up. Exclude this combination fromtrading $\implies$ IC. **Mechanisms**: social choice function + payment rule. **Revelation principle**: For any mechanism, there is a truthful mechanism with the same outcome and payments $\implies$ IC, IR, budget-balance (no external subsidies).
Clarke Tax (VCG): Maximize sum of declared valuations. Make agents pay tax for the damage to the other agents. An auction is a VCG mechanism, generalizes Vickrey IR, non-negative Collusion when 2 claim same exaggerated valuation $\implies$ no tax, tax must be wasted, agents lose utility, solutions truly optimal no approx.
Alternative IC: single-peaked median, minimum.

Coalitions

Coalitions: Coalition N stable if $\nexists S \subset N$ and $\exists A \in S$ that has higher utility in S than in N. **Core of a game:** Set of payoff distributions for which the Grand Coalition is stable (often empty). Core non-empty iff $v(N) \ge \sum_{S \subseteq N} \lambda(S)v(S)\ \forall \lambda$ s.t. $\sum_{S: i \in S} \lambda(S) = 1\ \forall i \in N$.
Convex game: $v(S \cup T) \ge v(S) + v(T) - v(S \cap T)\ \forall S, T \subset N$ (supperaditive if it holds for $S \cap T = \emptyset$). Convex $\Rightarrow$ non-empty core. **Carrier of a game**: minimal coalition of agents that completely decide the result. **Shapley value**: Stable payoff distribution. Unique vector giving expected distribution of returns of the game. SV_i = average value of added payoff when added that agent to a sub-coalition (over all orders). Agents not in any carrier coalitions has $SV(a_i) = 0$. **Weighted graph games**: Agents are nodes, edges payoff of coalition. Value of a coalition is the sum of edge weights in the subgraph. $SV_i = w(a_i, a_i) + 0.5 \sum_{e_i : e_i = (a_i, a_j), j \neq i} w(e_i)$. Marginal contribution nets allow hyperedges.

Voting protocols

Properties of voting protocols: *Pareto-Optimality:* If every agent prefers d_i over d_j, d_j cannot be preferred over d_i in social choice. *Monotonicity:* If an agent raises its preference for wining alternative, it remains winner. *Non-imposition:* For each alternative d_i, $\exists$ set of agent preference orders s.t. d_i wins. With monotonicity $\Rightarrow$ pareto opt. *Independence of losing alternatives:* If social choice fn $d_i \succ d_j$, introducing d_l order doesn't change. *Non-dictatorship:* Protocol doesn't always choose alternative preferred by same agent. **Condorcet winner:** Alternative that beats on ties all others in pairwise majority vote. Pareto-optimal, independence of losing alternative, monotonic. Always selected by majority voting. **Majority graph:** Arc $d_i \to d_j$ if $d_i \succ d_j$ for majority. Condorcet has only outgoing edges. If $\nexists$, alternative in winning cycle introduced last wins. **Borda count:** Not independent of losing alternatives. **Slater ranking**: vote between every pair of alternatives, pick the smallest transformation to obtain a majority graph. **Kemeny scores:** For each relation between subsequent choices, count how many voters rank the two choices in opposite way. Kemeny score of joint order (found using branch-and-bound - DFS optimization) = sum of these counts. Winner is order with lowest Kemeny score. **Gibbard-Satterthwaite theorem**: any deterministic voting protocol ($\ge$ 3 alternatives) has one of these properties: dictatorial, non-truthful, or $\exists$ a candidate that cannot win. For 2 alternatives, majority voting satisfy all properties. **Heuristics to find manipulations:** *Plurality:* Vote for most preferred alternative that is within some ϵ of winning. *Sensitive rules:* Rank desired outcome first, order all others in opposite order of preferences.