GOAL PROGRAMMING → find the $x \in \mathbb{R}^n$ to get a particular value from $[f_1(x), \dots, f_m(x)]$

MULTI-AGENT SYSTEMS → each agent can
1) recieve infos from environment
2) act on environment
3) receive reward from environment

} Then they need to cooperate or compete to reach a goal

OPTIMIZATION TYPES

① Local vs Global
- local : gradient methods
- global : space filling curves (effective in low dimensional spaces)

② Deterministic vs Stochastic (eg. SGD or EA) or SIMULATED ANNEALING

③ Single vs. Multi Objective

④ Single vs Multi Agent

⑤ Heuristic vs Metaheuristic
- Heuristic : problem specific methods like greedy
- Metaheuristic : doesn't need domain knowledge, generates solutions typically with stochastic methods [nature inspired algorithms → GA, Ant Colony, Swarm Intelligence]

Swarm Intelligence (SI) → decentralized system of agents, interacting locally with one another and the environment
- emerges intelligence that is unknown from single agent

Multi Objective Optimization

$$\begin{cases} \min \; F(x) = (f_1(x), \dots, f_n(x)) \\ x \in D \end{cases}$$

search space

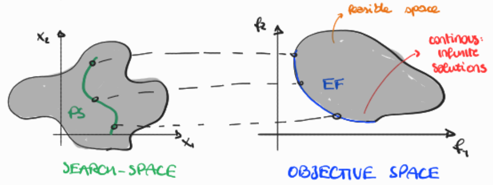
- x può essere continua, discreta o una mixing variables

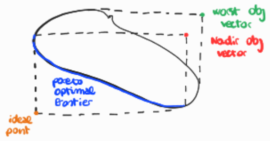
$\vec{x}$ dominates $\vec{y}$ ⟺ 1) $f_i(x) \leq f_i(y) \;\; \forall i$ 2) $\exists j : f_j(x) < f_j(y)$

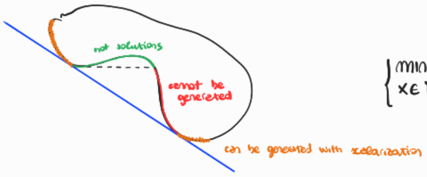
- strongly dominates if $\forall i \;\; f_i(x) < f_i(y)$

$\vec{x}$ is a pareto optimum if it is not dominated by any other solution
- objectives are conflicting, otherwise we have an ideal point $\in D$

PS : pareto set
EF : efficient frontier







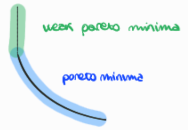
$$\begin{cases} \min \; w^T \cdot F(x) \\ x \in D \end{cases} \quad \text{with } \sum w_i = 1 \wedge w_i \geq 0$$

· can generate all the solutions ⟺ P is convex

$\bigcup_{\alpha \geq 0} S_\alpha = \{\text{Weak minimum}\}$ $(\nexists x \in S : f_i(x) < f_i(x^*) \; \forall i)$

$\bigcup_{\alpha > 0} S_\alpha \subseteq \{\text{Minimum}\}$

equal in the linear case



To obtain the final solution we need to weight the objectives
1) a priori → single obj optimization
2) a posteriori → get the pareto optimal set and the choose the prefered solution
3) interactive → providing feed-backs and zone to focus (from humans or AI)

Objectives of the optimizers
1) find solutions as close as possible to EF
2) try to reach the extremes of EF
3) solutions as uniform distributed as possible

} META-TARGETS

Two algorithms 1) scalarization 2) population based → more solutions per iteration (eg. GA)

GA → avoid problems with non-convex P and generate with a single pass more points of EF
- handles well non-differentiable objectives

GENETIC ALGORITHMS

→ meta-heuristic optimization algorithms, taking inspiration by evolution

1) A solution may be encoded as string, if binary : $\underbrace{\overset{2^4}{0100}}_{x_1}\;\underbrace{\overset{2^6}{010010}}_{x_2}$ we can map them with $x_i = x_i^{min} + \frac{x_i^{max} - x_i^{min}}{2^{e_i} - 1} DV(s_i)$ s_i string int

- cross over operator → select two strings and recombine them in two offsprings by mix together the genetic materials
  - single point cross-over : randomly select a crossing site and exchange all bits on its right
  - uniform cross-over → select a bit from one of the two parents with p probability

  not all strings under goes the cross over, we P_c to model the P to select a string

- mutation operator → randomly mutate some genes at random (probability of P_m)
- Elitist → when the best solution (or all the non dominated ones) can be selected without mutation and crossover to the next generation (will be selected if not outperformed by the offspring)

2) Vector of real numbers $x \in \mathbb{R}^n$ ⇒ perturb with noise, combine $x_i x_n$ with averaging, ...

replace only a few individuals at each generation, more computationally efficient, but more generations needed to converge

1) Steady State + Mutation

- elitist

```
create randomly initial population of pop_size
evaluate their fitness ( obj + penalization if not feasible)
while (not termination condition)
  - choose at random (or ∝ fitness) one individual from curr population
  - mutate and evaluate fitness
  - add to current population
  - remove worst individual
```

2) SS + Cross-Over + Mutation

→ inside the while:

```
- choose at random two individuals
- cross-over to generate two offspring
- mutate both, evaluate
- remove the worst two
```

potential full population replacement in one generation → more aggressive

3) Generational GA + Cross-Over + Mutation

- still elitist

```
t = 0 , create Pt randomly (pop-size), evaluate Pt
while (not finish)
  - create Qt of offspring from Pt of the same size (pop-size)
  - evaluate Qt
  - Rt = Qt ∪ Pt        (2 pop-size)
  - create Pt+1 by removing the worst pop-size from Rt
```

To create Qt :

```
while ( size Qt < pop_size)
    select first parent with BTS
    select second parent with BTS
    generate offsprings
    mutate and evaluate
    add to Qt
```

→ Binary Tournament Selection (if P constrained)

Pick two individuals at random from Pt

- if both are feasible keep the best
- if only one is feasible keep that
- keep the one that violates less the constraints

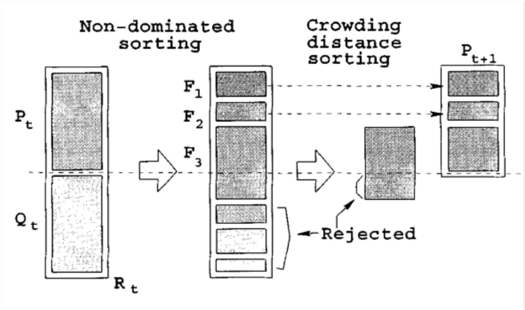
MOGA

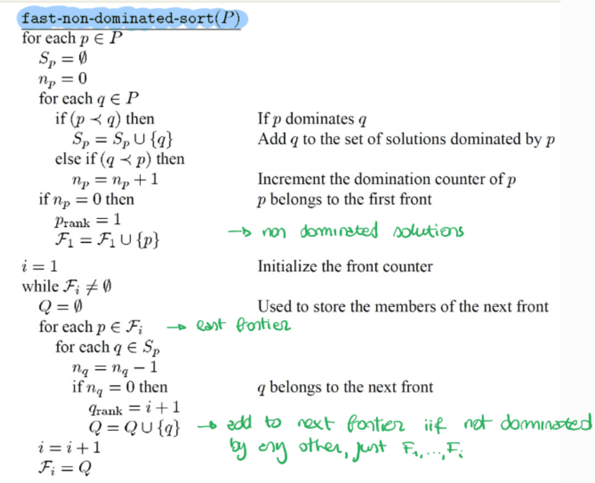
- Family I → pareto dominance based algorithms (NSGA-II, PAES)
- Family II → indicator based algorithms (SMS-EMOA, HyPE, CHEA)
- Family III → decomposition based algorithms (MOEA/D)

NSGA-II (non-dominated sorted GA)

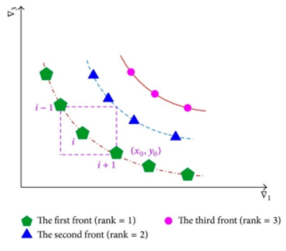
- params: pop_size, n_generation, p_cross, p_mut ↘ no niche size as I
- elitist

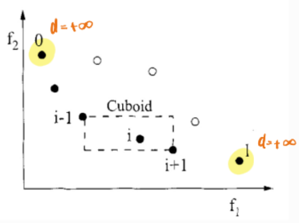
```
P0 ← random initial population of size pop_size
Q0 ← offsprings from P0 of size pop_size
for i=1 to max_gen
    Rt = Pt ∪ Qt
    F ← fast-non dominated sort of Rt        [F = (F1, ...,)]
    i ← 1 , Pt+1 ← {∅}
    while |Pt+1| + |Fi| < pop-size
        Pt+1 = Pt+1 ∪ Fi
        i ← i+1
    compute crowding distance of Fi ⤴ last frontier
    K ← pop_size - |Pt+1|         # elements to be added from Fi
    Pt+1 ← Pt+1 ∪ best K elements in Fi
    Qt ← generate offsprings from Pt+1
```





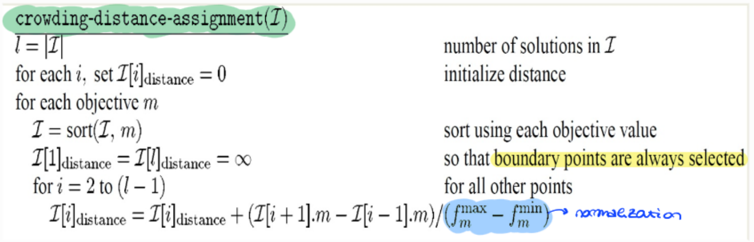
```
fast-non-dominated-sort(P)
for each $p \in P$
   $S_p = \emptyset$
   $n_p = 0$
   for each $q \in P$
      if $(p \prec q)$ then                 If $p$ dominates $q$
         $S_p = S_p \cup \{q\}$              Add $q$ to the set of solutions dominated by $p$
      else if $(q \prec p)$ then
         $n_p = n_p + 1$                    Increment the domination counter of $p$
   if $n_p = 0$ then                        $p$ belongs to the first front
      $p_{\text{rank}} = 1$
      $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$      → non dominated solutions
$i = 1$                                     Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$
   $Q = \emptyset$                          Used to store the members of the next front
   for each $p \in \mathcal{F}_i$   → last frontier
      for each $q \in S_p$
         $n_q = n_q - 1$
         if $n_q = 0$ then                  $q$ belongs to the next front
            $q_{\text{rank}} = i + 1$
            $Q = Q \cup \{q\}$   → add to next frontier iif not dominated by any other, just F1,...,Fi
   $i = i + 1$
   $\mathcal{F}_i = Q$
```





- scelgo quelle con crowding distance maggiore perche' zona meno affollata in quel fronte
  → soluzioni uniformi

- works also on non-convex ES



```
crowding-distance-assignment($\mathcal{I}$)
$l = |\mathcal{I}|$                                            number of solutions in $\mathcal{I}$
for each $i$, set $\mathcal{I}[i]_{\text{distance}} = 0$         initialize distance
for each objective $m$
   $\mathcal{I} = \text{sort}(\mathcal{I}, m)$                    sort using each objective value
   $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$    so that boundary points are always selected
   for $i = 2$ to $(l - 1)$                                  for all other points
      $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$ → normalization
                                                             ↳ equal weight among objectives
```

pareto archived ES

$(\mu-\lambda)$ - PAES → mutation only (evolutionary strategies)

initially only mutation

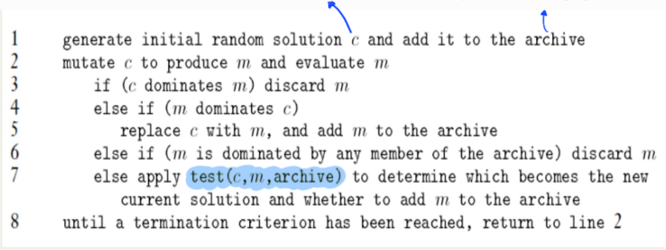
μ : number of current solutions

λ : number of mutants generated from current solutions when $\lambda=1$: steady state algorithm

(1-1) PAES

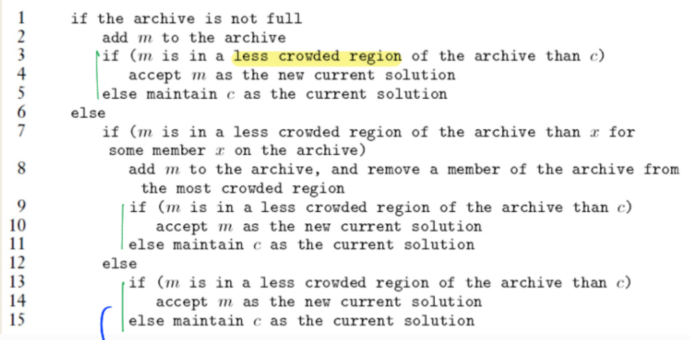
current solution

- past ND solutions
- approximation of current PS



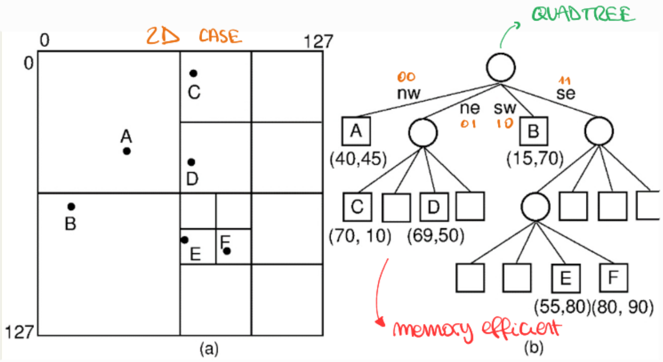
```
1  generate initial random solution c and add it to the archive
2  mutate c to produce m and evaluate m
3      if (c dominates m) discard m
4      else if (m dominates c)
5          replace c with m, and add m to the archive
6      else if (m is dominated by any member of the archive) discard m
7      else apply test(c,m,archive) to determine which becomes the new
           current solution and whether to add m to the archive
8  until a termination criterion has been reached, return to line 2
```

test (c, m, archive)



```
if the archive is not full
    add m to the archive
    if (m is in a less crowded region of the archive than c)
        accept m as the new current solution
    else maintain c as the current solution
else
    if (m is in a less crowded region of the archive than x for
     some member x on the archive)
        add m to the archive, and remove a member of the archive from
         the most crowded region
        if (m is in a less crowded region of the archive than c)
            accept m as the new current solution
        else maintain c as the current solution
    else
        if (m is in a less crowded region of the archive than c)
            accept m as the new current solution
        else maintain c as the current solution
```

m may be the current solution but $\notin$ archive



- Assuming the range is defined for each objective I can repeatedly bisection each objective range

$2^{\ell \times d}$ hyperpercubes ℓ: # of space bisection d: space dimension

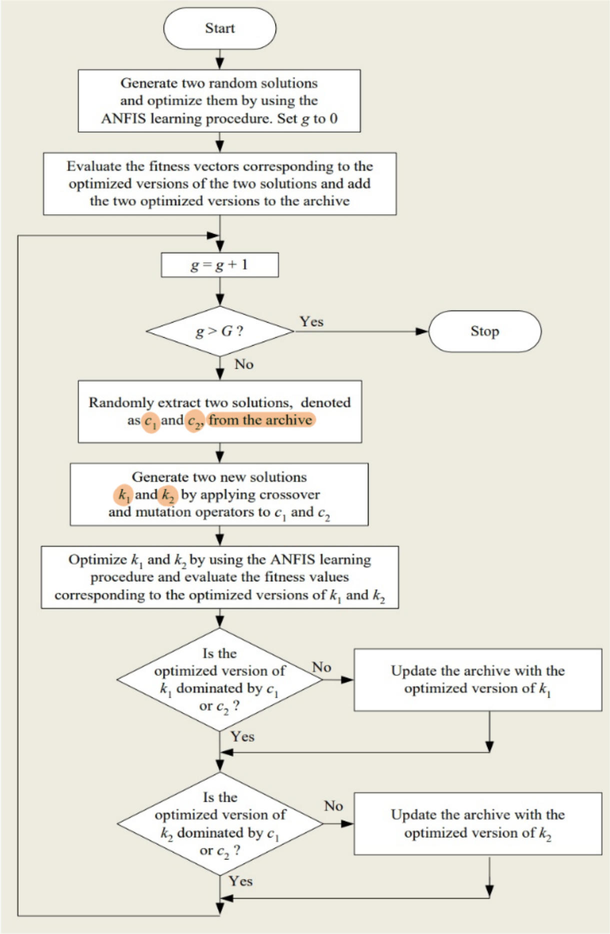
Adaptive method: specify ℓ but not the range ⇒ variable size hyperpercubes

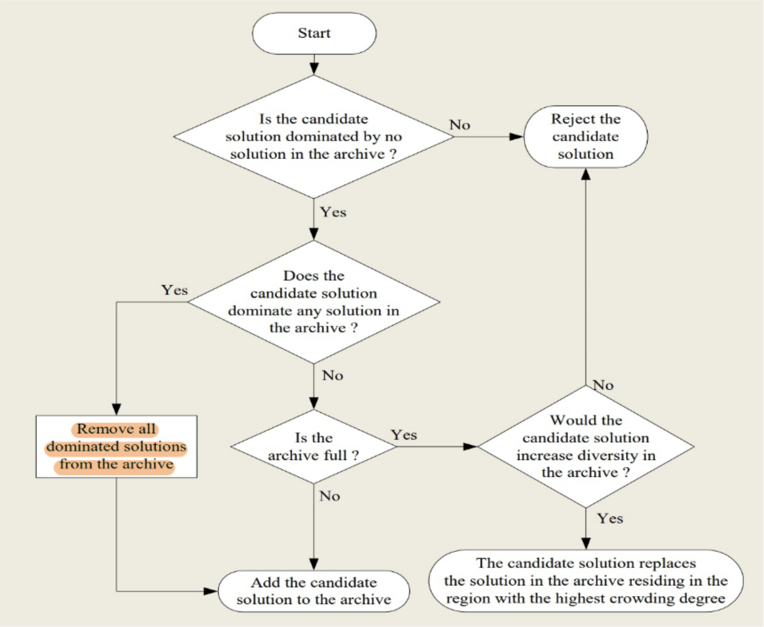
- the range for m objective is the one of current solutions
- recompute the grid ⇔ the Δrange > th

→ In n-D I would have need 2^n split in the tree

- Than concat the binary strings obtained and use it as index for a map containing the crowding number
  - → In case of known obj. space (eg. ROC space) we could use static grids to reduce complexity
- This CD is more efficient than standard one

(2-2) PAES


Start
Generate two random solutions and optimize them by using the ANFIS learning procedure. Set g to 0
Evaluate the fitness vectors corresponding to the optimized versions of the two solutions and add the two optimized versions to the archive
$g = g + 1$
$g > G$?
Yes
Stop
No
Randomly extract two solutions, denoted as c_1 and c_2, from the archive
Generate two new solutions k_1 and k_2 by applying crossover and mutation operators to c_1 and c_2
Optimize k_1 and k_2 by using the ANFIS learning procedure and evaluate the fitness values corresponding to the optimized versions of k_1 and k_2
Is the optimized version of k_1 dominated by c_1 or c_2 ?
No
Update the archive with the optimized version of k_1
Yes
Is the optimized version of k_2 dominated by c_1 or c_2 ?
No
Update the archive with the optimized version of k_2
Yes


Start
Is the candidate solution dominated by no solution in the archive ?
No
Reject the candidate solution
Yes
Does the candidate solution dominate any solution in the archive ?
Yes
Remove all dominated solutions from the archive
No
Is the archive full ?
Yes
No
Would the candidate solution increase diversity in the archive ?
No
Yes
Add the candidate solution to the archive
The candidate solution replaces the solution in the archive residing in the region with the highest crowding degree

SMS-EMOA → S-Metric Selection-based Evolutionary Multi-objective Algorithm

lesbegue integral

$S_z(P) = \Lambda\left(\{q \in \mathbb{R}^n : (\exists p \in P : p \prec q) \wedge (q \prec z)\}\right)$

- Hyper Volume is called S measure → severae definitions (exact or approximate)
- it is an elutist algorithm $S(P_t) \leq S(P_{t+1})$

Maximize the Area between the pareto frontier and the Nadir Point

```
Algorithm 1. SMS-EMOA
$P_0 \leftarrow$ init()                       /* Initialise random population of $\mu$ individuals */
$t \leftarrow 0$
repeat
    $q_{t+1} \leftarrow$ generate($P_t$)  → steady state      /* generate offspring by variation */
    $P_{t+1} \leftarrow$ Reduce($P_t \cup \{q_{t+1}\}$)        /* select $\mu$ best individuals */
    $t \leftarrow t+1$
until termination condition fulfilled
```

Point that if removed would affect S measure less

1)

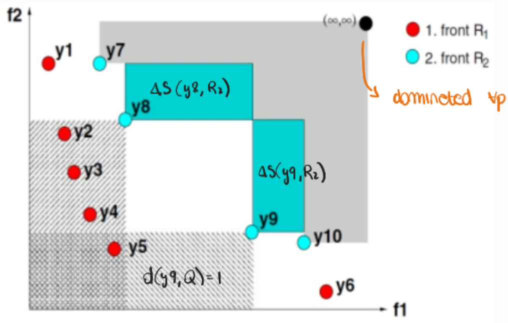
```
Algorithm 2. Reduce(Q)
$\{\mathcal{R}_1, \ldots, \mathcal{R}_v\} \leftarrow$ fast-nondominated-sort($Q$)     /* all $v$ fronts of $Q$ */
$r \leftarrow \text{argmin}_{s \in \mathcal{R}_v}[\Delta_{\mathcal{S}}(s, \mathcal{R}_v)]$   $\Delta S = S(R_v) - S(R_v \setminus \{s\})$   /* $s \in \mathcal{R}_v$ with lowest $\Delta_{\mathcal{S}}(s, \mathcal{R}_v)$ */
return $(Q \setminus \{r\})$  least Point   ⇒ $S(P_{t+1}) \geq S(P_t)$   /* eliminate detected element */
```

2)

```
Algorithm 3. Reduce(Q)
$\{\mathcal{R}_1, \ldots, \mathcal{R}_v\} \leftarrow$ nondominated-sort($Q$)     /* all $v$ fronts of $Q$ */
if $v > 1$ then
    $r \leftarrow \text{argmax}_{s \in \mathcal{R}_v}[d(s, Q)]$              /* $s \in \mathcal{R}_v$ with highest $d(s,Q)$ */
else
    $r \leftarrow \text{argmin}_{s \in \mathcal{R}_1}[\Delta_{\mathcal{S}}(s, \mathcal{R}_1)]$      /* $s \in \mathcal{R}_1$ with lowest $\Delta_{\mathcal{S}}(s, \mathcal{R}_1)$ */
end if
return $(Q \setminus \{r\})$                    /* eliminate detected element */
```

1) more computationally efficient

2) keep points that cover the gaps of prev fronts



dominated $\forall p \in P_t$

$d(s, Q) = |\{y \in Q : y \prec s\}|$ number of point in Q that dominates s

- more computationally efficient ($d(s,Q)$ not usable when $v=1$)

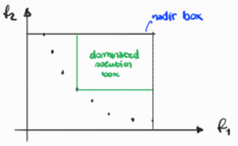
ΔS function $\in O(\mu^3)$

- maximize both the vicinity to the pareto Point and also the sparseness of the solution

→ with first method I would prefer y8 with second y9

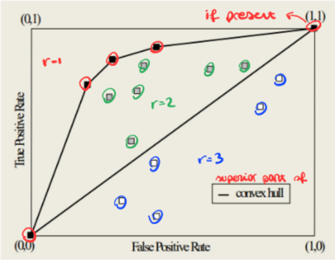
HYPE → Use a montecarlo approach to compute S

- generation of random points inside nadir point box
- compute the ratio between points inside the dominated solution box and the total
- it gives an idea of the importance of the solution



FV-MOEA → it is exact but faster then SMS-EMOA, since to compute $\Delta S(p, Q)$ we need just the neighbour of p, not the all set

CHEA (convex-hull based algorithm)



- convex hull: smallest convex set containing all the binary classifiers

1) Generate random initial population of size Npop, P_0 and evaluate TPR, FPR
   Npop different classifiers
   - create offsprings Q_0 from P_0 (random selection, crossover + mutation) of size Npop

2) Evaluate FPR and TPR $\forall q \in Q_k$
   - build $R_k \leftarrow P_k \cup Q_k$
   - compute the convex hull of $R_k \cup \{(0,0), (1,1)\}$
   may not be present in R_k

3) assign a rank (1,2,3) to each solution and call N_1, N_2, N_3 the # of points with i rank
   rank 1 → points in the convex hull → they are also non-dominated
   rank 2 → other candidate classifiers (above random line)
   rank 3 → non-candidate classifiers

4) Generate P_{k+1}
   - if $N_1 > N_{pop}$ → randomly select, otherwise add all
   - if $N_1 < N_{pop}$ if $N_2 > N_{pop} - N_1$ → randomly select, otherwise insert all
   - randomly extract $N_{pop} - N_1 - N_2$ from rank 3 and insert it, if still missing points

5) if $K = K_{max}$ ⇒ stop, otherwise create Q_{k+1} offspring of size Npop from P_{k+1} using binary tournament based on ranks and apply cross over + mutation

   - $K \leftarrow K+1$ and goto step 2

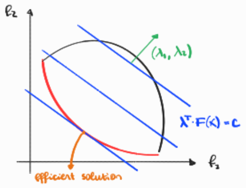
MOEA\D → based on the decomposition into N single-objective sub-problems

- problems are solved collaboratively → best solution found so far by one can affect the best solution found by another one in its neighborhood
- it uses both a population of size N and an unbounded archive of non-dominated solutions found so far → elutist algorithm
  ↳ optional, get more than N solutions

It uses N lambda vectors (or weight vector) provided by the user

→ in this way I will get N solutions ∈ ES

→ WS (weighted sum scalarization) works well in convex problems



weighted sum scalarization

n different sub-problems

$$\begin{cases} \min \; g^{ws}(x, \lambda^i) = \vec{\lambda}^i \cdot \vec{F}(x) = \lambda^i_1 f_1(x) + \ldots \\ \text{where } \sum \lambda^i_j = 1 \;\wedge\; \lambda^i_j \geq 0 \;\; \forall i \end{cases}$$

Chebyshev Scalarization (alternative to WS)

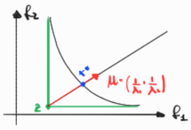
this is not differentiable

each term is continous wrt x (when elided | |)

$$\begin{cases} \min_{x\in\Omega} g^c(x \mid \lambda, z) = \min_{x\in\Omega} \max_{1\le i\le m} \{ \lambda_i \, | f_i(x) - z_i | \} = \min_{x\in\Omega} \| \lambda \circ (f(x) - z) \|_\infty \\ z_i = \min_{x\in\Omega} f_i(x) \\ \sum \lambda_i = 1 \wedge \lambda_i \ge 0 \end{cases}$$

- for any x^* pareto optimal solution, $\exists \lambda$: x^* is optimal for the above problem → · I can find any ideal point
  · works also for non-convex problems

  → $g^c(x \mid z, \lambda)$ is non-smooth wrt x and λ, so gradient method doesn't work



The neighbour of i-th subproblem are the ones with similar weight vectors (that identifies the search direction)

- they have similar obj. functions and with high probability a similar optimal solution

At each generation and sub-problem:

1) Obtain the current solutions from some neighbours (T-1)

2) Generate new solution by applying crossover + mutation from (its own solution and) the borrowed ones

3) - Replace its own solution with the newer one if improve its objective
   - Pass the new solution to some neighbours that will replace if it improves their objective
     ↳ T-1

1) Initialization

- compute L_2 distances between any pair of weight vectors and determine the T closest to each λ_i . $B(i) = \{ i_1, \dots, i_T \}$
  - neighbourhood size
  - contains i
- randomly generate the initial population and evaluate $F(x_i)$ $\forall i \in 1, \dots, N$ (for each individuals)
  ↳ N = pop_size
- initialize z as $z_j = \min_{1\le i\le N} f_j(x_i)$

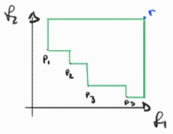
2) UPDATE

For i=1 to N do: N agents

- randomly select e and k in B(i) and generate x_{child} from x_e and x_k with crossover + mutation
- repair with some heuristic if x_{child} violates the constrains
- evaluate $F(x_{child})$
- $\forall j \in 1, \dots, m$ if $z_j > f_j(x_{child})$ update it
- $\forall j \in B(i)$ if $g^c(x_{child} \mid \lambda_j, z) < g^c(x_j \mid \lambda_j, z)$ update it (Neighbour / self update)
- here we could mantain an archive of non-dominated solutions

3) STOPPING CRITEREON → if stop criterion satisfied return $(\{x_1, \dots, x_N\}, \{F(x_1), \dots, F(x_n)\})$ else step 2

MOEA performance comparison → can be done in different ways



1) Hyper-Volume → exactly or by using montecarlo methods (we need a reference nadir point)

2) GD - generational distance

avg distance of A to Z

$$GD(A) = \frac{1}{n} \left(\sum_{i=1}^{n} d_i^p \right)^{\frac{1}{p}}$$

↳ L_2 distance between a_i and nearest $z_j \in Z$

where $A = \{a_1, \dots, a_n\}$ non-dominated objective vector set
$Z = \{z_1, \dots, z_m\}$ the reference set
$p \in \mathbb{N}$

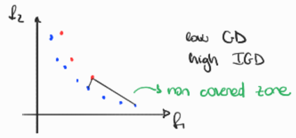
3) IGD - inverted generational distance

$$IGD(A) = \frac{1}{m} \left(\sum_{i=1}^{m} d_i^p \right)^{\frac{1}{p}}$$

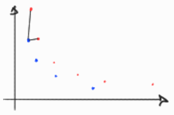
↳ L_2 distance from z_i and NN in A

· how well-spreaded are the solutions wrt reference points

4) Delta metric $\Delta(A) = \max \{ GD(A), IGD(A) \}$



low GD
high IGD



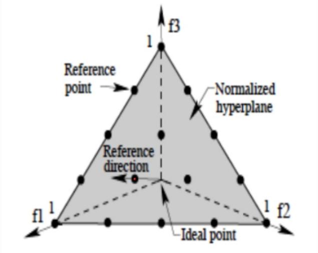
- low IGD
- high GD

MANY OBJECTIVE OPTIMIZATION (from 4 to 30, if > 30 then massive-objective)

Problems:
1) most of the solutions are non-dominated → no solutions to eliminate
2) Difficult to use crowding distance or hypervolume because becomes computationally heavy (and curse of dimensionality)
3) Difficult to visualize the ES and needs exponentialy more points to be covered
4) Ineffective recombination operations since two distant solutions tend to generate an offspring distant from both, not inheriting beneficial characteristics

NSGA-III

· based on a set of reference points provided by the user



$M=3 \quad p=4 \qquad H=\frac{6!}{4!\,2!}=15$

0) Generation of reference points (optional) ↪ just at start: If not provided we can generate them on the normalized hyper-plane

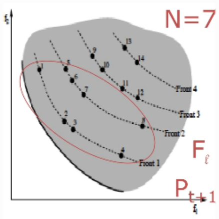
$\#\text{reference points} = \binom{M+p-1}{p}$ ← typically = pop. size

$M \to$ # of obj functions
$p \to$ # of divisions per edge of the simplex

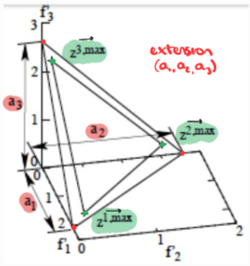
Das-Dennis Algorithm

· each point is associated with a reference (search) direction

1) Identifying the non-dominated frontiers → generation of P_{t+1}



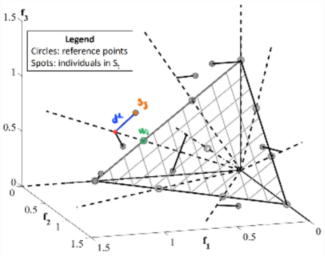
- Parent (P_t) and offsprings (Q_t) ← crossover + mutation are combined into R_t
- non-dominated sorting of R_t
- collect and save fronts from 1 to ℓ to S_t, where ℓ is the first integer that $\sum_{i=1}^{\ell} |F_i| \geq N$ (elitism)
- if $|S_t| = N \rightarrow P_{t+1} = S_t$ (return P_{t+1})
  else $P_{t+1} = \bigcup_{i=1}^{\ell-1} F_i$ then remaining points will be added to P_{t+1} from F_ℓ but not using crowding distance (step 2 and 3)



2) Normalization of population members ($|S_t| \neq N$)

- identify ideal point of S_t ($\vec{z}^{min}$) $\quad z_i^{min} = \underset{s \in S_t}{\text{argmin}}\, f_i(s)$
- translate everything, making the ideal point the origin $\quad f_j'(s) = f_j(s) - z_j^{min}$
- identify extreme points for each objective (nadir points could be calculated) $\quad \vec{z}^{i,max} = f(s^{i,max})$, ← M extreme vectors $\quad s^{i,max} = \underset{s \in S_t}{\text{argmax}}\, f_i(s)$
- generate the M-dimensional hyper-plane with this M extreme vectors, compute $a_j \in \mathbb{R}$ as the intercepts of this plane with the axis
- $f_j''(s) = \frac{f_j(s) - z_j^{min}}{a_j} = \frac{f_j'(s)}{a_j} \quad \forall x \in S_t, \forall j = 1, \ldots, M$
  ↳ can handle different scaled objectives

· the reference points if generated in step 0 are already in this hyperplane, otherwise (provided by user) are projected in Π



3) Association of population members ($|S_t| \neq N$)

Assign each $s_i \in S_t$ to a reference point $\in Z$ using the minimum perpendicular distance

$$d^\perp(s, w) = \left\| f''(s) - \underbrace{w^T f''(s)}_{\|s\| \cdot \|w\| \cdot \cos\theta} \overbrace{\frac{w}{\|w\|^2}}^{\hat{w} \text{ versor}} \right\|_2$$

assign s to $\pi(s) = \underset{w \in Z^r}{\text{argmin}}\, d^\perp(s, w)$ ← reference points set

- create the vector $\vec{\pi}$ in this way:

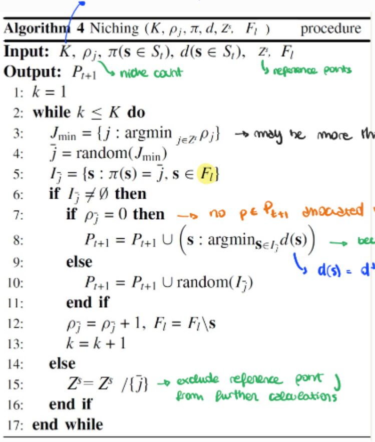
$$\vec{\pi} = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_{|P_{t+1}|} \\ \vdots \\ \pi_{|S_t|} \end{bmatrix} \begin{matrix} \} \, s \in P_{t+1} \\ \} \, s \in F_\ell \end{matrix} \Bigg\} S_t$$

4) Compute Niche Count

$\forall$ reference point $w_i \in Z^r$: $\quad \rho_i = \sum_{s \in \underbrace{S_t \setminus F_\ell}_{\text{current } P_{t+1}}} \mathbb{I}[\pi(s) = w_i]$

5) The Niching Procedure

N − $|P_{t+1}|$ (# of points to add)



```
Algorithm 4 Niching (K, ρ_j, π, d, Z^s, F_l)    procedure
Input: K, ρ_j, π(s ∈ S_t), d(s ∈ S_t), Z^s, F_l
Output: P_{t+1}
k = 1
while k ≤ K do
    J_min = {j : argmin_{j∈Z^s} ρ_j}
    j̄ = random(J_min)
    I_j̄ = {s : π(s) = j̄, s ∈ F_l}
    if I_j̄ ≠ ∅ then
        if ρ_j̄ = 0 then
            P_{t+1} = P_{t+1} ∪ (s : argmin_{s∈I_j̄} d(s))
        else
            P_{t+1} = P_{t+1} ∪ random(I_j̄)
        end if
        ρ_j̄ = ρ_j̄ + 1, F_l = F_l \ s
        k = k + 1
    else
        Z^s = Z^s /{j̄}
    end if
end while
```

ρ_j ↳ niche count

Z^s ↳ reference points

J_{min} → may be more than one

$\rho_{\bar{j}} = 0$ → no $p \in P_{t+1}$ associated with reference $\bar{j}$

$P_{t+1} = P_{t+1} \cup (s : \text{argmin}_{s \in I_{\bar{j}}} d(s))$ → because $\bar{j}$ is under-represented ↳ $d(s) = d^\perp(s, \pi(s))$

$Z^s = Z^s / \{\bar{j}\}$ → exclude reference point j from further calculations

· NSGA II/III no parameters needed, while MOEA\D requires T and $\vec{\lambda}$

· NSGA-III uses niche meanwhile NSGA-II uses crowding distance to select points from F_ℓ
↳ favours better spred since uses information of all P_{t+1} points selected

- with higher M exponentially more points are needed to cover correctly the space

- If there are contrains there is a binary tourmeut selection for generating the mating pool for Q_t (using $CV(p_i)$ ← contrain violation if both unfeasible or random if both feasible)

PURE LEXICOGRAPHIC Multi-Objective OPTIMIZATION

$$\begin{cases} \text{lexmin} & f_1(x), \dots, f_m(x) \\ x \in \Omega \end{cases}$$

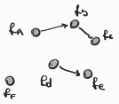
where the importance of minimizing f_1 is infinitely more important than f_2 and so on

→ the lexicographic order allows to compare vectors $[3,4] \leq_{lex} [3,7]$ → defines a total order

vectors $[3,7] \leq_{lex} [2,8]$ ⇔ euclidean scalars $3+7\eta \leq_{eucl} 2+8\eta$

PRIORITY CHAINS (PC)

→ there maybe some priority chains among objectives



$$\min \begin{bmatrix} \text{lexmin}\,(f_1^{(1)}(x), f_1^{(2)}(x), \dots, f_1^{(p_1)}(x)) \\ \text{lexmin}\,(f_2^{(1)}(x), f_2^{(2)}(x), \dots, f_2^{(p_2)}(x)) \\ \vdots \\ \text{lexmin}\,(f_m^{(1)}(x), f_m^{(2)}(x), \dots, f_m^{(p_m)}(x)) \end{bmatrix} \Rightarrow \min \begin{bmatrix} f_1^{(1)}(x) + \text{①}^{-1} f_1^{(2)}(x) + \dots + \text{①}^{1-p_1} f_1^{(p_1)}(x) \\ f_2^{(1)}(x) + \text{①}^{-1} f_2^{(2)}(x) + \dots + \text{①}^{1-p_2} f_2^{(p_2)}(x) \\ \vdots \\ f_m^{(1)}(x) + \text{①}^{-1} f_m^{(2)}(x) + \dots + \text{①}^{1-p_m} f_m^{(p_m)}(x) \end{bmatrix} \triangleq \min \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}$$

Gross Scalar

PAN (polynomial algorithmic number) $\Sigma \cdot S(\alpha)\,\eta^n$

$f_i(x)$ is a chain in the directed graph

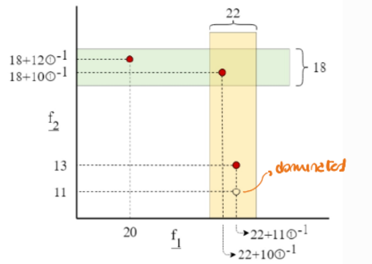
① → grossone number : cardinality of $\mathbb{N}$

①^{-1} → reciprocal of grossone number : smallest number greater than zero

f_1^1 may be the profit

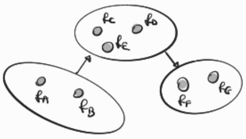
f_1^2 may be the profit from sustainable investments → secondary objective

Pareto lexicographic Dominance (PC) : A dominates B : $A \prec B$ ⇔
$$\begin{cases} f_i(x_A) \leq f_i(x_B) & \forall i = 1, \dots, m \\ \exists j : f_j(x_A) < f_j(x_B) \end{cases}$$



- I could adapt both NGSA-II and MOEA\D Algorithms

PRIORITY LEVELS (PL)



$$\text{lexmin} \left[\min \overset{\text{level 1}}{\begin{pmatrix} f_1^1(x) \\ \vdots \\ f_{m_1}^1(x) \end{pmatrix}}, \dots, \min \overset{\text{level } p}{\begin{pmatrix} f_1^p(x) \\ \vdots \\ f_{m_p}^p(x) \end{pmatrix}} \right] = \min \left[\begin{pmatrix} f_1^1(x) \\ \vdots \\ f_{m_1}^1(x) \end{pmatrix} + \dots + \text{①}^{1-p} \begin{pmatrix} f_1^p(x) \\ \vdots \\ f_{m_p}^p(x) \end{pmatrix} \right]$$

- find pareto non-dominated solutions from L_i and these will be used to search on L_{i+1}

PL Dominance Given A and B belonging to the subfronts $F_{\underline{i}}$ and $F_{\underline{j}}$: $A \prec^* B \iff \underline{i} < \underline{j}$

- SUBFRONTS : we take the fronts set of L_1, for each of them we further split based on the L_2 and recursevely until L_p

F_i front is defined by a gross scalar eg : $\underset{\text{global rank}}{\underline{i}} = 2\text{①}^0 + \underset{\text{lv.2 rank}}{7\text{①}^{-1}} + 3\text{①}^{-2}$ all the solution belonging to the second front of L_1 objectives, 7th front to L_2 objectives ---

problem: a posteriori nature of the definition. To compute $F_{\underline{i}}$ and $F_{\underline{j}}$ I need to find all other individuals

PL-NSGA-II

determines final rank, called with lvl = 0

Algorithm 1 Priority Levels fast non-dominated sort.

```
/* This function is recursive */
/* P is the population, lvl is the current level to consider */
procedure PL_FAST_NONDOMINATED_SORT($P, lvl$)
    /* Base case of recursion */          eg = -2 if 3 levels are present
    if $lvl < min\_lvl$ then return P
    /* The first iteration also initializes all ranks to 0 */
    if $lvl == 0$ then
        for all $p \in P$ do
            $p_{\underline{rank}} = 0$
    /* Ranking within the priority level to determine subfronts */
    $F^{(lvl)} = fast\_nondom\_sort\_in\_level(P, lvl)$ = $[F_{\underline{1}}, \dots]$
    /* Repeat for every subfront found */
    for all $F_{\underline{i}} \in F^{(lvl)}$ do
        /* In the next priority level */
        $F_{\underline{i}} = PL\_fast\_nondominated\_sort(F_{\underline{i}}, lvl\text{-}1)$
return $F^{(level)}$
```

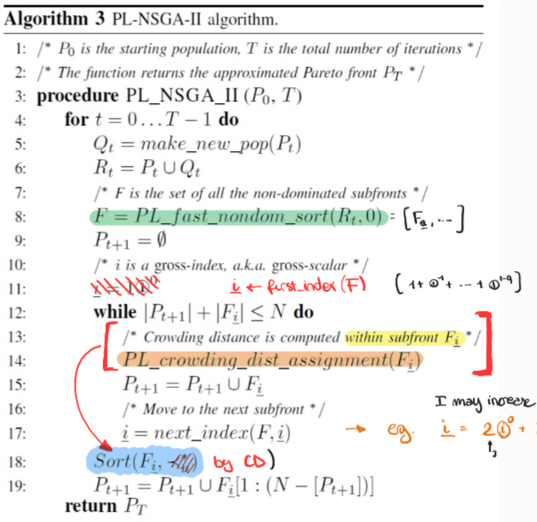
```
procedure FAST_NONDOM_SORT_IN_LEVEL($P, lvl$)
    for all $p \in P$ do
        $S_p = \emptyset$        → points that p dominates
        $n_p = 0$        → # of points that dominate p
        for all $q \in P$ do
            /* non-dominance at specified priority level */
            if $p \prec^{lvl} q$ then
                $S_p = S_p \cup \{q\}$
            else if $q \prec^{lvl} p$ then
                $n_p = n_p + 1$
        if $n_p == 0$ then
            /* First subfront within P */
            $p_{\underline{rank}} = p_{\underline{rank}} + \text{①}^{lvl}$
            $F_{\underline{i}} = F_{\underline{i}} \cup \{p\}$        first sub-front set
    /* Start from the first subfront */
    $\underline{i} = \text{①}^{lvl}$
    while $F_{\underline{i}} \neq \emptyset$ do        i-th subfront set accumulator
        $Q = \emptyset$
        for all $p \in F_{\underline{i}}$ do
            for all $q \in S_p$ do
                $n_q = n_q - 1$        → because we removed p
                if $n_q == 0$ then
                    /* q belongs to the i–th subfront */
                    $q_{\underline{rank}} = q_{\underline{rank}} + \underline{i} + \text{①}^{lvl}$        next sub front
                    $Q = Q \cup \{q\}$
        /* Index of the next subfront */
        $\underline{i} = \underline{i} + \text{①}^{lvl}$
        $F_{\underline{i}} = Q$
return $[F_{\text{①}^{lvl}}, \dots, F_{\underline{i}}]$
```

$cd = cd_I\,\text{①}^0 + \dots + cd_p\,\text{①}^{1-p}$

Algorithm 2 Priority Levels crowding distance assignment.

```
/* F is a leaf-front in the hierarchy of population fronts partitioning */
procedure PL_CROWDING_DIST_ASSIGNMENT($F$)
    $n = |F|$
    for all $i \in F$ do
        $F[i]_{\underline{dist}} = 0$
    /* For each level of priority q; p is the index of the last level */
    for $q = 1 \dots p$ do
        for $j = 1 \dots m_q$ do
            $F = sort\left(F, f_j^{(q)}\right)$
            /* +Inf means "full-scale" (IEEE 754 standard) */
            $F[1]_{\underline{dist}}$ += +Inf $\text{①}^{1-q}$
            $F[n]_{\underline{dist}}$ += +Inf $\text{①}^{1-q}$
            for $i = 2 \dots (n-1)$ do
                /* PL_crowd_dist_ass. has infinitesimal parts */
                $F[i]_{\underline{dist}} \mathrel{+}= \text{①}^{1-q} \frac{f_j^{(q)}(F[i+1]) - f_j^{(q)}(F[i-1])}{f_j^{(q)max} - f_j^{(q)min}}$
```

→ level 1 rank will have $\underline{i} \in \mathbb{N}$



Algorithm 3 PL-NSGA-II algorithm.

```
/* $P_0$ is the starting population, $T$ is the total number of iterations */
/* The function returns the approximated Pareto front $P_T$ */
procedure PL_NSGA_II ($P_0$, $T$)
    for $t = 0 \dots T-1$ do
        $Q_t = make\_new\_pop(P_t)$
        $R_t = P_t \cup Q_t$
        /* $F$ is the set of all the non-dominated subfronts */
        $F = PL\_fast\_nondom\_sort(R_t, 0)$ = [$F_{\underline{0}}$, ...]
        $P_{t+1} = \emptyset$
        /* $i$ is a gross-index, a.k.a. gross-scalar */
        ~~$\underline{i} = 1①^{0}$~~    $\underline{i} \leftarrow$ first_index(F)    ( 1+ ①^{-1}+ ... + ①^{1-q} )
        while $|P_{t+1}| + |F_{\underline{i}}| \le N$ do
            /* Crowding distance is computed within subfront $F_{\underline{i}}$ */
            $PL\_crowding\_dist\_assignment(F_{\underline{i}})$
            $P_{t+1} = P_{t+1} \cup F_{\underline{i}}$
            /* Move to the next subfront */
            $\underline{i} = next\_index(F, \underline{i})$    → eg. $\underline{i} = 2①^{0} + 3①^{-1} + 7①^{-2}$
        Sort($F_{\underline{i}}$, ~~[illegible]~~ by CD)
        $P_{t+1} = P_{t+1} \cup F_{\underline{i}}[1 : (N - [P_{t+1}])]$
    return $P_T$
```

I may increase in order:

$\underline{i} = 2①^{0} + 3①^{-1} + 7①^{-2}$

$\uparrow_3 \quad \uparrow_2 \quad \uparrow_1$

Real word Application

- Vehicle crash Worthiness $\qquad$ lexmin $\left[\min \begin{pmatrix} mass(x) \\ toe(x) \end{pmatrix}, \min \begin{pmatrix} toe(x) \\ -acc(x) \end{pmatrix} \right]$

I could put -acc(x) if a company wants

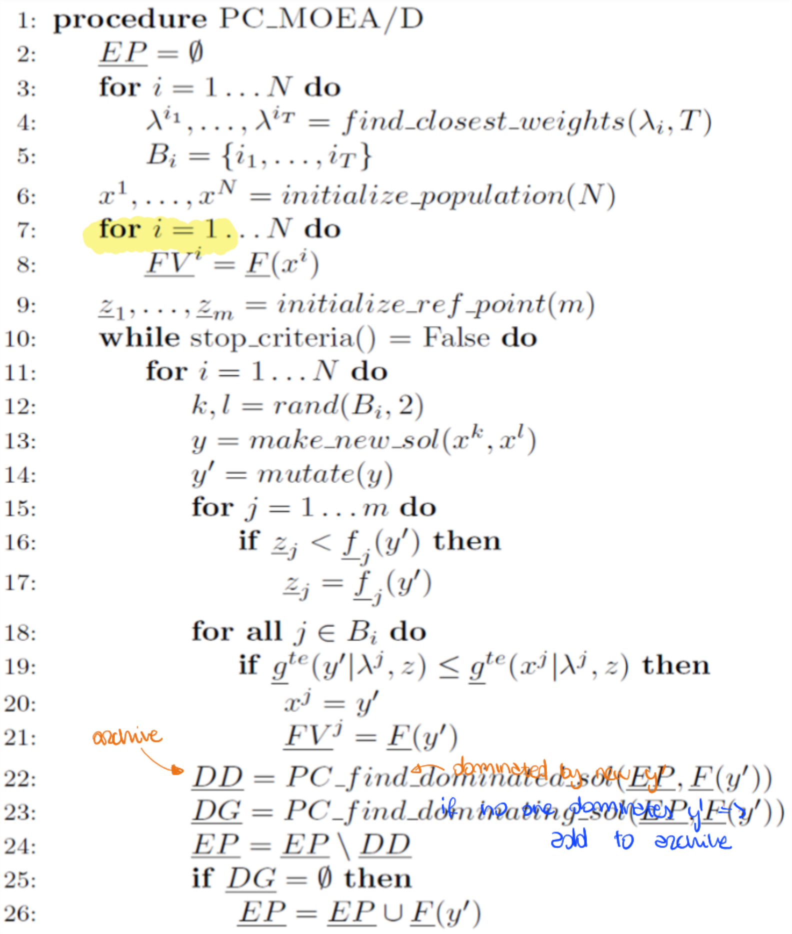
↳ intrusion resistance

- Aircraft design: 10 objectives that could be divided by importance (eg. noise, mass, ...)

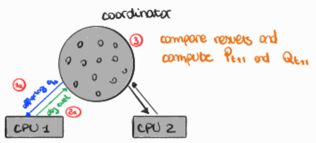
PC-NSGA-II → it is completely identical to NSGA-II but substitute fi with $\underline{fi}$ in CD

$F[1]_{\underline{dist}}$ = ①

PC-MOEA-D



```
procedure PC_MOEA/D
    $\underline{EP} = \emptyset$
    for $i = 1 \dots N$ do
        $\lambda^{i_1}, \dots, \lambda^{i_T} = find\_closest\_weights(\lambda_i, T)$
        $B_i = \{i_1, \dots, i_T\}$
    $x^1, \dots, x^N = initialize\_population(N)$
    for $i = 1 \dots N$ do
        $\underline{FV}^i = \underline{F}(x^i)$
    $\underline{z}_1, \dots, \underline{z}_m = initialize\_ref\_point(m)$
    while stop_criteria() = False do
        for $i = 1 \dots N$ do
            $k, l = rand(B_i, 2)$
            $y = make\_new\_sol(x^k, x^l)$
            $y' = mutate(y)$
            for $j = 1 \dots m$ do
                if $\underline{z}_j < \underline{f}_j(y')$ then
                    $\underline{z}_j = \underline{f}_j(y')$
            for all $j \in B_i$ do
                if $\underline{g}^{te}(y'|\lambda^j, z) \le \underline{g}^{te}(x^j|\lambda^j, z)$ then
                    $x^j = y'$
                    $\underline{FV}^j = \underline{F}(y')$
  archive →
            $\underline{DD} = PC\_find\_dominated\_sol(\underline{EP}, \underline{F}(y'))$    ← dominated by new y'
            $\underline{DG} = PC\_find\_dominating\_sol(\underline{EP}, \underline{F}(y'))$    if no one dominates y' ⇒ add to archive
            $\underline{EP} = \underline{EP} \setminus \underline{DD}$
            if $\underline{DG} = \emptyset$ then
                $\underline{EP} = \underline{EP} \cup \underline{F}(y')$
```



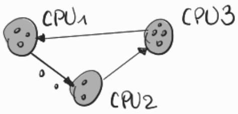
PARALLELIZATION of MOEA

→ typically is really simple to // MOEA

COORDINATOR WORKERS APPROACH

- a coordinator generates the offspring population and distributes the computation to the workers
- the workers compute the solution of the current generation that are collected by the coordinator that will generate the next generation

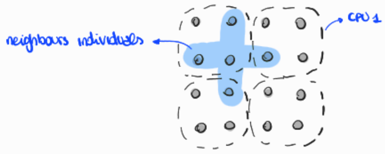
} needs coordination



THE ISLAND MODEL

-decentralized (if a CPU fails we can continue) -population is spreaded

- On each CPU (island) evolve a sub-population independency
- After K generations shares the best solutions with other islands → random or determined migration



THE CELLULAR MODEL

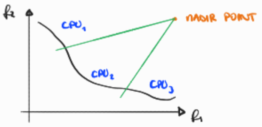
-decentralized -population is spreaded

-The individuals are spatially arranged → solutions of one migrates only to the neighbours (defined by a pre-det. schema)

CONE SEPARATION MEA (cspMEA)

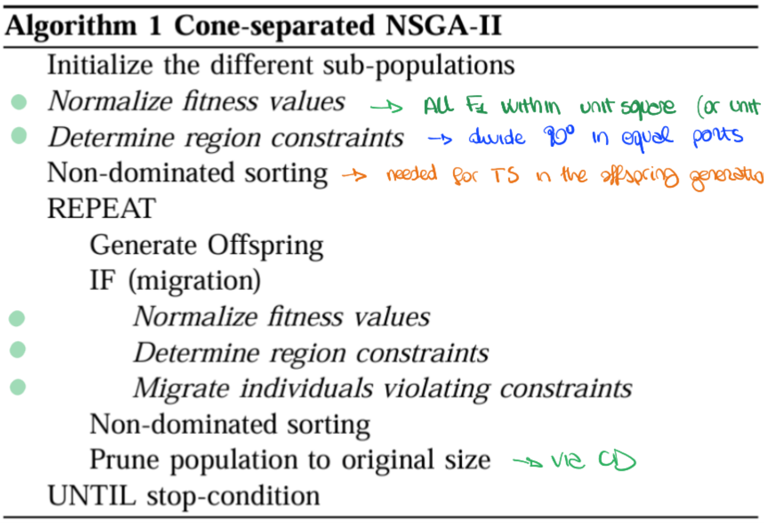
→ is not general purpose as the others, but specific (redefined the NSGA-II algorithm)

· Divide the objective space in n cones n= #CPU
· usable for 2D / 3D problems
- At each population P_t the nadir point change (dynamic)



Typically a master-slave approach (or just a decentralized communication)

● → requires inter-process communication



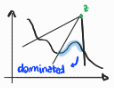
Algorithm 1 Cone-separated NSGA-II

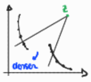
```
  Initialize the different sub-populations
● Normalize fitness values → All Fi within unit square (or unit hypercube) → NADIR POINT
● Determine region constraints → divide 90° in equal parts
  Non-dominated sorting → needed for TS in the offspring generation
  REPEAT
      Generate Offspring
      IF (migration)
●         Normalize fitness values
●         Determine region constraints
●         Migrate individuals violating constraints
      Non-dominated sorting
      Prune population to original size → via CD
  UNTIL stop-condition
```

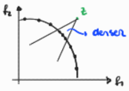
CROWDED TS:
1) compare the two front ranks
2) if equal use crowding distance

- It is better than random individuals assignments since it exploit the search space

- Local pareto front may be dominated by other front (∉ global front)
- The solutions of the global pareto front may be unevely distributed
- The migration rate can be really high





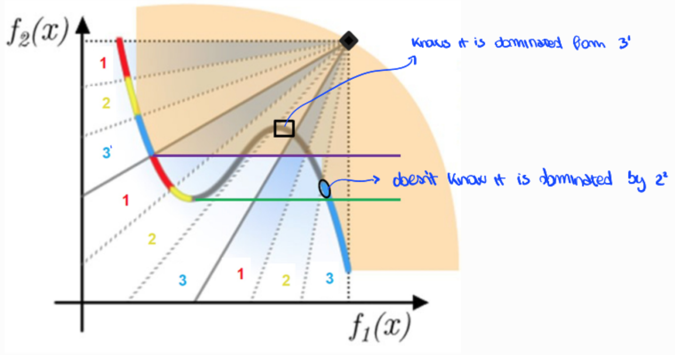


MICRO-CONE SEPARATION PARALLEL MOEA (mspMEA)

→ Do not only devide the unit square into n cones, but each cone is divided in Ngroup microcones → each of them a CPU
- when Ng = 1 → cspMEA ⇒ mspMEA

- drawback → the offspring generated is more likely to fall outside the microcones of the CPU (parents may come from different microcones)
  - thus the contraint is considered as soft instead of hard

- at each generation a fixed number of random individuals violating the soft constraints are migrated to the right CPU
  → allow to control the migration rate
  → Added the binary tournament to favour recombination of the individuals falling inside the microcones assigned to the CPU


$f_2(x)$
1
2
3'
1
2
3
1
2
3
Knows it is dominated from 3'
doesn't know it is dominated by 2'
$f_1(x)$

→ Each CPU has an idea on the global font (glocal front)

1) get R_t and compute global extremes

2) Perform non dominated sort between all microcones of a CPU

3) For last front select iteratevely one point for microcone untie reached pop_size or no more point

   To select the point for each microcone:

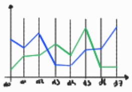
   a) If ∃ points inside ⇒ use crowding distance

   b) If ∄ points inside ⇒ use microcone distance (angular)

4) Once pop_size reached perform the random migration

VISUALIZE DATA in HIGH DIMENSION

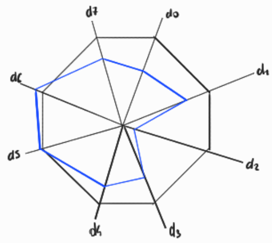
1) Parallel coordinate



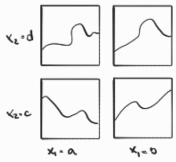
→ Two non-dominated solutions

- If they intersect they are non-dominated

2) Radar Chart

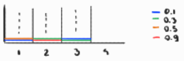


3) Scatter plot matrix → triangular matrix of plots (dim_i vs dim_j)



4) Nested Axes Plot for 4 and 5D → works particularly well if some are are discrete

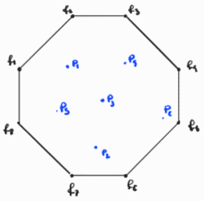
5) Heatmap → similar to // coordinates but instead of lines the points are represented with different coloured segment
  - no dominance can be seen



6) Bubble chart → 4D = 2D + bubble size + colour

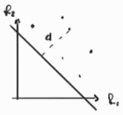
7) Self Organizing Maps (SOM) → NN that lower the dimension from M to 2
  - count how many points fall inside each cell → visualize them in a 2D grid
  - their problems is that close cells could map very distant zone of a manifold if it overlap
    → iSOM was been introduced for efficient sets → near cells maps for near points in the original manifold

8) RadViz → normalize the obj between (0,1], map each obj to the vertex of a polygon
  - the single point position can be understand imagine it connected by all the obj vertexes by a spring



3D RadVis → adds a dimension as the distance of the solution to the hyperplane (1,0,...), (0,1,....), ...

- in this way we understand the dominated/non-dominated solutions



QUANTUM COMPUTING

A qubit is a physical entity (electron, atom, ...) that can be at the same time in two distinct states

→ It is in a superposition of two states

- Decay when measured → it destroys superposition

Ψ function measures the probability the electron is in a specific point in space at a given time

→ follows a PDE called Shrodinger Equation → wave equation

$$\overset{\text{state}}{|\psi\rangle} = \alpha \overset{\text{pure state}}{|0\rangle} + \beta|1\rangle$$

- $\alpha, \beta \in \mathbb{C}$, $\alpha^2 + \beta^2 = 1$
- α^2 is the probability of being in state $|0\rangle$
- Super position can be represented using a sphere

- Quantum Physics can be axiometized

When we have 2 qubits (4 pure states) they can also entangle

→ the two are not independent → some super state is preferred

→ two can prepared H-T, if I see one I immediately know the second, the outcome was decided at start

Different from classical correlation (when two states have common causes and knowing one, reveal the other)

1) Violates local realism (Bell Inequalities) → object influenced only by it's sorroundings
2) instantaneous but infos max at speed of light
3) Mesurement has an effect

Since Shrödinger is a wave equation then exists interfearence between paths

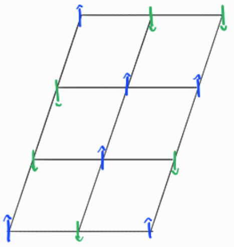
→ eg. Mac-Zender Interferometer or double-slit

→ used in quantum computing to increase the probability of finding the qubits in the correct final state for the problem

Tunneling → when a particle cross an energy barrier even with not sufficient energy (thanks to its wave nature)

1) In optimization can be used to escape local minima (by a thermal jump)
2) In sensing can be used to denoise the signal and reach the true one

ADIABATIC QUANTUM COMPUTERS



→ some NP-hard problems could be solved in polynomial time

Problems are written in terms of the Ising Model (spin ↑ or ↓)

- each possible arrangement is a candidate solution
- total energy of the arrangement → quality of the solution

Condition: the energy needs to change slowly and smoothly → ADIABATIC

- if rush the system could jump in the wrong state

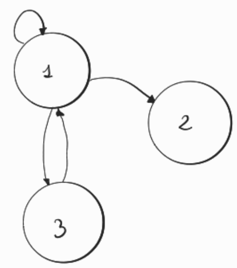
- Quite tolerant to noise wrt other implementations

Problem is solved via Quantum Annealing

1) start via a simple land-scape with obvious global minimum
2) slowly reshape the landscape until the one that encode the problem → quantum tunnelling helps to not get stuck

Applications 1) Quantum NN 2) Quantum Kernel Methods 3) QRL 4) Q Generative Models

MARKOV PROCESS → Stocastic Process that satisfies the markov property



State space $S = \{1,2,3\}$

- markovian property $P(S_t = \bar{s} \mid \bar{S}_{t-1}, \dots, \bar{S}_1) = P(S_t = \bar{s} \mid \bar{S}_{t-1})$
- can be defined a transiction matrix $P \in \mathbb{R}^{|S| \times |S|}$ where $P_{ij} = P(S_t = i \mid S_{t-1} = j) \geq 0$, $\sum_{i=1}^{|S|} P_{ij} = 1 \quad \forall j$
- Starting distibution $s^{(0)} \in \mathbb{R}^{|S|}$ (eg $s^{(0)} = [1,0,0]^T$)

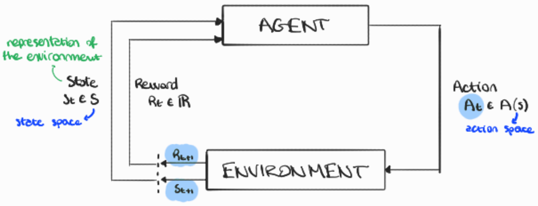
$s^{(t+1)} = P\, s^{(t)}$ → $s^{(t+1)} = P^t s^{(0)}$

- The stationary distribution s^* satisfy: $s^* = P s^*$ ⇒ s^* is an eigenvector of P associated with $\lambda = 1$
  - s^* represents the probability of being in each state in the long run
- Ergodic Theorem: If a chain is reducible (can reach any state from any other) and a-periodic (not locked in cycles)

⇒ 1) $\exists!\, s^*: \; s^* = P s^*$
2) $s^{(t)} \to s^*$ regardless $s^{(0)}$

Implications
1) Sampling long run episodes estimates $\mathbb{E}_{s^*}$
2) Policy gradient exists uniquely $\left[\nabla_\theta J \propto \sum_s d_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(a|s)\, Q(a,s) \right]$ and independently from $s^{(0)}$

RL → branch of ML where an agent learns to take actions within an environment, by maximizing the rewards obtained



$p(s', r \mid s, a) \doteq P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$ — depends only on last state and action

dynamics function $p: S \times R \times S \times A \to [0,1]$

$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1 \quad \forall s, a \in S \times A(s)$

- To satisfy the markov property S must carry all the important historical infos

$p(s' \mid s, a) \doteq P(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in R} p(s', r \mid s, a)$

state transiction function $p: S \times S \times A \to [0,1]$

$r(s,a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \underbrace{\sum_{s' \in S} p(s', r \mid s, a)}_{p(r \mid s, a)}$

reward function $r: S \times A \to \mathbb{R}$

The objective is to maximize the cumulative reward it receives in the long run

simplest one: $G_t \doteq R_{t+1} + R_{t+2} + \dots + R_T$ — final time step (could be finite eg. TicTacToe or $T = +\infty$)

return (must be a function of the reward sequence)

$G_t \doteq R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ — consider absorbing final state $t > T$ with $R_t = 0$

discount rate $\gamma \in [0,1]$

1) R_{t+k} is worth only $\gamma^{k-1} R_{t+k}$ what it would if received immediatly
2) If $\gamma < 1 \wedge \{R_t\}$ bounded ⇒ G_t has a finite value
   - If $\gamma = 0$ the agent is "myopic", trying to maximize only immediate rewords
3) $G_t \doteq R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) = R_{t+1} + \gamma G_{t+1}$

Policy: $\pi(a|s) = P(A_t = a \mid S_t = s)$

If agent is following policy π ⇒ $p(a|s) = \pi(a|s)$

$r(s,\pi) = r_s^\pi = \sum_{a \in A} \pi(a|s)\, r_s^a = \mathbb{E}_\pi[r_s^a]$

$P_{ss'}^\pi = \sum_{a \in A} \pi(a|s)\, P_{ss'}^a = \mathbb{E}_\pi(P_{ss'}^a)$ — $P_{ss'}^a = p(s' \mid s, a)$

Value function: $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{+\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$

$\underbrace{p(a_t, r_{t+1}, s_{t+1}, \dots, s_T, r_T \mid S_t)}_{\text{distribution over trajectories from } s_t} = \prod_{k=0}^{T} \underbrace{\pi(a_{t+k} \mid s_{t+k}) \cdot p(s_{t+k+1}, r_{t+k+1} \mid s_{t+k}, a_{t+k})}_{p(a_{t+k}, s_{t+k+1}, r_{t+k+1} \mid s_{t+k})}$

Expected return when in s and follow the π policy
- how good is to be in that state, it doesn't inform on what action is better

- v_π and q_π can be estimated from experience → Monte Carlo Approximation
- q_π says also what is the best action to take

Action-Value function $q_\pi(a,s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$ — after a, follow π policy

$v_\pi(s) = \mathbb{E}_\pi[q_\pi(s,a)] = \sum_{a \in A} \pi(a|s)\, q_\pi(s,a)$

Markov Decision Process $\text{MDP} = \langle \overset{\text{state space}}{S}, A, p, r, \gamma \rangle$

can be seen as an extension of Markov Chains with a_t and r_t

- model that formalizes decision making in scenarios where outcomes are pontially random and partly under control of a decision maker → MDP is the environment
- follows the markov property
- solving a MDP means to find an optimal policy π^*

Agent $\langle \pi, v_\pi \rangle$ or $\langle \pi, q_\pi \rangle$

- evaluate and improve π under a MDP

POLICY Evaluation → how worth a policy π is

- since v_π is unknown we can keep track of rewards achived through time following π, in this way we can estimate v_π by taking the mean of G_t following a visit to s

total expectation $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}(X \mid A)]$

BELLMAN EQUATIONS

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1}] \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right] = \sum_a \pi(a|s) \left[r(s,a) + \gamma \sum_{s'} p(s' \mid s, a)\, v_\pi(s') \right]$$

$$= \sum_{a \in A} \pi(a|s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right] = R_s^\pi + \sum_{s'} P_{ss'}^\pi v_\pi(s')$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t] = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \overbrace{\sum_{a' \in A} \pi(a' \mid s')\, Q_\pi(s', a')}^{v_\pi(s')}$$

Closed form solution of Bellman Equation

R^π vector with entries r_s^π
P^π matrix with entries $P_{ss'}^\pi$

Can be rewritten in matrix form $\overset{\in \mathbb{R}^{|S|}}{V_\pi} = R^\pi + \gamma \overset{\in \mathbb{R}^{|S|\times|S|}}{P^\pi} V_\pi \Rightarrow V_\pi = (I - \gamma P^\pi)^{-1} R^\pi$

- Intractable to solve in real world scenarios

Optimal Policy A policy π is better then π' (weakly dominates) $(\pi \geq \pi') \iff v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in S$

$V_{\pi^*}(s) = \max_\pi V(s) = V^*(s) \quad \forall s \in S$

- $\forall$MDP exists at least one optimal policy $\pi_* \doteq \arg\max_\pi \mathbb{E}_\pi [G_1 | s_1 \sim \rho(s)] = \arg\max_\pi \mathbb{E}_{s_1 \sim \rho(s)} [V_\pi(s_1)]$

initial state distribution

Optimal Value Functions $v_*(s) \doteq \max_\pi v_\pi(s)$ → largest expected return achivable from each s

$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$

Th: $\forall$MDP $\exists \pi^*(s)$ a deterministic optimal policy: $\overset{\text{greedy}}{\pi^*(a|s)} = \begin{cases} 1 & \text{if } a = \arg\max_{a\in A} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases}$

Bellman Optimality Equation

- $v^*(s) = \max_{a\in A} q^*(s,a) = \overset{\text{non linear}}{\max_{a \in A}} R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v^*(s')$
- $q^*(a,s) = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v^*(s') = R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a \max_{a'\in A} q^*(a',s')$

→ implies determinist optimal policy with max
→ Independent from π

→ These equations are non-linear, so in general $\nexists$ a closed solution V_*
- we need ways to approximate → iteratives approaches

DYNAMIC PROGRAMMING

→ can be used to solve problemes that:

1) can be decomposed in sub-problems → bellman equations
2) can store and reuse subproblems solutions → done by value function (computation of $\mathbb{E}$ return over a trajectory)

- Assume full MDP knowledge and finite MDP (finite A, S)

① Policy Iteration

→ alternates policy evaluation and improvement until convergence → Each iteration is called GPI (generalized policy iteration)

$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \rightarrow \dots \xrightarrow{E} V_{\pi_*} \xrightarrow{I} \pi_*$

1) Policy Evaluation: Input $S, A, R, P, \pi, \varepsilon$

→ I could compute V_π exactly but it is computationally heavy → iterative method

$O(|S|^3)$ vs. $O(|S|^2 \cdot |A|)$ per iteration

```
$V_0(s) = 0 \quad \forall s \in S$
$k = 0$
while $k == 0 \vee \|V_k - V_{k-1}\| > \varepsilon$:
    $\forall s$: $V_{k+1}(s) = \sum_{a\in A} \pi(a|s) \left[ R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a V_k(s') \right] = R_s^{\pi(s)} + \gamma \sum_{s'\in S} P_{ss'}^{\pi(s)} V_k(s')$
    $k{+}{+}$
return $V_k$
```

assumed $\pi(a|s) = \pi(s)$

2) Policy Improvement $\underset{\text{deterministic}}{\pi'(s)} \leftarrow \arg\max_{a\in A} \underbrace{R_s^a + \gamma \sum_{s\in S'} P_{ss'}^a V_k(s')}_{q_\pi(a,s)} \quad \forall s \in S$

computed using old π

- improve just by looking a step-ahead
→ typically V_k takes more to converge wrt π
→ do not need q_π actually

Th: Policy Improvement lets $\pi' \leftarrow \overset{\text{as above}}{\text{greedy}}(v_\pi) \wedge \pi \neq \pi' \Rightarrow V_{\pi'} > V_\pi$

1) $q_\pi(s, \pi'(s)) = \max_{a\in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = V_\pi(s)$

in case of equality $V_\pi(s) = \max_{a\in A} Q_\pi(s,a) \rightarrow V_\pi = V_{\pi'} = V_{\pi^*} \Rightarrow \pi = \pi' = \pi^*$

2) $v_\pi(s_t) \leq q_\pi(s_t, \pi'(s_t)) = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t, a_t = \pi'(s_t)] = \mathbb{E}_{\pi'}[r_{t+1} + \gamma v_\pi(s_{t+1}) | s_t] \leq \mathbb{E}_{\pi'}(r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) | s_t] = \mathbb{E}_{\pi'}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_\pi(s_{t+2}) | s_t] \leq \dots \leq V_{\pi'}(s_t)$

Th: Policy Iteration Convergence

- $|S|$ and $|A|$ must be finite
- R must be bounded
- $0 \leq \gamma < 1$

$\Rightarrow \{V_k\}$ converges to V_* $\wedge$ $\{\pi\} \rightarrow \pi_*$

② Value Iteration

→ do not model $\pi(s)$, computes it only at the end

$V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_*$

- uses Bellman Optimality Equation and not Bellman Equation

$V_{k+1}(s) = \max_{a\in A} R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a V_k(s')$ and then compute $\pi_* \leftarrow \text{greedy}(V_{\pi^*})$

ASYNCRONOUS DP

- some state may not need to have their values updated as often as others
- in place updates of V and π
- Early stopping → do not wait for full convergence

} improve performances

PROBLEMS of DP

- high computational effort, scaning all S and A each iteration $O(|S|^2 \cdot |A|)$ → do not scale with $|S|$
- new approaches must be considered sample-based → model-free, model-based

→ DP method performs bootstrapping and not sampling

MODEL FREE EVALUATION

→ algorithms that do not need to know transitions probabilities $p(s'|s,a)$

- episode → list of experiences made between initial and final state (or max # of experiences T)
- experience$_i$ = < State, Action, Reward, Next_State > → sample

1) MONTE-CARLO RL

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t^T \mid S_t = s\right] \qquad G_t^T = \sum_{k=t}^{T} \gamma^{k-t} r_{k+1}$$

In order to estimate V_π we have two possibilities:

1) First-Encounter: $V_\pi(s)$ is the avg of returns following first visit to s across episodes

$$\hat{V}(s) = \frac{1}{N}\overset{\text{across N episodes}}{\sum_{i=1}^{N}} \underbrace{G_i}_{\text{first encounter return in episode } i} \to V_\pi(s) \quad \text{as } N \to +\infty$$ for large number law

2) Every-Encounter: consider all the visit to s across a single episode

```
Input: π, num_episodes
    $N(s) \leftarrow 0 \quad \forall s \in S$
    $Returns(s) \leftarrow 0 \quad \forall s \in S$
    for $e \leftarrow 1$ to num_episodes do
        generate using π an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$
        $G \leftarrow 0$
        for $t \leftarrow T-1$ to $0$ do:
            $G \leftarrow [\gamma]^* G + R_{t+1}$
            [ if $S_t \notin \langle S_0, \dots, S_{t-1} \rangle$ then ]*  → only present in first-encounter
                $Returns(S_t) \leftarrow Returns(S_t) + G$  → sum across episodes
                $N(S_t) \leftarrow N(S_t) + 1$
    $V(s) \leftarrow \frac{Returns(S_t)}{N(S_t)} \quad \forall s \in S$        [ $G(s_t,a_t) \leftarrow \frac{Returns(s_t,a_t)}{N(s_t,a_t)}$ ] → needs two different counters
    return V                                                              the if is updated with
                                                                          $\langle S_t, a_t \rangle \notin \langle s_0, a_0, \dots, s_{t-1}, a_{t-1} \rangle$
```

- I could use the incremental mean to avoid storing Returns(S_t)

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left[G - V(S_t)\right]$$

- In non-stationary MDP (transitions and rewards changes), V cannot converge and we will use exponential decay to forget about the past $V(S_t) \leftarrow V(S_t) + \underbrace{\alpha}_{\text{fixed regardless } N(S_t) \,\to\, \text{prioritize recent observations}} (G_t - V(S_t))$ $\quad \alpha \in (0,1)$

- Offline learning because wait the end of episode to update V, since I need to compute G_t
- Unbiased since we are computing correctly the expected value
- High Variance wrt TD learning

2) TEMPORAL-DIFFERENCE (TD) LEARN

→ MC methods needs to wait the end of an episode to make an update, meanwhile TD methods only wait the next timestep

TD(0): $V(S_t) \leftarrow V(S_t) + \alpha\left[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{new estimation for } G_t} - V(S_t)\right]$

- Converges to optimality with high probability ⟺ $0 < \alpha \ll 1$
- samples order affects the result a lot
- TD converges faster than MC in practice → online learning

- given π, TD(0) converges to V_π ⟺ 1) π induces an irreducible Markov chain (reach any state from any other)
  2) finite MDP
  3) $\sum_{k=1}^{\infty} \alpha_k = +\infty \wedge \sum_{k=1}^{\infty} \alpha_k^2 < +\infty$ $\quad [\alpha_k = \frac{1}{k}]$

```
Input(π, $\alpha \in (0,1]$, num_ep, γ)
Initialize $V(s) \in \mathbb{R}$ randomly except $V(terminal) = 0$
for $e \leftarrow 1$ to num_ep
    $S_0 \leftarrow$ initial state        $t \leftarrow 0$
    while $S_t \neq$ terminal
        $a_t \leftarrow \pi(S_t)$
        Take action $a_t$ and observe $(S_{t+1}, r_{t+1})$
        $V(S_t) \leftarrow V(S_t) + \alpha[r_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
        $t \leftarrow t+1$
```

- Biased since use bootstrapping (use of estimated values in update) → biased towards the current estimation; meanwhile MC samples

$$V_\pi(s) \doteq \mathbb{E}_\pi\left[\overset{\text{true}}{G_t} \mid S_t = s\right] \quad \leftarrow \text{MC based}$$

$$\overset{!}{=} \mathbb{E}_\pi\left[R_{t+1} + \gamma \underbrace{V_\pi(S_{t+1})}_{\text{approximation}} \mid S_t = s\right] \quad \leftarrow \text{TD based}$$

- Low variance since a lot of updates (shorter stochastic chains)

eg. MC: $\text{Var}\left[G_t \mid S_t\right] = \text{Var}\left[R_{t+1} + \gamma R_{t+2} + \dots\right]$

DIFFERENCES TD vs. MC

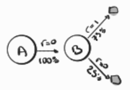
- MC converges to MSE solution $V_\pi = \arg\min_V \sum_{e=1}^{E}\sum_{t=0}^{T_e-1} (G_t - V(s_t))^2$, meanwhile TD(0) to maximum likelihood MDP

because TD(0) assumes and uses the MDP structure in one-step update

$$\begin{cases} \hat{P}^a_{ss'} = \frac{1}{N(s,a)} \sum_{e=1}^{E}\sum_{t=0}^{T_e-1} \mathbb{I}(s_t^e, a_t^e, s_{t+1}^e) \\ \hat{R}^a_s = \frac{1}{N(s,a)} \sum_{e=1}^{E}\sum_{t=0}^{T_e-1} \mathbb{I}(s_t^e, a_t^e)\, r_t^e \end{cases}$$

- MC doesn't respect Markovian Property → update V_π on all the episode

meanwhile TD(0) does → update V_π only on s_t and s_{t+1}



A,0,B,0
B,1
B,1
B,1

B,1
B,1
B,1
B,0

MC: $V_\pi(A) = 0$

TD: $V_\pi(A) = \frac{3}{4}$ → as if I first estimate $\hat{P}^a_{ss'}$ and $\hat{R}^a_s$ and then compute $\mathbb{E}[G_t \mid S_t = A]$

MODEL FREE OPTIMIZATION

ON-POLICY OPTIMIZATION

→ There are two types of policies:

1) Target policy: used to optimize decision making (typically greedy)
2) Behavioural policy: used to navigate the env (could be random or ε-greedy)

↘ In on-policy algorithms the two are equal, meanwhile different in off-policy ones

→ Problems of Greedy approaches for updating the policy in on-policy algorithms: $\pi_{k+1}(a|s) \leftarrow \arg\max_{a\in A} R^a_s + \gamma \sum_{s'\in S} P^a_{ss'} V_{\pi_k}(s')$

1) R^a_s and $P^a_{ss'}$ are unknown → use Q instead of V: $\pi_{k+1}(a|s) = \arg\max_{a\in A} Q_{\pi_k}(s,a)$
2) There is no exploration of the possible actions since only greedy choiches are made
   - ε-greedy select with $p = 1-\varepsilon$ the greedy choiche and with $p=\varepsilon$ a choiche at random
   - Exploration (get infos about the word) - Exploitation (getting rewords) tradeoff

not deterministic

greedy action

$$\pi_{k+1}(a|s) = \begin{cases} \frac{\varepsilon}{|A(s)|} + (1-\varepsilon) & \text{if } a = \arg\max_{a\in A} Q_{\pi_k}(a,s) \\ \frac{\varepsilon}{|A(s)|} & \text{otherwise} \end{cases}$$

$m = |A(s)|$

Th: ε-greedy policy improvement: ε-greedy generates a policy π_{k+1} that $V_{\pi_{k+1}} \geq V_{\pi_k}$

$\max_a Q_{\pi_k}(s,a) \geq \sum_a w_a \cdot Q_{\pi_k}(s,a)$ with $\sum w_a = 1 \wedge w_a \geq 0$

Proof: $\mathbb{E}_{\pi_{k+1}}[Q_{\pi_k}(s,a)|s] = \sum_{a\in A} \pi_{k+1}(a|s)\, Q_{\pi_k}(s,a) = \frac{\varepsilon}{m}\sum_{a\in A} Q_{\pi_k}(s,a) + (1-\varepsilon)\max_{a\in A} Q_{\pi_k}(s,a) \geq \cancel{\frac{\varepsilon}{m}\sum_{a\in A} Q_{\pi_k}(s,a)} + \cancel{(1-\varepsilon)} \sum_{a\in A} \frac{\pi_k(a|s) - \cancel{\frac{\varepsilon}{m}}}{\cancel{1-\varepsilon}} Q_{\pi_k}(s,a) = \sum_{a\in A} \pi_k(a|s)\, Q_{\pi_k}(s,a) = V_{\pi_k}(s)$

- recursively apply this inequality for policy improvement theorem $V_{\pi_{k+1}} \geq V_{\pi_k}$

GLIE PROPERTY (Greedy in the Limith with Infinite Exploration):

π has GLIE property $\overset{\text{sufficient}}{\Longrightarrow}$ Q-style update rule converges (π^*, Q^*)

DEF: a stochastic policy has GLIE property if:

1) $\lim_{k\to+\infty} N_k(s,a) = +\infty$ — each state-action pair is visited infinitely many times
2) $\lim_{k\to+\infty} \pi_k(a|s) = \mathbb{I}(a = \arg\max_{a'\in A} Q_k(s,a'))$ — policy converges to the greedy one → guarantees tha GLIE converges to optimum π^*

→ If ε-greedy has $\varepsilon_k \in \Theta(\frac{1}{k})$ ⇒ it has GLIE property since:

1) $\sum_{k=1}^{\infty} \varepsilon_k = \sum_{k=1}^{\infty} \frac{1}{k} = +\infty$ → infinite exploration
2) since $\varepsilon_k \to 0$, tends to greedy selection

1) ON-POLICY MC

```
Initialize $Q(s,a) \in \mathbb{R}$ arbitrarely $\forall s,a \in S\times A$, $\pi_1(s) \in A(s)$ arbitrarely   [$Q \approx q^*$]
for $e \leftarrow 1$ to max_ep
    generate an episode following $\pi_e$: $s_0, a_0, r_1, \dots$   → $\pi$ both behavioural and target policy
    $G \leftarrow 0$
    for $t \leftarrow T-1$ to $0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $N(s_t,a_t) \leftarrow N(s_t,a_t) + 1$
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \frac{1}{N(s_t,a_t)}[G - Q(s_t,a_t)]$
    $\varepsilon \leftarrow \frac{1}{e}$
    $\pi_e(a|s) \leftarrow \varepsilon$-greedy with Q $\forall a,s$
```

2) SARSA

→ similar to TD evaluation on Q instead of V

- sample$_i$ = $\langle S, A, R, S', A' \rangle$ → transitions of $\langle S, A \rangle$ pairs

Update Rule: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$

→ converges (as all the weighted averages methods) to $q^*(s,a)$

$\Longleftrightarrow \sum_{t=1}^{\infty} \alpha_t = +\infty \wedge \sum_{t=1}^{\infty} \alpha_t^2 < +\infty \wedge$ GLIE policy $\pi_t(a|s)$

just sufficient exploration is needed

```
Input ($\alpha \in (0,1]$, num_ep, $\gamma$)   [estimate $q^*$]
Initialize $Q(s,a) \in \mathbb{R}$ randomly except Q(terminal) = 0
for $e \leftarrow 1$ to num_ep
    $S_0 \leftarrow$ initial state   $t \leftarrow 0$   $a_0 \leftarrow \pi(S_0, Q)$
    while $S_t \neq$ terminal
        Take action $a_t$ and observe $(s_{t+1}, r_{t+1})$
        $a_{t+1} \leftarrow \pi(s_{t+1}, Q)$ → target policy in this iteration and exploratory on the next one   [$\varepsilon$-g]
        $Q(s_t,a_t) \leftarrow Q(s,a) + \alpha(R_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s,a)]$
        $t \leftarrow t+1$
```

- EXPECTED SARSA $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \overbrace{\sum_{a'\in A} \pi(a'|s')\, Q(s',a')}^{\mathbb{E}_\pi[Q(s',a_{t+1})|s']} - Q(s,a)]$

→ fails when |A| is too large to compute

→ prevents the randomness of $a' \sim \pi$ to increase the variance by taking on $\mathbb{E}_{a\sim\pi}$

- has the same bias but lower variance wrt SARSA

↳ more robust updates ⇒ faster convergence

OFF-POLICY OPTIMIZATION

1) MC OPTIMIZATION

Importance Sampling $\mathbb{E}_p[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ with $x_i \sim f(x)$

Law of big numbers

$\frac{1}{N}\sum_{i=1}^{N} f(x_i) = s \xrightarrow{\text{large } N} \mathcal{N}(\mu, \sigma^2) = \begin{cases} \mu = \mathbb{E}_p[f(x)] \\ \sigma^2 = \frac{1}{N} V_p[f(x)] \end{cases}$ → unbiased

random variable since changing sample change s

$$\mathbb{E}_p(f(x)] = \sum_x P(x) f(x) = \sum_x Q(x) \frac{P(x)}{Q(x)} f(x) = \mathbb{E}_q\left[\frac{P(x)}{Q(x)} f(x)\right]$$

p: target policy; q: behavioural policy

- requirements: $p(x) > 0,\ q(x) > 0$ otherwise canot sample from some areas

I can estimate $\mathbb{E}_p$ with samples from $x \sim q$ $\mathbb{E}_q\left(\overset{w(x)}{\frac{P(x)}{Q(x)}} f(x)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{P(x_i)}{Q(x_i)} f(x_i) = r$ → unbiased but different variance $V_q[r] = \frac{1}{N} V_q\left[\frac{P(x)}{Q(x)} f(x)\right]$

There are two samples:
1) G_t^{μ} = return from μ (behavioural)
2) G_t^{π} = return of target policy computed by μ experiences

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + r\left[G_t^{\pi} - Q(s_t, a_t)\right]$$

$P(x_1, \dots, x_n \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid x_1, y) \cdot \dots$ chein rule

$$P(A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi) = \prod_{k=t}^{T-1} \pi(A_k \mid S_k, \text{past})\, p(S_{k+1} \mid S_k, A_k, \text{past})$$

$$= \prod_{k=t}^{T-1} \pi(A_k \mid S_k)\, p(S_{k+1} \mid S_k, A_k)$$ → Markovian property

importance ratio; unknown

$$\Rightarrow \omega_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k)\, \cancel{p(S_{k+1} \mid S_k, A_k)}}{\prod_{k=t}^{T-1} \mu(A_k \mid S_k)\, \cancel{p(S_{k+1} \mid S_k, A_k)}} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{\mu(A_k \mid S_k)}$$

$$\mathbb{E}_\mu\left(\omega_{t:T-1} G_t^{\mu} \mid S_t = s\right] = \mathbb{E}_\pi\left[G_t^{\pi} \mid S_t = s\right] = V_\pi(s)$$

→ unfortunetly ω_t and the episodes increases the variance
- $\omega_t \neq 1$ → when π and μ behave differently
- μ and π should be as close as possible (like ε-greedy and greedy)

```
using μ sample an episode $s_0, a_0, r_1, \dots$
$W_T \leftarrow 1,\ G^{\mu} = G^{\pi} \leftarrow 0$
for $t \leftarrow T-1$ to $0$
    $W_t \leftarrow W_{t+1} \cdot \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)}$    (π: typically deterministic; μ: typically stochastic)
    $G^{\mu} \leftarrow R_{t+1} + \gamma G^{\mu}$
    $G^{\pi} \leftarrow W_t G^{\mu}$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[G^{\pi} - Q(s_t, a_t)\right]$
$\pi(S_t) \leftarrow \arg\max_{a \in A} Q(s_t, a)$
```

2) TD optimization

depends on μ for a (influence only r_{t+1})

reward r_{t+1} just depends on current <a,s> pairs not on all trajectory

$$V(s_t) \leftarrow V(s_t) + v\left[\frac{\pi(a \mid s)}{\mu(a \mid s)}\left(r_{t+1} + \gamma V(s_{t+1})\right) - V(s_t)\right]$$

$$V_\pi(s) = \mathbb{E}_\mu\left[\omega_t \overbrace{\left(r_{t+1} + \gamma G_{t+1}^{\mu}\right)}^{G_t^{\mu}} \mid S_t = s\right]$$

→ scaling by just one factor lower significantly the variance

3) Q-LEARNING

→ do not use importance sampling

- $\pi(s) = \arg\max_{a \in A} Q(a, s)$
- $\mu(a \mid s) = \begin{cases} \frac{\varepsilon}{|A(s)|} + (1-\varepsilon) & \text{if } a = \arg\max_{a \in A} Q_{\pi_k}(a, s) \\ \frac{\varepsilon}{|A(s)|} & \text{otherwise} \end{cases}$

doesn't depend on μ

importance sample is not needed because a_t is given

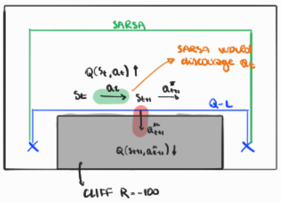
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + v\left[\underline{r_{t+1}^{\mu} + \gamma Q(s_{t+1}, a_{t+1}^{\pi})} - Q(s_t, a_t)\right]$$

$$= r_{t+1}^{\mu} + \gamma Q(s_{t+1}, \arg\max_{a \in A} Q(s_{t+1}, a'))$$

$$= r_{t+1}^{\mu} + \gamma \max_{a \in A} Q(s_{t+1}, a')$$

- optimal convergence with GLIE and $\sum v_t = +\infty \wedge \sum v_t^2 < +\infty$
- Q-learning is a tabular method since I can express Q by a $\mathbb{R}^{|S| \times |A|}$ matrix

SARSA vs Q-Learning (on-policy vs off-policy)



- Q-learning is learning the actual optimal path (greedy) → optimistic
- SARSA is learning the safest path from ε-greedy → more conservatory in dangerous situations
  - it learns a sub-optimal policy

N-STEPS BOOTSTRAPPING

→ n between MC methods with $n \to +\infty$ and TD(0)

$$G_t \doteq \overbrace{R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n R_{t+n+1}}^{n+1 \text{ rewards}} + \underbrace{\gamma^{n+1} R_{t+n+2} + \dots + \gamma^{T-t-1} R_T}_{\gamma^{n+1} V(S_{t+n+1}) \quad \text{bootstrapping}}$$

1) TD(n)

$$G_t^{(n)} \doteq \overbrace{\sum_{k=0}^{n-1} \gamma^k R_{t+k+1}}^{n \text{ rewards}} + \gamma^k V(S_{t+n})$$

↳ will be the one of $t+n$ steps ahead

$$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

→ typically the best result is in between $n=1$ and $n=+\infty$

- The first time R_{t+n+1} is available is n timesteps forward (online but delay)

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^{(n)} - V(S_t) \right]$$

2) TD(λ)

→ weighted mean of $G_t^{(n)}$ for different n

→ offline

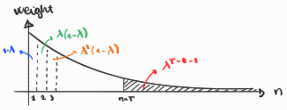
$$G_t^{(\lambda)} = (1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} G_t^{(n)}$$

or big N (slide)

→ λ controls bias - variance trade-off

$$\lambda \in (0,1) \Rightarrow \sum_{n=1}^{+\infty} (1-\lambda) \lambda^{n-1} = 1$$

short term rewards — long term rewards



$\lambda = 0 \Rightarrow G_t^{(\lambda)} = G_t^{(1)}$ TD(0)

$$\lambda = 1 \Rightarrow G_t^{(\lambda)} \overset{\substack{\text{episodic} \\ \text{(end at T)}}}{=} \cancel{(1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)}}^{\lambda=1} + \overbrace{(1-\lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} G_t^{(n)}}^{\text{post termination contributions}} = G_t (1-\lambda) \sum_{m=0}^{+\infty} \lambda^{m+T-t-1} = G_t (1-\lambda) \lambda^{T-t-1} \sum_{m=0}^{+\infty} \lambda^m$$

$= G_t$ because it sees all the timesteps $t : T$

$m = n-T+t$

geometric serie

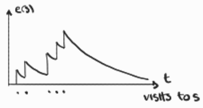
$$= G_t \cancel{(1-\lambda)} \lambda^{T-t-1} \cdot \frac{1}{\cancel{1-\lambda}} = G_t \cdot \lambda^{T-t-1} = G_t \quad \text{MC}$$

3) Backward TD (λ)

→ implement TD(λ) in an online way

- Associated with each state there is what is called an eligibility trace $e_t(s) \in \mathbb{R}^+$

→ e(s) records which states have been recently visited



At each step update $e_t(s)$ $\forall s \in S$ in this way:

$$e_t(s) = \begin{cases} \overset{\text{decay factor}}{\gamma\lambda} e_{t-1}(s) & \text{if } s \neq s_t,\ t \neq 0 \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t,\ t \neq 0 \\ 0 & \text{if } t = 0 \end{cases}$$

We define $\overset{\text{error}}{\delta_t} = r_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$

The update policy is $V(s) \leftarrow V(s) + \alpha \underset{\text{constant } \forall s \in S}{\delta_t} e_t(s)$ $\forall s \in S$

- BTD(0) has $e_t(s) = \mathbb{I}(s = s_t)$, thus equivalent to TD
- meanwhile with BTD(1) you get MC, but better → online + extendable to continuing tasks

→ can be shown that TD(λ) and BTD (λ) are equivalent (with $\gamma\lambda < 1$)

$$\underbrace{\alpha \left[\sum_{k=t}^{\infty} (\gamma\lambda)^{k-t} \delta_k \right]}_{\text{total update of } V(s_t) \text{ from BTD}(\lambda)} = \underbrace{\alpha \left[G_t^{\lambda} - V(s_t) \right]}_{\text{TD}(\lambda) \text{ update of } V(s_t)}$$

→ At the end of the episode they produced the same updates

```
∀episode
e(s) ← 0      ∀s∈S
S₀ ← initialize    t ← 0
while Sₜ ≠ terminal
        a ← π(Sₜ)
        take action a and observe (r, Sₜ₊₁)
        δ ← r + γV(Sₜ₊₁) − V(Sₜ)
        e(Sₜ) ← e(Sₜ) + 1
        ∀s ∈ S :
                V(s) ← V(s) + αδ e(s)
                e(s) ← γλe(s)
        t ← t+1
```

↳ exponential decay

4) SARSA (n)

Defining $Q_t^{(n)} \leftarrow \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n Q(s_{t+n+1}, a_{t+n+1})$

We got the update $Q(s,a) \leftarrow Q(s,a) + \alpha \left[Q_t^{(n)} - Q(s,a) \right]$

5) SARSA (λ)

Defining $Q_t^{(\lambda)} \leftarrow (1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} G_t^{(n)}$

we got the update $Q(s,a) \leftarrow Q(s,a) + \alpha \left[Q_t^{(\lambda)} - Q(s,a) \right]$

6) Backward SARSA (λ)

$$Q(S,A) \leftarrow Q(S,A) + \alpha \delta_t E_t(S,A) \quad \forall s,a \in S \times A(s)$$

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \mathbb{I}(s_t = s, a_t = a)$$

Approximate Solutions Methods

- Tabular methods have difficulties for large spaces (images) or continous ones
  - they lack the ability to recognize similar states and generalize

No more tabular version of V and Q but we define two parametric functions

$\hat{V}: S \times \Theta \rightarrow \mathbb{R}$ $\hat{Q}: S \times \Theta \rightarrow \mathbb{R}^{|A|}$

→ abilities to generalize from unseen states

→ needs to be differentiable for SGD

Two problems 1) Non-Stationary data → environment may change (thus the policy should change too)

2) Non-IID data → correlated events that could cause some problem with the convergence of SGD

MSE: $\mathcal{L}(\theta) = \mathbb{E}_{\mu}\left[(\overset{\text{unknown}}{v_\pi(s)} - \hat{v}(s,\theta))^2\right]$

μ: state probability distribution (how much we care about the error in state s)

LINEAR APPROXIMATORS $\hat{V}(s,\theta) = \overset{\text{state space encoding function}}{\phi(s)}^T \theta$

- $\mathcal{L}(\theta)$ is quadratic

- $\nabla_\theta \mathcal{L} = -2(v_\pi(s) - \hat{v}(s,\theta)) \cdot \frac{\partial \hat{v}(s,\theta)}{\partial \theta} = -2(v_\pi(s) - \hat{v}(s,\theta))\phi(s)$ $\Rightarrow$ $\theta_{t+1} \leftarrow \theta_t \overset{\text{gradient descend}}{+} \gamma\left[v_\pi(s) - \hat{v}(s,\theta)\right]\phi(s)$

- linear approximator is enough thanks to Representer Theorem (difficult part is to find the kernel ϕ)

Since $v_\pi(s)$ is unknown we need to replace it with the empirical target G_t

MC: G_t is unbiased but noisy

$\mathbb{E}[G_t \mid s = s_t] = v_\pi(s_t)$

- strong convergence property

```
loop:
    generate with $\pi$ an episode $S_0, A_0, R_1, \dots$
    for each step $t$:
        $\theta \leftarrow \theta + \alpha[G_t - \hat{v}(s_t,\theta)]\ \phi(s)$
```

TD(0): G_t is reliable but biased

- weak convergence property (for biased GD but faster)

- TD(0) under linear approximations converges

```
loop (episodes):
    $S_0 \leftarrow$ initialize
    loop (steps):
        $A \sim \pi(a \mid s)$
        take action $A$ and observe $R, S'$
        $\theta \leftarrow \theta + \alpha[R + \gamma \hat{v}(S',\theta) - \hat{v}(s,\theta)]\ \phi(s)$
        $S \leftarrow S'$
```

- Since the target depends on current θ the GD will be biased

⇒ semi-gradient methods because the gradient is not the true gradient but an approximation

- All the bootstrapping are affected

The eligibility for TD(λ) can be defined as: $E_t = \gamma\lambda E_{t-1} + \phi(s_t)$

- We can use batched episodes (instead of iteratives methods) to generate more stable gradients → solves iid problems with dataset sampling from experience replay buffer $(s,a,r,s') \sim D$
  - I could add a temporal weight to overcome the non-stationary data

DNN APPROXIMATORS Deep Q-Network (DQN) is a DNN to approximate Q

- it uses an experience replay buffer → breaks correlation between consecutives experience

cannot be done with Deep-SARSA since it depends on $a' \sim \pi_{old}$

- uses ε greedy policy

- It uses a separete NN to estimate Q-values target (θ^- parameters) → prevents rapid oscillations caused by moving targets (→ diverges otherwise)
  - it is updated periodically (not on every epoch $\theta^- = \theta$) → it reduces the bias

$\theta^- = \tau\theta + (1-\tau)\theta^-$ if $\tau = 1$ ⇒ hard update, otherwise soft

Simple DQN: $\mathcal{L}(\theta) = \overset{\text{over replay buffer}}{\mathbb{E}_D}\left[(r + \gamma \max_{a' \in A} \hat{Q}(s',a',\overset{\text{semi-gradient update}}{\theta^-}) - \hat{Q}(s,a,\theta))^2\right]$

- Problem: upward positive bias from taking the maximum of a noisy esteem $\hat{Q}(s,a,\theta^-)$ → problem is solved if the noise on $\hat{Q}_\theta$ and $\hat{Q}_{\theta^-}$ are decorrelated

⇒ splits action selection ($\hat{Q}_\theta$) and action evaluation ($\hat{Q}_{\theta^-}$) and then vice-versa

Double DQN:

$$\mathcal{L}(\theta) = \mathbb{E}_D\left[(r + \gamma \hat{Q}(s', \underset{a' \in A}{\operatorname{argmax}}\ \hat{Q}(s',a',\theta), \theta^-) - \hat{Q}(s,a,\theta))^2\right]$$

$$\mathcal{L}(\theta^-) = \mathbb{E}_D\left[(r + \gamma \hat{Q}(s', \underset{a' \in A}{\operatorname{argmax}}\ \hat{Q}(s',a',\theta^-), \theta) - \hat{Q}(s,a,\theta^-))^2\right]$$

- randomely select if updating θ or θ^- at each iteration

Duelling Networks

$$\hat{Q}(s,a,\overset{\theta=(\theta_V,\theta_A)}{\theta}) = \hat{V}(s,\theta_V) + \left[\hat{A}(s,a,\theta_A) - \max_{a'\in A}\hat{A}(s,a',\theta_A)\right]$$

Uses two fuction : 1) Value function $\hat{v}(s,\theta_V)$ → expected reward for being in a state s

2) Advantage function $\hat{A}(s,a,\theta_A)$ → quantifies the advantage to choose a particuear action in state s

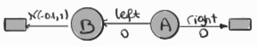
$A(s,a) = Q(s,a) - V(s)$

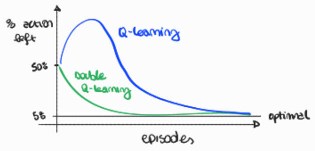
- In this way the model can learn independently what states are the best and what actions are advantageus and then combine them in the Q value $\hat{Q}(s,a,\theta_V,\theta_A) = \hat{V}(s,\theta_V) + \hat{A}(s,a,\theta_A)$

→ more stable and robust

1) can estimate Q-values from unseen action better, because it has a stronger starting point than $\hat{q}(s,a)$

2) faster convergence when a lot of similar actions since $V(s)$ will be shared and fixed and only $\hat{A}$ will change





→ once expirienced a positive reward going left ⇒ bias towards B

- no guarentees of convergence, higher variance, π diff

POLICY GRADIENT METHODS

→ learn a parametrized policy $\pi(a|s,\theta)$ not a value function to get the policy → more direct

- typically π^* approximation converges faster than V^*
- can be used in continous A space
- wrt ε-greedy the value of $\hat{\pi}_\theta$ change continuously with $d\theta$ changes → stronger convergence

I would like to $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta \pi_{\theta_t}(\overset{\text{optimal action in state } s}{a^*}|s)$. Unfortunately a^* is unknown ⇒ weight the updates $\theta_{t+1} \leftarrow \theta_t + \alpha \hat{Q}(s,a)\dfrac{\nabla_\theta \pi_{\theta_t}(a|s)}{\pi_{\theta_t}(a|s)}$ → correction in order to not update too much more probable a

$= \theta_t + \alpha \hat{Q}(s,a)\nabla_\theta \ln \pi_{\theta_t}(a|s)$

POLICY GRADIENT THEOREM

$\hat{\pi} \in C^1(\mathbb{R}^n)$ continuously differentiable

In any policy gradient method where $\underbrace{L(\theta) \propto \mathbb{E}_{\hat{\pi}}[V_{\hat{\pi}}(s)]}_{\text{perform gradient ascend}}$ (true value function wrt $\hat{\pi}$) $= \sum_s d_{\hat{\pi}}(s) V_{\hat{\pi}}(s) = \sum_s d_{\hat{\pi}}(s)\sum_a \hat{\pi}(a|s) q_{\hat{\pi}}(s,a)$ ⟹

$$\nabla_\theta J(\theta) \propto \sum_s \underbrace{d_{\hat{\pi}}(s)}_{\text{on policy state distribution}} \sum_a q_{\hat{\pi}}(s,a)\nabla_\theta \hat{\pi}(a|s,\theta) = \mathbb{E}_{\underset{s\sim d_{\hat{\pi}}(s)}{\hat{\pi}}}\left[\sum_a \nabla_\theta \hat{\pi}(a|s,\theta) q_{\hat{\pi}}(s,a)\right] \overset{\nabla_\theta \ln \hat{\pi} = \frac{1}{\hat{\pi}}\nabla_\theta \hat{\pi}}{=} \mathbb{E}_{\hat{\pi}}\left[\sum_a \hat{\pi}(a|s,\theta)\nabla_\theta \ln \hat{\pi}(a|s,\theta) q_{\hat{\pi}}(s,a)\right] =$$

$$= \mathbb{E}_{\hat{\pi}}\left[\nabla_\theta \ln \hat{\pi}(a|s,\theta) q_{\hat{\pi}}(s,a)\right]$$

$s\sim d_{\hat{\pi}}(s),\ a\sim \hat{\pi}$
$(s,a)\sim d_{\hat{\pi}}(s,a)$

$d_{\hat{\pi}}(s) = \lim_{t\to+\infty} P(s_t = s | s_0, \hat{\pi})$

$\nabla_\theta J(\theta)$ depends both on : 1) action selection $(\hat{\pi}_\theta)$

2) state distribution $(d_{\hat{\pi}})$ → it's change is a function of the environment, and thus unknown

- we do not need $\nabla d_{\hat{\pi}_\theta}$ for previous theorem and neither $\nabla q_{\hat{\pi}_\theta}(s,a)$

MC policy gradient

G_t is an unbiased sample of $Q_{\hat{\pi}}(s,a)$ ⇒ $G_t = \mathbb{E}_{\hat{\pi}}[Q_{\hat{\pi}}(s_t,a_t) | s_t,a_t]$

By sampling the gradient policy theorem

$$\nabla_\theta L(\theta) \propto \overbrace{\underbrace{\frac{1}{N}\sum_{i=1}^{N}}_{\text{collect } N \text{ episodes}}\sum_{t=1}^{T} \nabla_\theta \ln \hat{\pi}_\theta(a_t^i|s_t^i,\theta)}^{\text{Maximum Likelihood gradient}}\ \underset{\text{weights more successful trajectories}}{G_t^i} \Rightarrow \theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta L(\theta)$$

- Weighted Maximum Likelihood

→ on-policy approach (otherwise importance sampling, but higher variance)

ACTOR CRITIC METHODS

We want to reduce MC policy gradient variance

Two entities : 1) Actor $\hat{\pi}$ that generates actions

2) Critic $\hat{Q}$ that evaluates the quality of the action taken
- estimated by TD, MC, … methods
- or using DNN if $w \neq \theta$ two distinct NN, no gradient conflicts

→ Using a function to estimete G instead of the environment (stochastic) has lower variance but higher bias

COMPATIBLE FUNCTION APPROXIMATION THEOREM

If 1) $\nabla_w \hat{Q}(s,a,w) = \nabla_\theta \ln \hat{\pi}(a|s,\theta)$ [compatibility hyp.] holds when : $Q(s,a,w) = \nabla_\theta \ln \hat{\pi}(a|s,\theta)^T w$

2) $\mathbb{E}_{\hat{\pi}}[(\underset{\text{true function}}{Q_{\hat{\pi}}} - \hat{Q})^2] \xrightarrow{t\to+\infty} 0$ [unbiased property]

⇒ The policy gradient theorem still holds true $\nabla_\theta L(\theta) \propto \mathbb{E}_{\underset{(s,a)\sim d_\pi(s,a)}{\hat{\pi}}}[\nabla_\theta \ln \hat{\pi}(a|s,\theta)\hat{Q}(s,a,w)]$

The variance can be further reduced using a base line $B(\cdot)$ → represents empirical knowledge about the model

$\nabla_\theta L(\theta) \propto \mathbb{E}_{\hat{\pi}}\left[\sum_a \nabla_\theta \hat{\pi}(a|s,\theta)\left(Q_{\hat{\pi}} - B(\cdot)\right)\right]$ → keep only the variance about the action, not future states

- If $B(\cdot)$ depends only on s ⇒ doesn't change the gradient since $\mathbb{E}_{\hat{\pi}}\left[\sum_a \nabla_\theta \hat{\pi}(a|s,\theta) B(s)\right] = \mathbb{E}_{\hat{\pi}}\left[B(s)\nabla_\theta \sum_a \hat{\pi}(a|s,\theta)\right] = \mathbb{E}_{\hat{\pi}}[B(s)\nabla_\theta 1] = 0$

- common choice $B(s) \doteq V_{\hat{\pi}}(s)$ → reducing the variance of gradient updates (no bias introduced) → faster convergence

- becomes the Advantage function

→ Without baseline if $Q_{\hat{\pi}}(s,a) > 0\ \forall(s,a)$ ⇒ the network couldn't learn to do less of an action but just more the others

Advantage function $A_{\hat\pi}(s,a) = Q_{\hat\pi}(s,a) - V_{\hat\pi}(s)$ measures the value added by action a to state s

- advantage policy gradient reformulation $\nabla_\theta \mathcal{L}(\theta) \propto \mathbb{E}_{\hat\pi}[\nabla_\theta \ln \hat\pi(A|S,\theta)\, A_{\hat\pi}(S,A)]$

→ In actor-critic scenario: $\hat A(s,a,w,p) = \hat Q(s,a,w) - \hat V(s,p)$

- thus we should learn (θ, w, p) but in reality:

$$A_{\hat\pi}(s,a) = \mathbb{E}_{s'}\Big[\overbrace{r + \gamma V_{\hat\pi}(s')}^{Q_{\hat\pi}(s,a) = \mathbb{E}_{s'}[r+\gamma V_{\hat\pi}(s')\,|\,s,a]} - V_{\hat\pi}(s)\,\Big|\, s,a\Big]$$

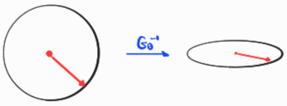
$A_{\hat\pi}$ could be biased if $V_{\hat\pi}$ estimate is not perfect

NATURAL POLICY GRADIENT

Policy gradients could perform poorly because of plateaus or discontinuities in policy space → small θ change causes big $\hat\pi$ changes

→ critic $\hat Q$ not valid anymore

Solution: make changes to θ proportional to policy variation



$$\begin{cases} d^* = \underset{d}{\text{argmin}}\ \mathcal{L}(\theta_t + d) \\ \text{s.t.}\ D_{KL}(\hat\pi(\theta_t)\,\|\,\hat\pi(\theta_t+d)) \leq \varepsilon \end{cases}$$

ε trust region radius

↳ limits the loss of information (penalize d that alters a lot $\hat\pi_\theta$)

$$D_{KL}(\hat\pi(\theta_t)\,\|\,\underset{\theta_t+d}{\hat\pi(\theta_{t+1})}) = \int_{S\times A} \hat\pi(a|s,\theta_t) \ln \frac{\hat\pi(a|s,\theta_t)}{\hat\pi(a|s,\theta_{t+1})}\, ds\, da$$

Using the tylor approximation of second order of D_{KL} we can find that $\theta_{t+1} - \theta_t \propto G_\theta^{-1} \nabla_\theta \mathcal{L}(\theta) \overset{\text{policy th.}}{=} G_\theta^{-1} \nabla_\theta \ln \hat\pi(a|s,\theta)\, Q_{\hat\pi}(s,a)$

With G_θ the Fisher information matrix $G_\theta = \mathbb{E}_{\hat\pi}[\nabla_\theta \ln \hat\pi(a|s,\theta)\, \nabla_\theta \ln \hat\pi(a|s,\theta)^T]$

→ It's else a covariance matrix, where big entry in G_θ means small change in θ ⇒ big change in $\hat\pi$

→ Its inverse helps to normalize the parameters search space by making all equally important

If compatibility holds we know that $Q(s,a,w) = \nabla \ln \hat\pi(a|s,\theta)^T w$

$$\Rightarrow \quad \Delta\theta_t \propto G_\theta^{-1} \nabla_\theta \ln \hat\pi(a|s,\theta)\, \nabla_\theta \ln \hat\pi(a|s,\theta)^T w = G_\theta^{-1} G_\theta w = w$$

MODEL-BASED RL

→ learns the model and from it get the value function from simulated experience (planning)

learn faster with fewer data

- needs to learn $P_{ss'}^a$ and R_s^a
- The simplest one is table lookup: $P_{ss'}^a = \frac{1}{N(s,a)} \sum_{t=1}^{T-1} \mathbb{I}(s_t, a_t, s_{t+1})$ $\quad R_s^a = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbb{I}(s_t, a_t)\, r_t$

$M_\eta = \langle P_\eta, R_\eta \rangle$ model variables

- Inaccurate models lead to inaccurate policies → two source of errors (M_η and $\hat\pi$)

→ If the uncertainty is too high, go back to model-free or use an integrated approach

Integrated Approach:
1) M_η learning from real experience
2) $\hat V, \hat Q, \hat\pi$ learning from both M_η and real experience

BACKWARD SEARCH

Dyna-Q → integrated approach + deterministic environment

```
Input: $S, A, \gamma$
Initialize $Q(s,a)$ and $M(s,a)$    $\forall (s,a) \in S\times A$
loop:
    $S \leftarrow$ current state
    $A \leftarrow \varepsilon\text{-greedy}(s,Q)$
    execute A and observe $(R,S')$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$    # TD(0)
    model $M(s,a) \leftarrow (r,s)$    # Assuming deterministic env.
    for n times:
        choose $s \in S$ from $H$ history
        choose $a \in A$ from $H(a)$
        $(r,s') \leftarrow M(s,a)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
```

It has an extension Dyna-Q+ to add an exploration bonus

→ Dyna-Q fails if the env is not static (r changes)

$(r + K\sqrt{T_{sa}}, s') \leftarrow \text{Model}(s,a)$

↳ #timesteps last tried (s,a)

- Increasing the reward induces the policy to follow those a tried a long time ago

- In the slides is said that (s,a) ~ weighed distribution $W_s(T) = R_s^T + K\sqrt{T}$

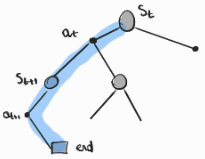
Dyna-2 → Uses two Q functions:

1) Q^T (transient) → adjusted from simulation
2) Q^P (persistent) → learned only from real data

- Takes action combining both Q^T and Q^P $\quad a_{t+1} \leftarrow \varepsilon\text{-greedy}(Q^T(s_t,\cdot) + Q^P(s_t,\cdot))$

FORWARD SEARCH

builds a search tree with current state S_t as root → uses M_η as look ahead
→ simulates episodes from now with the model
→ apply model-free methods to simulated episodes



NAIVE MC Search

· you are not learning $\hat{\pi}$ from the simulations
→ inefficient

```
Input  M, π, S_t
for all a ∈ A do:
    simulate K episodes  $\{S_t, a, R^k_{t+1}, S^k_{t+1}, a^k_{t+1}, ----, S^k_T\} \sim M_\eta, \pi$
    $Q(s_t,a) = \frac{1}{K}\sum_{k=1}^{K} G^k_t$   → mean return on simulation
$a_t \leftarrow \arg\max_a Q(s_t,a)$
```

completely from simulations

MC TREE-SEARCH

→ Builds a search tree with visited states and actions

1) Selection: start from root S_0 and traverse the tree using a tree policy → like UCB to balance exploration and exploitation

starting point where I want to take an action

initially empty

strategy to select which node to visit after: ε-greedy

stops when:
a) reach a node with an action that is not expanded
b) using tree policy the next node to visit is not on the tree
c) terminal state is reached → in this case skip to back propagation

2) Expansion: case a) → add action node and a sampled state from M_η
case b) → action was already present, add only s ⇝ stochastic result state after a

3) Simulation: from the state S_t simulate using a default policy until terminal state reached
$\{S_t, A^k_t, R^k_{t+1}, ----, S^k_T\} \sim M_\eta, \pi$

greedy

4) BackPropagation: update every (s,a) along the path from selected node to root

a) Increment visit count $N(s,a)$ by one

b) $G^k_t \leftarrow r^k_t + \gamma G^k_{t+1}$ (initial G is the return of selected node along the generated path)

c) Update Q estimate $Q(s,a) = \frac{1}{N(s,a)} \sum_{k} \sum_{u=1}^{T_k} \mathbb{I}(S^k_u = s, A^k_u = a)\, G^k_u$ → could be done more efficiently with incremental mean

# of simulations done

Repeat the four phases until there is computation time, then take a_t and discard the tree (some implementation codes reuse it)

→ Exhaustive search is impractible in complex spaces
→ Highly parallelizable
→ The two policies improves

TD-SEARCH

→ uses bootstrapping instead of sampling simulation · apply SARSA from (S_t, a_t)
$\Delta Q(S_t, a_t) = \alpha[R_t + \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)]$ · could be use to simulate n steps ahead like TD(n)

From S_t update Q at each iteration → lower variance
· Dyna2 learns Q^T in this way

EXPLORATION-EXPLOITATION TRADE-OFF

In ε-greedy the exploration is random (noise + greedy)

Two strategies available: 1) Optimism → estimates action-state uncertainty and explore the more uncertain
2) Information State → uses hystory knoledge to explore

MULTI-ARMED BANDIT SCENARIO

- deterministic starting state S_0
- multiple actions that brings to a different terminal state (no state space) with rewards that depends on each probability distribution

→ we want to find $R^{a}_{S_0}$ (expected reward) in order to maximize $\sum_{t=1}^{k} r_t$

↳ cumulative return over actions

- using greedy we are exploiting our current knowledge to maximize one-step-ahead return, but we are not exploring
- using ε-greedy may produce greater total reward in the long run

true action value (unknown)

$$V^* = Q(a^*) = \max_a Q(a)$$

regret: expected err doing suboptimal a

$$I_t = \mathbb{E}_{a_t \sim \pi}[V^* - Q(a_t)]$$

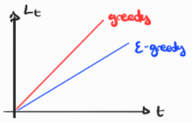
total regret

$$L_T = \sum_{t=1}^{T} I_t$$

minimize $\sum_{t=1}^{T} I_t$ ⇔ maximize $\sum_{t=1}^{T} r_t$

number of selection of a action T timesteps (depends on π)

$$L_T = \sum_{t=1}^{T} \mathbb{E}[V^* - Q(a_t)] = \sum_a \mathbb{E}_\pi[N_T(a)]\,(V^* - Q(a))$$



greedy (no exploration) or ε-greedy (random exploration) has linear regret

- in greedy $\mathbb{E}[N_T(a)]$ can be T for a single a^* ⇒ $L_T = T(V^* - Q(a^*))$
- in ε-greedy $\mathbb{E}[N_T(a)] \geq \frac{\varepsilon}{|A|}T$ ⇒ $L_T \geq \frac{\varepsilon}{|A|} T \sum_a (V^* - Q(a))$

Hoeffding Inequality: let $x_1, \ldots, x_n$ random variables in $[0,1]$, let $\bar{x}_t = \frac{1}{t}\sum_{k=1}^{t} x_k$ ⇒

$$P(\mathbb{E}[X] - \bar{x}_t \geq u) \leq e^{-2tu^2} \quad \forall u \geq 0$$

rewards in $[0,1]$

estimate $\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{s=1}^{t} r_s \cdot \mathbb{I}(a_s = a)$

(UCB) Upper Confidence Bound: Call the upper bound on $Q(a)$ → $\hat{U}_t(a)$ ⇒ $P[Q(a) > \hat{Q}_t(a) + \hat{U}_t(a)] \leq e^{-2N_t(a)\hat{U}_t(a)^2} = p$

→ $\hat{U}_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$ → less time chosen greater the uncertainty

OPTIMISM Between all the high probable (1-p) models choose the most optimistic to improve the policy $\hat{Q}_t(a) + \hat{U}_t(a)$

- If we reduce $p_t = t^{-\alpha}$ (fixed $\alpha > 0$) we have a sublinear policy

$$a = \arg\max_a \left[\hat{Q}_t(a) + \sqrt{\frac{\alpha \log t}{2N_t(a)}}\right] = \pi(S_0^t)$$

increases with uncertainty and time

→ Guarantees asymptotic convergence to optimal action

- It has sublinear regret → because it doesn't use random exploration but it's based on uncertain actions that could be optimal

STATE OF THE ART MODELS

PROXIMAL POLICY GRADIENT

We want to limit the variation of π across weight updates

. Is heavy to put a constraint on D_{KL} or to compute G_θ^{-1}

To keep D_{KL} small at every step we can:

1) PPO - Penalty (proximal policy optimization)

$$\mathcal{L}(\theta) = \underset{\text{over all episode}}{\mathbb{E}_t}\left[\overbrace{\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{k-1}}(a_t \mid s_t)}}^{\text{how much } P \text{ of taking } a_t \mid s_t \text{ would increase with } \theta_t}\hat{A}_w(a_t,s_t) - \underbrace{\beta_n D_{KL}(\pi_\theta \,\|\, \pi_{\theta_{k-1}})}_{\text{penalize big } D_{KL}}\right]$$

(a_t, s_t) were collected via $\pi_{\theta_{k-1}}$ → importance sampling

$$\beta_{n+1} = \begin{cases} \frac{\beta_n}{2} & \text{if } d < \frac{2}{3}\bar{d} \\ 2\beta_n & \text{if } d > \frac{3}{2}\bar{d} \\ \beta_n & \text{otherwise} \end{cases}$$

user-defined th.

2) PPO - Clipping

$$\mathcal{L}(\theta) = \mathbb{E}_t\left[\min\left\{\overbrace{\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{k-1}}(a_t \mid s_t)}}^{r_t}, g(\varepsilon, A_t)\right\}\hat{A}_w(a_t,s_t)\right]$$

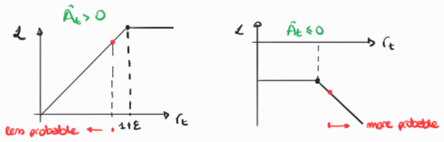
$$g(\varepsilon, A_t) = \begin{cases} 1+\varepsilon & \text{if } \hat{A}_t > 0 \\ 1-\varepsilon & \text{otherwise} \end{cases}$$

We are clipping r_t:

1) if $\hat{A}_t$ is good ($\hat{A}_t > 0$) by $1+\varepsilon$

2) if $\hat{A}_t$ is bad by $1-\varepsilon$

→ prevents too big changes (r_t really high)



. There is no reward of having $r_t \notin [1-\varepsilon, 1+\varepsilon]$

DEEP DETERMINISTIC POLICY GRADIENT

DPG Theorem: let $\mathcal{L}(\theta) = \mathbb{E}_{s\sim d(s)}\left[\overset{\text{reward}}{R}(s, \overset{\text{deterministic}}{\pi_\theta(s)})\right]$ with $\pi_\theta, R, P \in C^1$ $\Rightarrow$ $\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{s\sim d(s)}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q(s,a)\big|_{a=\pi_\theta(s)}\right]$

more efficient specifically in continuous-action space

Deterministic compatible Value Approximation Theorem: If we use $\hat{Q}_w(s,a) \in C^1$ a parametrized version of $Q(s,a)$ → DPG theorem holds ⟺

1) $\nabla_a \hat{Q}_w(s,a)\big|_{a=\pi_\theta(s)} = \nabla_\theta \pi_\theta(s)^T w$

2) $\mathbb{E}_{s\sim d(s)}\left[(\nabla_a \hat{Q}_w(s,a) - \nabla_a Q(s,a))^2\big|_{a=\pi_\theta(s)}\right] \xrightarrow{t\to+\infty} 0$

guarantee to have no bias substituting Q with $\hat{Q}_w$

From 1) derive that $Q_w(s,a) = a^T \nabla_\theta \pi_\theta(s)^T w$

→ $\delta_t = r_t + \gamma Q_{w_t}(s_{t+1}, \pi_\theta(s_{t+1})) - Q_{w_t}(s_t, a_t)$ → TD(0) error

① $\theta_{t+1} = \theta_t + \overset{\text{learning rate } \theta}{\lambda_\theta} \nabla_\theta \mathcal{L} = \theta_t + \lambda_\theta \nabla_\theta \pi_\theta(s) \nabla_a Q(s,a)\big|_{a=\pi(s)}$ $\Rightarrow$ $\theta_{t+1} = \theta_t + \lambda_\theta \nabla_\theta \pi_\theta(s) \nabla_\theta \pi_\theta(s)^T w_t$

② $w_{t+1} = w_t + \lambda_w \delta_t \nabla_w Q_w(s_t, a_t)$ $\Rightarrow$ $w_{t+1} = w_t + \lambda_w \delta_t \nabla_\theta \pi_\theta(s)\, a$

Natural Policy Modification: $\theta_{t+1} = \theta_t + \lambda_\theta w_t$ when we use the Fisher matrix to weight $\nabla_\theta \mathcal{L}(\theta)$

TWIN DELAYED DDPG (TD3)

→ DDPG is sensitive to hyperparameters and tends to overestimate Q

① TARGET POLICY SMOOTHING

→ add noise to π, making it difficult to exploit Q errors (smoothing)

$a = \text{clip}\left[\pi_\theta(s) + \text{clip}(\varepsilon, -c, +c), a_l, a_h\right]$ with $a_l, a_h \in \mathbb{R}$ $a_l < a_h$, $c \in \mathbb{R}^+$, $\varepsilon \sim N(0,1)$

② CLIPPED DOUBLE-Q LEARNING

Uses two Q_{w_1}, Q_{w_2} trained independently → Difficult to have the same overestimation

$y(r,s') = r + \gamma \min_{i=1,2} Q_{w_i}(s', \overset{\text{clipped}}{a_\theta(s')})$

$L(w_1) = \mathbb{E}_D\left[(Q_{w_1}(s,a) - y(r,s'))^2\right]$ → same for w_2 (start from different $\bar{w}_1 \neq \bar{w}_2$)

$\pi_{\theta'} = \underset{\pi_\theta}{\arg\max}\, \underbrace{\mathbb{E}_D\left[Q_{w_1}(s, \pi_\theta(s))\right]}_{-\mathcal{L}(\theta)}$ ↳ uses DPG to update θ, but not compatibility → BIAS

③ DELAYED POLICY UPDATES

→ updates policy π_θ once every $\overset{\text{typically } d=2}{d}$ critic update (w_1, w_2)

. this allows to train π_θ with a lower variance value estimate → more stable

SOFT ACTOR CRITIC (SAC)

→ We can use entropy regularization to enforce exploration

· Given a density function p and a random variable $x \sim p$ ⟹ $H(p) \doteq -\mathbb{E}_x[\ln(p(x))]$

Define a new loss with exploratory bonus: $\pi^* = \arg\max_{\pi} \mathbb{E}_b\Big[\underbrace{\sum_{t=0}^{+\infty} \gamma^t (R(s_t,a_t)}_{G_t} + \boxed{v H(\pi(\cdot|s_t))}\Big]$ $v>0$

↳ entropy bonus → more exploration

not used in TD3

- Approximate $G_t = \min_{i=1,2} \hat{Q}_{w_i}(a_t, s_t)$ ⟹ $\arg\max_{\pi_\theta} \mathbb{E}_{\substack{s\sim D \\ a\sim\pi_\theta}}\left[\min_{j=1,2} Q_{w_j}(s,a) - v \ln \pi_\theta(a|s)\right]$

↳ typically decreased over time v_t

reparametrization trick since is diffucult to compute directly π_θ^* I could divide it in two:

$a_\theta(s,\varepsilon) \doteq \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon) \in [-1,+1]$ with $\varepsilon \sim \mathcal{N}(0,1)$ now $\pi_\theta \overset{\text{circa}}{\sim} \mathcal{N}(\mu_\theta, \sigma_\theta)$

· Decoupled randomness of $\pi_\theta(a|s)$ from model parameters

⟹ $\pi^* = \arg\max_{\theta} \mathbb{E}_{\substack{s\sim D \\ \varepsilon\sim\mathcal{N}}}\left[\min_{j=1,2} Q_{w_j}(s, a_\theta(s,\varepsilon)) - v \ln \pi_\theta(a_\theta(s,\varepsilon)|s)\right]$ ⟹ $\pi_\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$

- Can be written the entropy bellman equation in order to learn Q

$$Q_\pi(s,a) = \mathbb{E}_{\substack{s'\sim p \\ a'\sim\pi(\cdot|s')}}\left[R(s,a) + \gamma\big(Q_\pi(s',a') - v\ln(\pi(a'|s'))\big) \,|\, s,a\right]$$

⟹ $\mathcal{L}(w_i) = \mathbb{E}_\pi[\delta_t^2]$ → similar to TD3 but with entropy correction

Differences
1) TD3 uses a deterministic policy
2) SAC uses entropy regularization
3) SAC doesn't use policy smoothing

ALPHA THEORY

Mathematical field in which the Axiom of Archimede lacks

- let F be a totally ordered field → $\forall x,y \in F,\ 0<x<y,\ \exists n \in \mathbb{N}:\ y<nx$

  $\nexists n \in \mathbb{N}:\ \alpha < nx \quad \forall x \in \mathbb{R} \subset \mathbb{E}$

Axiom 1 (existence): $\exists$ an ordered field $\mathbb{E} \supset \mathbb{R}$ whose number are called euclidian number [and a function $\lim_{n\uparrow\alpha}: \underset{\text{sequence}}{\mathbb{R}^{\mathbb{N}}} \to \mathbb{E}$: axiom 3]

DEF: given $\varepsilon \in \mathbb{E}$ then:

1) ε is infinite $\iff \forall n \in \mathbb{N},\ |\varepsilon| > n$
2) ε is finite $\iff \exists n \in \mathbb{N}:\ \frac{1}{n} < |\varepsilon| < n$
3) ε is infinitesimal $\iff \forall n \in \mathbb{N},\ |\varepsilon| < \frac{1}{n}$

Axiom 2: $\exists$ a function $num: \overset{\text{set of sets}}{\mathcal{U}} \to \mathbb{E}$

- $num(A) = |A|$ if A is finite
- $num(A) < num(B)$ if $A \subset B$ → $num(\mathbb{N} \cup \{a\}) = \alpha + 1$ but $|\mathbb{N} \cup \{a\}| = |\mathbb{N}|$
- $num(A \cup B) = num(A) + num(B) - num(A \cap B)$
- $num(A \times B) = num(A) \cdot num(B)$ → contrast with cantour $num(\mathbb{N} \times \mathbb{N}) = \alpha^2$ but $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$

DEF: $\alpha \doteq num(\mathbb{N})$

$\eta \doteq \alpha^{-1} \qquad \lim_{n\uparrow\alpha} \frac{1}{n} = \eta$

Axiom 3: every sequence $\varphi, \varphi' : \mathbb{N} \to \mathbb{R}$ has a unique α limit which satisfy the following properties

- If $\varphi(n) = n \quad \forall n \in \mathbb{N} \Rightarrow \lim_{n\uparrow\alpha} \varphi(n) = \alpha$
- If $\varphi(n) = r \quad \forall n \in \mathbb{N} \Rightarrow \lim_{n\uparrow\alpha} \varphi(n) = r$
- $\lim_{n\uparrow\alpha} \varphi(n) + \lim_{n\uparrow\alpha} \varphi'(n) = \lim_{n\uparrow\alpha} [\varphi(n) + \varphi'(n)]$
- $\lim_{n\uparrow\alpha} \varphi(n) \lim_{n\uparrow\alpha} \varphi'(n) = \lim_{n\uparrow\alpha} [\varphi(n) \cdot \varphi'(n)]$

IMPLICATIONS:
1) if $\varphi(n) \in \psi(n) \Rightarrow \varphi[\alpha] \in \psi[\alpha]$ $\in$ operator preservance
2) Given A non empty : $A^* = \{ \varphi[\alpha] \mid \varphi: \mathbb{N} \to A \}$ eg. $f^*: A^* \to B^*$
3) Gives $\mathbb{R}^* = \{ \varphi[\alpha] \mid \varphi: \mathbb{N} \to \mathbb{R} \}$

Axiom (Internal set): If $\psi(n)$ is a sequence of non-empty set $\Rightarrow \lim_{n\uparrow\alpha} \psi(n) \overset{\text{internal set}}{=} \{ \lim_{n\uparrow\alpha} \varphi(n) \mid \varphi(n) \in \psi(n) \ \forall n \}$

sequence that picks an element $\forall n \in \mathbb{N}$ from ψ
eg. $\psi(n) = \{0, \frac{1}{n}\}$ then $\varphi(n) = \frac{1}{n} \in \psi(n)$

Axiom (Extension): let $\varphi: \mathbb{N} \to \mathbb{R}$ and $f: \mathbb{R} \to \mathbb{R}$ be a function for which $f \circ \varphi$ is defined $\Rightarrow f$ can be uniquely extended at $\varepsilon = \lim_{n\uparrow\alpha} \varphi(n)$ as follows:

$f^*(x) = f(x) \quad \forall x \in \mathbb{R}$

$f^*(\varepsilon) = \lim_{n\uparrow\alpha} f(\varphi(n)) \qquad f^*: \mathbb{E} \to \mathbb{E}$

↳ can be demonstrated that it doesn't depend on φ definition

- define $\varphi(n) = x \quad \forall x \in \mathbb{R} \Rightarrow f^*(x) = \lim_{n\uparrow\alpha} f(x) = f(x)$

  ↳ constant sequence $\psi(n) = f(x)$

DEF An elementary formula is a finite string of symbols in first order logic (alphabet, $\underbrace{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow}_{\text{logic connectives}}$, $\underbrace{\exists, \forall}_{\text{quantifiers}}$, $=$, $\in$)

Th (Transfer Principle) let $\sigma(x_1, \dots, x_n)$ an elementary formula and let $\varphi_1, \dots, \varphi_k$ be arbitrary sequences

$\sigma(\varphi_1(n), \dots, \varphi_k(n))$ holds almost every where $\iff \sigma(\lim_{n\uparrow\alpha} \varphi_1(n), \dots, \lim_{n\uparrow\alpha} \varphi_k(n))$

↳ does not hold in a measure zero set
$\exists n^* \in \mathbb{N}:\ \forall n > n^*\ \ \sigma(\varphi_1(n), \dots \varphi_k(n))$ holds

→ guarantees that some properties are preserved by the α-limit eg. continuity, differentiability, ...

DEF (completeness) X is complete $\iff$ every non-empty subset of X having an upper-bound must have a supremum in X

- Transfer principle doesn't transfer $\mathbb{R}$ completeness

  eg. $\mu(0) = \{ y \in \mathbb{E} \mid y \text{ is infinitesimal} \}$ is bounded by 1 but doesn't have a sup in $\mathbb{E}$

$\mathbb{R}$ is too large to fit in computers

Algorithmic Field is the set of all numbers that can be represented exactly in a machine (also number not present in the original field it approximate like nan, $\pm\infty$) $(\hat{\mathbb{R}}, \hat{\mathbb{E}})$

1) Fixed length representation
   - faster code with possibility of hardware accellerators
   - deterministic time consumption

2) Symbolic representation
   - variable size with 'infinite' precision → programs will be slower

Algorithmic Number (AN) : $\varepsilon \in \mathbb{E}$ is an AN $\iff$ can be represented by a finite sum $\varepsilon = \sum_{k=0}^{\ell} r_k \alpha^{s_k}$ with $r_k \in \mathbb{R}$, $s_k \in \mathbb{Q}$, $s_k > s_{k+1}$

Th (AN normal form) : Any AN ε can be represented by $\varepsilon = \alpha^p P(\eta^{\frac{1}{m}})$ with $p \in \mathbb{Z}$, $m \in \mathbb{N}$, $P(x)$ is a polynomial with real coefficients : $P(0) \neq 0$

· parallelism $1.37 \cdot 10^3$ $\quad$ 10^3 is α^p and $1.37 = 1 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2}$ is $P(\eta^{\frac{1}{m}})$ polynomial with negative coefficients in α

Two problems arise with AN :

1) The inverse of a AN is not always an AN (not closed under inversion) eg. $(1+\alpha)^{-1} \notin \hat{\mathbb{E}}$ $\quad$ $\frac{1}{1+\alpha} = \frac{1}{\alpha(1+\eta)} \overset{taylor}{=} \eta\left[1 - \eta + \eta^2 - \eta^3 + \dots\right]$ $\quad$ $\frac{1}{1-x} = 1 + x + x^2 + \dots$

· 3^{-1} has infinite # of digits

2) They have variable length coding

To solve those we need truncation of $P(\cdot)$ $\quad$ $P(x) = p_0 x^0 + \dots + p_m x^m \Rightarrow tr_n(P(x)) := \begin{cases} P(x) & n \geq m \\ p_0 x^0 + \dots + p_n x^n & n < m \end{cases}$

ensures closeness under basic operations

Bounded AN (BAN) : $\varepsilon \in \mathbb{E}$ is a BAN $\iff$ can be represented by $\varepsilon = \alpha^p P_n(\eta)$ $\quad$ AN with $m=1$ $\quad$ [BAN $\subset$ AN] $\quad$ $10^n P(10^{-1})$ notazione scientifica

integer polynomium $\alpha^2\left[1 + 0.7\eta + 3.41\eta^2\right]$

- sufficient for elementary operations (needed in lex) but not for non-linear ones eg. $x^2 = \alpha \Rightarrow x = \sqrt{\alpha} \notin$ BAN
- $m=1$ because operation between BANs are a lot easier then ANs

LMOP (lexicographic M-O problems) $\quad \begin{cases} \operatorname{lex\,min}_x & f_1, \dots, f_n \\ x \in D \end{cases}$ $\quad$ $D \subseteq \mathbb{R}^n$ $\quad$ $f_i : \mathbb{R}^m \to \mathbb{R}$

```
$\Omega = D$
for $i = 1, \dots, n$ do:
    if $\Omega = \{\emptyset\} \vee |\Omega| = 1$ then
        break
    $\Omega = \{x \mid \operatorname{argmin}_{x \in \Omega} f_i(x)\}$
return $\Omega$
```

It can be reformulated with n optimization problems

PRE-EMPTIVE APPROACH

$$\begin{cases} \min_x f_i(x) \\ f_j(x) = \min_{x \in D} f_j(x) \quad \forall j = \{1, \dots, i-1\} \\ x \in D \end{cases}$$

→ after each run a constrain is added

- it gets more and more complex
- bad use of hardware (cache) and software (unless the warm start)
- nature of the problem may change eg. convex to non-convex with new constraints

SCALARIZATION $\quad \{w_i\}_{i=1}^n : \frac{w_{i+1}}{w_i} \ll 1$

$$\begin{cases} \min_x \sum_{i=1}^n w_i f_i(x) \\ x \in D \end{cases}$$

- no guarantees of problem equivalence
- choiche of weights are error-prone and arbitrarely → may induce numeric instabilities

NON-STANDARD SCALARIZATION $\quad w_i = \alpha^{1-i}$ → the choice is not unique, it is sufficient to have infinitesimal $\frac{w_{i+1}}{w_i}$

$$\begin{cases} \min_x \sum_{i=1}^n \alpha^{1-i} f_i(x) \\ x \in D \end{cases}$$

- the problems are equivalent and here we use a single objective
- the transfer principle guarantees the equivalence between optimal solutions of the original LMOP and its scalarization

Th (Equivalence) Given a LMOP with $f_1, \dots, f_n$ real functions, with priority induced by natural order

$\Rightarrow \exists$ an equivalent scalar program over the same domain, whose objective is non-standard with the following form : $F(x) = \beta_1 f_1(x) + \dots + \beta_n f_n(x)$

with $\beta_1, \dots, \beta_n \in \mathbb{E}$: $\frac{\beta_{i+1}}{\beta_i} \approx 0 \quad \forall i = \{1, \dots, n-1\}$

Proof: Let Ω be the domain of the two problems, and $w \in \Omega$ be a global maximum of LMOP

$\Rightarrow \nexists w' \in \Omega : f_1(w') > f_1(w) \vee \left[\nexists i \in \{2, \dots, n\} : f_j(w') = f_j(w) \quad \forall j = \{1, \dots, i-1\} \wedge f_i(w') > f_i(w)\right]$

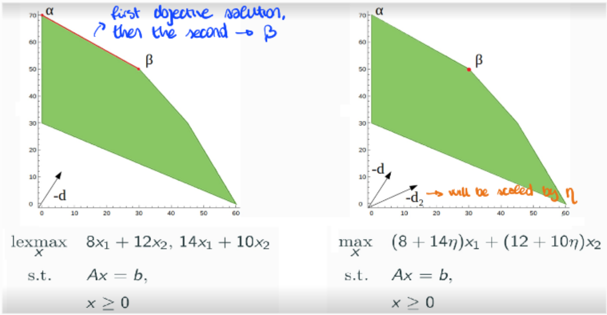
This is true $\iff \nexists w' \in \Omega : F(w') > F(w)$ by the definition of F itself

LMOLP (linear) $\quad \begin{cases} \operatorname{lex\,min}_x & c_1^T x, \dots, c_n^T x \\ Ax = b \\ x \geq 0 \end{cases} \iff \begin{cases} \min_x \sum_{i=1}^n c_i^T x \, \alpha^{1-i} \\ Ax = b \\ x \geq 0 \end{cases}$

· could use the NA-c-Simplex algorithm

→ simplex with non-standard cost function (same property eg. big-O)

→ transfer principle guarantees all the property of LP



$$\operatorname*{lexmax}_{x} \quad 8x_1 + 12x_2,\ 14x_1 + 10x_2$$
$$\text{s.t.} \quad Ax = b,$$
$$x \geq 0$$

$$\max_{x} \quad (8 + 14\eta)x_1 + (12 + 10\eta)x_2$$
$$\text{s.t.} \quad Ax = b,$$
$$x \geq 0$$

SIMPLEX PROBLEM what if a starting feasible solution is unknown

AUXILIARY PROBLEM

$Ax = b \quad \Rightarrow \quad \overset{m\times n}{A}x + \overset{m\times m}{I}s = b \quad \Rightarrow \quad \overset{m\times(n+m)}{[A \mid I]}\begin{pmatrix} x \\ s \end{pmatrix} = b$ $\quad (0,b)$ is feasible of the new problem

such that with other variables $=0$ $\quad A_B x_B = b$

setting $\vec{x} = \bar{0}$ we have that $B' = \{n+1, \dots, n+m\}$ is a valid base (column indexes of $[A|I]$ to get a $m\times m$ invertible matrix)

In fact $I^{-1} = I \quad \Rightarrow \quad s = b$

Two methods to make the original and auxiliary problem the same:

1) Two-Phase Method → I need to get $\bar{s} = \bar{0}$ to be feasible in the first problem

First-Phase: $\begin{cases} \min\limits_{x,s} \sum\limits_{i=1}^{m} s_i \\ Ax + Is = b \\ x, s \geq 0 \end{cases}$ If $s^* = \bar{0}$ move to phase 2. otherwise P is infeasible

- The starting base will be B'

Second-Phase: Solve P starting from previous x^*. $B = B'[1:n]$ ⇝ original variables

2) Big-M Method → try to make the two thing at once

$$\begin{cases} \min\limits_{x,s} \overset{\text{original}}{d^T x} + \overset{\text{penalty}}{M \sum\limits_{i=1}^{m} s_i} \\ Ax + Is = b \\ x, s \geq 0 \end{cases}$$

Th: $\exists M > 0$ sufficiently large : $\bar{x}' = [\bar{x}, 0]$ is the solution to the easier problem $\Leftrightarrow$ $\bar{x}$ is solution to the original one

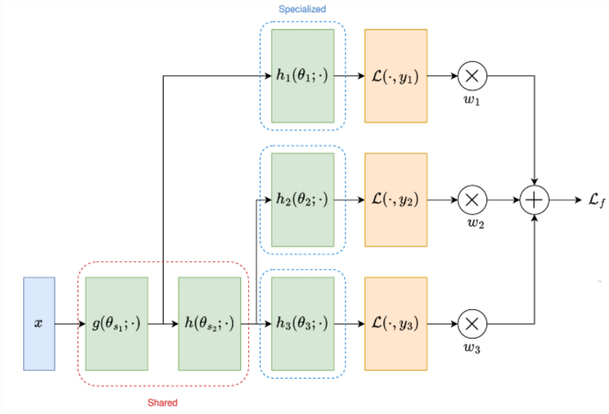
3) Infinite Big-M method (I-Big-M) M prioritize the zeroing of s over the minimization of $d^T x$
→ similar to lexicographic problem

set $M = \alpha$

- M is always large enough for equivalence → do not need to retry
- No numerical instabilities by very large M

HIERARCHICAL CLASSIFICATION → classification based on trees (hierarchy)

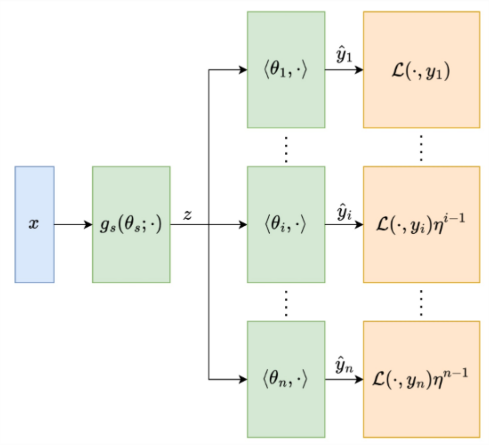
1) Branching DNN



$$\mathcal{L}_{B\text{-}DNN}(\theta) = \sum_{i=1}^{n} w_i(t)\, \mathcal{L}(f_i(\theta, x), y_i)$$

$$(w_1, w_2, w_3)\big|_{t=0} = (1,0,0]$$
$$(w_1, w_2, w_3)\big|_{t=\text{finish}} = [0,0,1]$$

2) Lexicographic DNN


x
$g_s(\theta_s;\cdot)$
z
$\langle\theta_1,\cdot\rangle$
$\hat{y}_1$
$\mathcal{L}(\cdot,y_1)$
$\langle\theta_i,\cdot\rangle$
$\hat{y}_i$
$\mathcal{L}(\cdot,y_i)\eta^{i-1}$
$\langle\theta_n,\cdot\rangle$
$\hat{y}_n$
$\mathcal{L}(\cdot,y_n)\eta^{n-1}$

lexmin $\mathcal{L}(f_1(\theta,x),y_1), \dots, \mathcal{L}(f_n(\theta,x),y_n)$

$\iff$ min $\sum_{i=1}^{n} L(f_i(\theta,x), y_i)\,\eta^{i-1}$

→ Cannot directly backpropagate otherwise updates of less important losses may worsen more important one

- perform a particular projection on gradient step to avoid it

Differences 1) Different depth splitting
2) Different losses

→ allows a NN to obtain a classification if unseen data

→ prioritized learning