

## Optimization theory

Parliamo nuovamente di ottimizzazione perché è alla base dei learning algorithms.

Abbiamo visto a DMGT i metodi di ottimizzazione quando la funzione obiettivo è convex e differentiable, nel caso non lo fosse possiamo usare i genetic algorithms.

Quando parliamo di multi-obj optimization siamo considerando un Multi-Agent System, cioè un sistema composto da più agents, i quali:

- Raccolgono info dall'ambiente
- Svolgono operazioni sull'ambiente
- Ricevono dei "premi" dall'ambiente

### Tip: di optimization algorithms

#### 1) Deterministic vs Stochastic

- Deterministic: Algoritmi che producono sempre lo stesso risultato, sono tipicamente local search
  - es: Newton method, ADAM, ... → eccezione: stochastic gradient descent
- Stochastic: Sono algoritmi basati sulla random-search
  - es: Simulated Annealing (sa), Genetic Algo → eccezione: space filling curves

#### 2) Local vs Global

#### 3) Single Agent vs Multi-Agent (Population Based)

#### 4) Heuristic vs Meta-Heuristic

- Heuristic: Algoritmi basati sul trial-and-error
- Meta-Heuristic: Tutti gli heuristic algorithms che sono ispirati alla natura
  - es: Genetic Algorithms, Swarm optimization ...

## 5) Single objective vs Multi-objective

Oss: Negli algoritmi basati sulla random search non facciamo riferimento alla "pure random search", perché sarebbe inefficiente dovrà alla ricerca uniforme sullo spazio di ricerca.

La random search si concentra sugli agenti più promettenti.

## Swarm Intelligence

Con Swarm Intelligence (Si) facciamo riferimento al comportamento collettivo di un sistema decentralizzato, self-organized, naturale o artificiale.

Tipicamente consiste in una popolazione di agenti (semplici) che interagiscono tra loro e con il loro ambiente seguendo semplici regole.

È fatto che non ci sia centralized control e con un certo grado di randomicità, l'interazione tra questi agenti produce una "intelligenza collettiva" che i singoli agenti non sarebbero in grado di riprodurre.

## Simulated Annealing

È un algoritmo stocastico, single agent, il quale consiste in una random search, cioè perturbare il punto corrente e valutare se la nuova posizione è migliore.

L'idea è di muoversi in un punto dove la funzione obj migliora, ma accettando possibili errori (punti con funzione obj peggiore) per i primi passi della random search.

Un punto peggiore viene accettato con una certa probabilità che diminuisce nel tempo. La diminuzione è gestita dalla funzione  $e^{-\Delta t}$  dove  $\Delta t$  gestisce la velocità della convergenza.



Il problema del SA algorithm è che è difficile implementare la perturbation perché è single-agent quindi non posso applicare uno scambio di info tra individui (come nel G.A), ma posso usare solo la mutation.

### Genetic Algorithms

I genetic algorithms, come abbiamo visto, sono algoritmi Meta-Heuristici e stocastici.

Gli individui possono essere rappresentati come stringhe binarie o come numeri reali.

La crossover operation permette di combinare due parenti in due discendenti, mixando il loro materiale genetico o facendone una weighted combination.

La mutation operation consiste nel mutare, con una certa probabilità, alcuni geni degli individui cambiando i loro valori con valori randomici (che siano conformi con il dominio dello specifico gene), questo permette di migliorare l'esplorazione.

L'elitism operation consiste nel propagare la miglior soluzione trovata nella successiva popolazione, nel caso dei multi-objective Genetic Algorithms non c'è il concetto di miglior soluzione ma solo il concetto di "non-dominated solution" le quali vengono propagate.

↓  
Solutions che hanno un migliore fitness value per tutte le objective functions rispetto agli altri individui della popolazione

### GA algorithms types

1) Steady state + mutation (più semplice)

a) Crea una popolazione iniziale e valuta la loro fitness

b) Finché una certa condizione non è rispettata (ad esempio il numero di generazione ...)

- Scegli in modo randomico un elemento dalla popolazione corrente (o proporzionale alla fitness)
- Mutalo e valuta la fitness

- Aggiungilo alla popolazione
- Rimuovi l'individuo con la fitness peggiore

2) Steady State + crossover + Mutation  $\rightarrow$  Usa Elitism

- a) Crea una popolazione iniziale e valuta la loro fitness
- b) Finché una certa condizione non è rispettata
  - Scegli in modo randomico due elementi dalla popolazione corrente (o proporzionale alla fitness)
  - Applica la crossover operation per generare due discendenti
  - Mutali e valuta le loro fitness
  - Aggiungi alla popolazione
  - Rimuovi: due individui con la fitness peggiore

3) Not Steady State + crossover + Mutation (canonical GA)  $\rightarrow$  Usa elitism

- a) Seta  $t=0$ , crea una popolazione iniziale ( $P_t$ ) e valuta la loro fitness
- b) Finché una certa condizione non è rispettata
  - Crea una nuova popolazione di discendenti ( $Q_t$ ) da  $P_t$ , della stessa dimensione, usando la crossover operation
  - Valuta la fitness degli elementi in  $Q_t$
  - Calcola  $R_t$  come l'unione tra  $P_t$  e  $Q_t$  ( $R_t$  ha il doppio della dimensione)
  - Crea  $P_{t+1}$  da  $R_t$  rimuovendo i peggiori individui (population-size) da  $R_t$

È lo schema usato da NSGA-II il quale è un multi-obj genetic algorithm

### Come Generare $Q_t$ da $P_t$

Per generare  $Q_t$  da  $P_t$  possiamo usare il seguente algoritmo.

Finché la dimensione di  $Q_t$  è minore della population-size

- 1) Seleziona il primo genitore tramite un **binary tournament selection** (BTS)
- 2) Seleziona il secondo genitore tramite un **binary tournament selection** (BTS)
- 3) Genera i due discendenti usando la crossover operation
- 4) Mutta i discendenti
- 5) Aggiungi i due nuovi discendenti a  $Q_t$

La **Binary tournament selection** consiste nello scegliere due individui random da  $P_t$  e:

- Mantenere quello con fitness migliore, se entrambi sono feasible (seleziona random se pareggio)
- Mantenere quello feasible, se l'altro non è feasible
- Mantenere quello che viola meno vincoli, se entrambi non sono feasible.

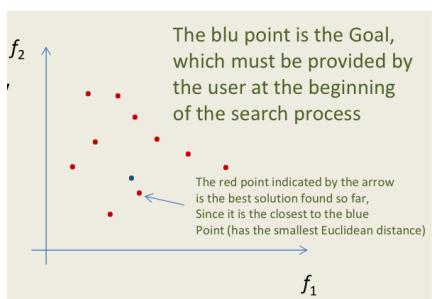
Utile soprattutto per i **constrained problems** per preferire feasible points rispetto ai non feasible.

### GA convergence

Genetic Algorithms non hanno una prova di convergenza, ma solo congettura sul perché sono così efficaci nel trovare l'ottimo globale in un tempo ragionevole.

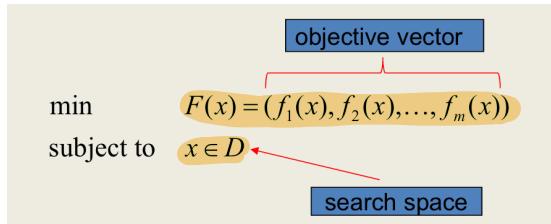
### Multiobjective Optimization Problem

I multiobjective optimization problems sono diversi dal "goal programming" il quale consiste nel minimizzare la distanza tra la migliore soluzione e l'obiettivo prefissato dall'utente. Nel goal programming le obj. functions non vengono ottimizzate simultaneamente



ma aggregato in una singola objective function.

In multiobjective optimization problems sono così definiti:

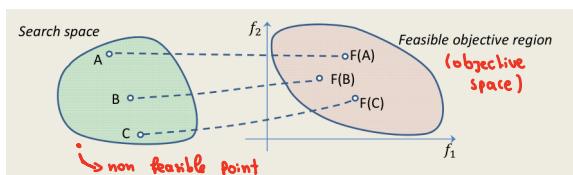


dove:  $x$  può essere discreto, continuo o una mixed variable

### Come comparare le soluzioni

Sia il nostro search space definito dai vincoli e l'objective space definito dalle funzioni obiettivo,  $y$  domina  $x$  se:

- $y$  non è peggiore di  $x$  per qualsiasi obj function
- $y$  è meglio di  $x$  in almeno una obj function

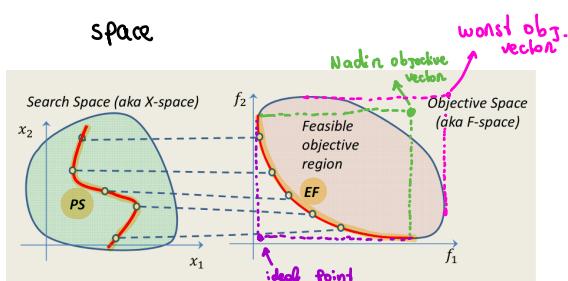


sempre peggiori  $\rightarrow$  **scartarla**

→  
 1)  $B$  domina  $A$  (perché stiamo minimizzando)  
 2)  $B$  e  $C$  non sono comparabili perché  
 $f_1(B) < f_1(C)$  e  $f_2(C) < f_2(B)$   
 $\downarrow$   
 $B$  e  $C$  sono **non-dominated solutions**

OSS: Ci possono essere vincoli anche sull'objective space!

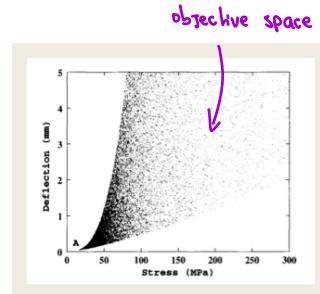
- $x$  è un **Pareto optimal** (soluzione ottimale del MOP) se è una **soluzione non-dominata**
- Il **Pareto set** (PS) è l'insieme dei Pareto optimal nel search space
- l' **Efficient set** (ES) è l'insieme delle immagini del Pareto set sull'objective space



Attenzione: Quello che cerchiamo solitamente è il **Pareto optimal set** (nel search space).

## Conflicting objectives

Multi-objective optimisation ha senso solo quando le funzioni obiettivo sono in conflitto perché altrimenti abbiamo una singola soluzione ottimale.



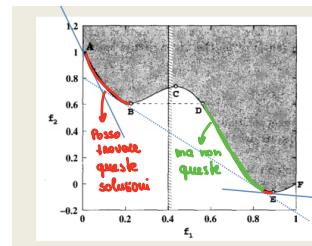
Noi siamo interessati ai conflicting problems.

## Limits of weighted sum approach (scalarization)

Il problema dello scalarization problem è che bisogna fornire i pesi prima dell'ottimizzazione ed è difficile pesare le funzioni correttamente perché ogni funzione obiettivo ha il suo range (non conosciuto all'inizio) e quando fissiamo i pesi, muoviamo l'objective space in una direzione specifica determinata dal ratio tra i pesi.

Pensando quando l'efficient set è convex è possibile trovare tutti i pareto optimal con differenti combinazioni dei pesi.

La situazione peggiora quando l'efficient set non è convex perché non è possibile trovare tutte le soluzioni ottimali

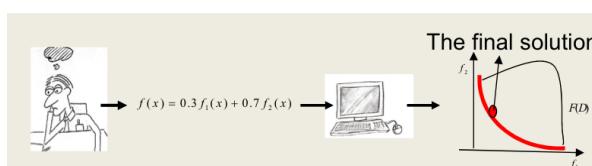


## Strategies in Multi-objective Decision Making

Quando abbiamo un multi-objective problem abbiamo bisogno di un decision maker il quale deve decidere quanto pesare le objective functions.

- 1) A-priori: Il decision maker decide quanto pesare le obj functions a priori  
(prima dell'esecuzione dell'optimiser)

→ **Scalarization**



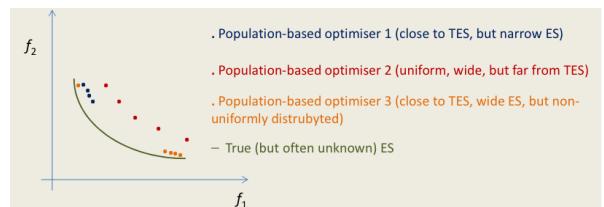
2) **A posteriori**: Viene eseguito l'optimiser, viene fornito al decision maker l'insieme delle Pareto optimal solutions il quale decide quale è la soluzione che preferisce.

3) **Interactive**: Il decision maker dà feedback durante l'ottimizzazione

### Obiettivi dei Multi-objective optimisers

- 1) Trovare soluzioni il più vicino possibile all' efficient set (non posso trovarlo tutto perché è continuo, cioè composto da infiniti punti)
- 2) Cercare di raggiungere, il più possibile, gli estremi dell' efficient set
- 3) Trovare soluzioni che siano il più possibile distribuite uniformemente lungo l' efficient set.

- Optimiser 1 ha soluzioni vicine a ES ma non uniformemente distribuite
- Optimiser 2 ha soluzioni uniformemente distribuite ed è in grado di approssimare l'inizio e la fine di ES ma le soluzioni non sono vicine a ES
- Optimiser 3 ha soluzioni vicine a ES, è in grado di approssimare l'inizio e la fine ma non sono uniformemente distribuite.



OSS: Usiamo algoritmi population based perché restituiscono un insieme di soluzioni diverse con un'unica esecuzione. Se usassimo algoritmi non population based, dovremmo eseguire l'algoritmo più volte usando differenti pesi (scalarisation).

### External archives

Sono strutture dati aggiuntive usate per salvare l'insieme delle non-dominated strategies trovate durante il processo di ottimizzazione. Possono essere bounded o unbounded.

## Multi-objective Genetic Algorithms (MOGA)

Le MOGA si dividono in 3 famiglie:

1) Pareto - Dominance based algorithms

↳ NSGA-II, PAES, ...

2) Indication-based algorithms

↳ SMS-EMOA, HypE, CMA-ES ...

3) Decomposition-based algorithms

↳ MOEA/D ...

### NSGA-II

È la versione migliorata del NSGA, è elitist e non usa archivi esterni.

Algoritmo:

1) Dalla popolazione  $P_t$  crea i discendenti (con l'operazione di crossover)  $R_t$ .

La loro unione produce  $R_t$  la quale è composta da  $2 \times \text{pop-size}$  elementi.

2) Dalla popolazione  $R_t$  devono essere scelti pop-size individui.

Per fare ciò si usano 2 operazioni:

a) Non-dominated sorting

Ondina gli individui sulla base di un rank.

Prima Frontiera di Pareto



• Trova tutte le non-dominated solutions in  $R_t$ , le quali formano  $F_1$

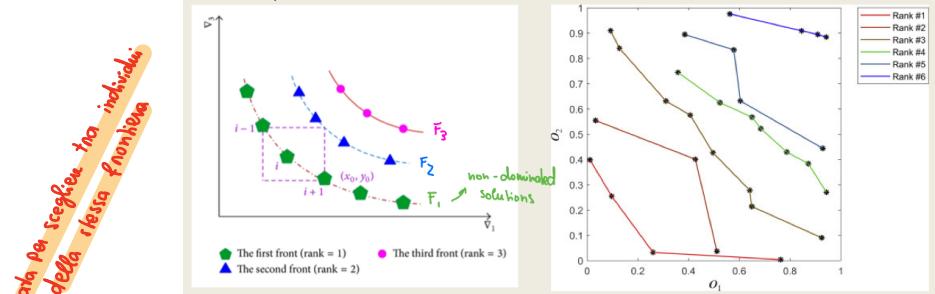
• Rimuove tutte le soluzioni in  $F_1$  da  $R_t$  e cerca nuovamente le non-dominated solutions, le quali formeranno  $F_2$ , e così via.

Se ad esempio devo prendere pop-size = 8 individui e  $|F_1|=6$  e  $|F_2|=4$ ,

prendo tutti gli individui da  $F_1$  e poi devo decidere quali 2 individui

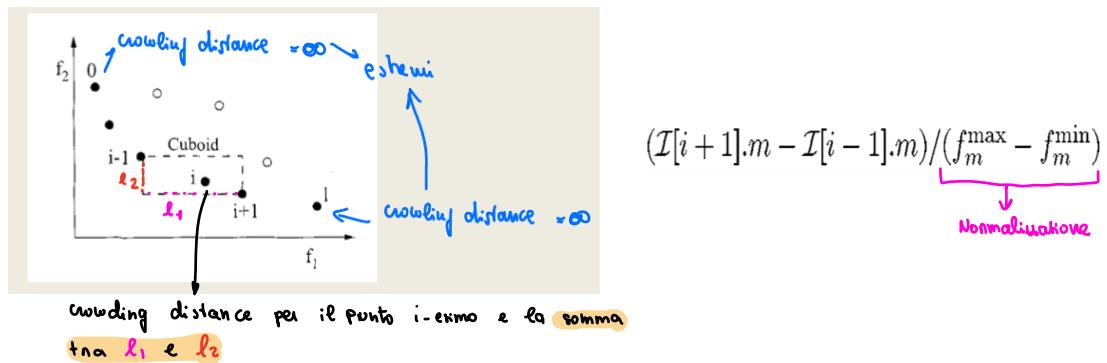
scegliere da  $F_2$ .

per fare ciò uso la 2° operazione chiamata Crowding distance Sorting.



### b) Crowding Distance Sorting

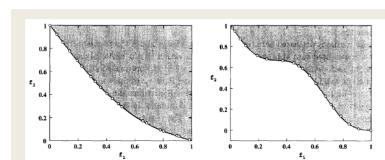
La crowding distance viene calcolata separatamente per ogni frontiera e, per il punto  $i$ -esimo, è la somma delle distanze tra le soluzioni adiacenti al punto su ogni funzione obiettivo.



Vengono scelte le soluzioni con crowding distance maggiore perché indica che la soluzione è meno "affollata" rispetto alle altre sullo stesso fronte.

Questo ci permette di avere soluzioni uniformemente distribuite.

NSGA-II lavora bene sia su efficient set convessi  
 che non convessi.



## PAES

è un MOGA il quale è mutation only (algoritmi chiamati anche Evolutionary Strategies) e usa archivi esterni.

In un  $(\mu + 1)$ -PAES vengono ottenuti  $\lambda$  offspring dalla mutazione di  $\mu$  parents.

### Algoritmo $(\mu + 1)$ -PAES

- 1) Genera una random solution iniziale  $c$  e aggiungi la all'archivio
- 2) Mutta  $c$  per produrre  $m$  e valuta  $m$ 
  - a) Se  $c$  domina  $m$ , scarta  $m$
  - b) Se  $m$  domina  $c$ , scarta  $c$  (e tutte le soluzioni nell'archivio dominate da  $m$ ) e aggiungi  $m$  all'archivio
  - c) Se  $m$  è dominato da un qualsiasi membro dell'archivio, scarta  $m$
  - d) Se  $m$  non domina  $c$ , e viceversa, applica la test( $c, m, archive$ ) function per determinare quale elemento diventa la current solution da mutare e, inoltre, se aggiungere  $m$  all'archivio.

### Test ( $c, m, archive$ ) Function

- 1) Se l'archivio non è pieno
  - a) Aggiungi  $m$  all'archivio  $\nearrow$  crowded distance
  - b) Se  $m$  è in una less crowded region nell'archivio rispetto a  $c$ , accetta  $m$  come current solution.

Altimenti manterrà  $c$  come current solution.

- 2) Se l'archivio è pieno

- a) Se  $m$  è in una less crowded region nell'archivio rispetto a  $x$ , dove  $x$  è un qualsiasi elemento dell'archivio,
- Aggiungi  $m$  all'archivio e rimuovi uno degli elementi dalla regione in cui è contenuto  $x$
  - Se  $m$  è in una less crowded region nell'archivio rispetto a  $c$ , accetta  $m$  come current solution.
- Altrimenti mantieni  $c$  come current solution.
- b) Altrimenti, se  $m$  è in una less crowded region nell'archivio rispetto a  $c$ , rimuovi  $c$  ed accetta  $m$  come current solution. Viceversa, scatta  $m$  e mantieni  $c$  come current solution.

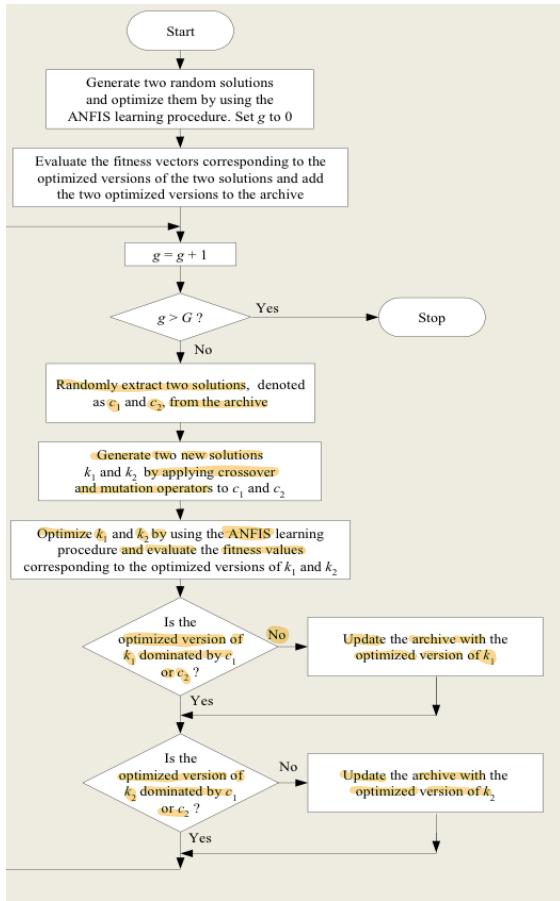
Attention:

- Durante l'esecuzione dell'algoritmo test, l'external archive è suddiviso in celle e la minore crowded region viene considerata su queste celle.  
La dimensione della griglia è dinamica per poter accettare soluzioni non-dominanti che "cascano" al di fuori della griglia, ed anche per poter ridurre la sua dimensione man mano durante le varie iterazioni (perché le soluzioni tenderanno a spostarsi verso la parte in basso a sx dell'objective space)
- Viene mantenuta anche una map della griglia, la quale indica per ogni cella quante e quali soluzioni dell'archivio risiedono in essa.

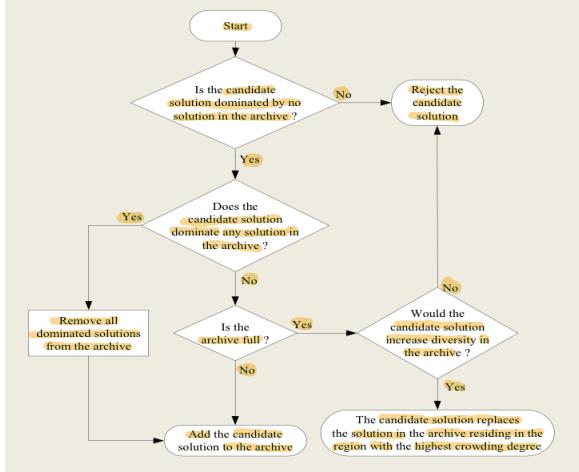
## Algoritmo (2+2) M-PAES

È un algoritmo sviluppato a Pisa che utilizza due soluzioni,  $c_1$  e  $c_2$ , prese a random dall'archivio ad ogni iterazione.

Due offspring vengono generati da esse tramite crossover, mutazioni e confrontati con  $c_1$  e  $c_2$ .



The test procedure for the (2+2)M-PAES



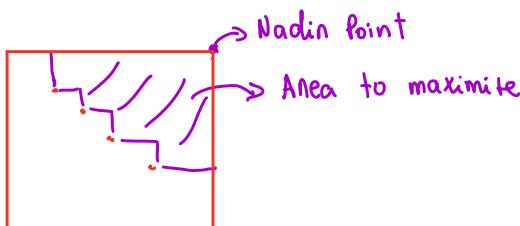
## Limiti di PAES

La gestione dell'Adaptive grid può essere complicata e computationalmente costosa, anche se in alcuni casi l'obj space è conosciuto (es ROC space) e può essere usata una griglia statica.

## SMs - EMOA

SMs - EMOA è un indicator-based algorithm il quale è basato sull'ottimizzazione dell' hypervolume, anche chiamato S-measure.

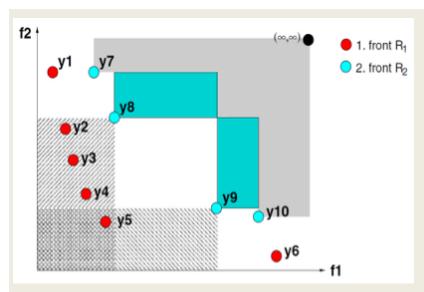
L'idea è di massimizzare l'area formata tra la pareto frontier e il Nadir point, perché man mano nella generazione nuove soluzioni non dominate portano ad un'area maggiore. Quando dobbiamo scegliere quali punti selezionare per la prossima generazione, scegliamo quelli che aumentano maggiormente l'area.



- Efficient algorithm
- Usa concetto di Non-dominated sort
- Solo uno scalare (Hypervolume) da confrontare tra due o più optimizers per capire il migliore

## Algoritmo

- 1) Genera  $q_t$  da  $P_t$ , unisci per generare  $R_t$  e scegli n° individui
- 2) Per scegliere individui da  $R_t$ 
  - Dividi individui in frontiere
  - Inserisci tutte le soluzioni nelle frontiere più importanti
  - Se c'è da scegliere tra individui della stessa frontiera, scegli quelli che massimizzano l' hypervolume.



Es priority:

$y_7, y_{10}, y_8, y_9$

Problema: Calcolare Hypervolume è computationalmente pesante

## Hype

Hype è un indication-based algorithm, simile a SMS-EMOA, il quale calcola

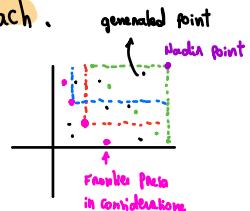
l' hypervolume approssimativamente usando un MonteCarlo approach.

- 1) Genera punti random (all'interno della box delimitata dal nadir point)

- 2) Valuta il ratio dei punti che finiscono all'interno

della box rispetto a quelli che finiscono fuori (Box = soluzioni dominate dalla soluzione presa in considerazione)

- 3) Ottengo una stima dell'importanza della soluzione



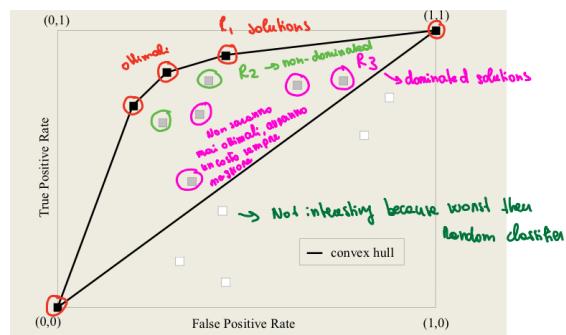
## CHEA

CHEA è un indication-based algorithm sviluppato in Pisa, su misura per lavorare sul ROC space.

Esso cerca di ottimizzare l'Area under the Convex Hull della ROC curve.

Solo i punti che sono sul convex hull della ROC curve sono ottimali.

- 1) Priorità maggiore ai punti sul convex hull
- 2) Second priority alle altre soluzioni non dominate
- 3) Third priority alle soluzioni dominate



## MOEA/D

MOEA/D è un decomposition based algorithm, cioè il multi-obj problem originale viene suddiviso in N single-obj sottoproblemi.

L'idea è che ogni sottoproblema fa riferimento a una singola search direction ed essi vengono risolti collaborativamente, cioè la miglior soluzione attuale di un subproblem può

influenzante la migliore soluzione di un subproblem NEL SUO VICINATO.

- Ogni subproblem ha un insieme di near subproblems con cui collaborare.
- Usa sia la popolazione corrente, sia un unbounded external archive formato dalle soluzioni non-dominante trovate fino a quel momento.

### Decomposition Strategy

MOEA/D usa  $N$  lambda vectors (search directions) fornite dall'utente.

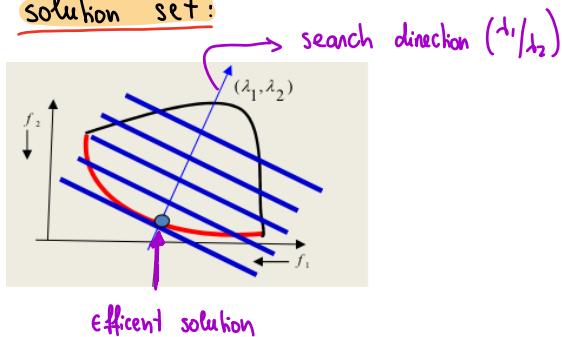
Ogni combinazione  $\lambda$  forma un subproblem in questo modo: ( $N$  subproblems)

$$\min g^{ws}(x, \lambda) = \lambda_1 f_1(x) + \lambda_2 f_2(x) \quad \rightarrow \text{weighted sum}$$

where  $\lambda_1 + \lambda_2 = 1$  and  $\lambda_1, \lambda_2 \geq 0$ .

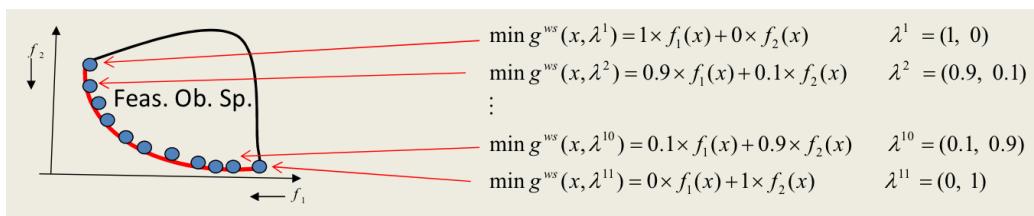
Per ogni problema trova una efficient solution, la quale appartiene all'efficient

solution set:



del Multi obj problem

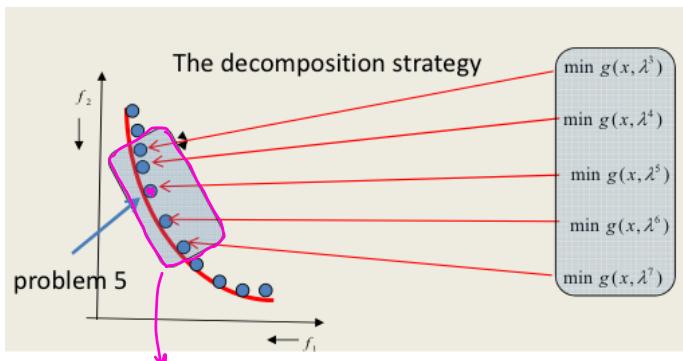
Risolvendo tutti gli  $N$  problemi ottengo  $N$  soluzioni che appartengono all' efficient set.



È possibile utilizzare la cheby shev scalarization al posto della weighted sum, la quale funziona meglio quando l'efficient set è non-convex.

## Neigh boundood

Due problemi sono vicini se i loro weight vectors sono vicini, cioè la distanza euclidea tra le loro due search directions è piccola.



es: Neighbourhood of \*

## Collaboration

Ad ogni generazione, ogni sub-problem fa le seguenti operazioni:

- 1) **Mating selection**: Randomicamente sceglie due vicini e ottiene le loro soluzioni
- 2) **Reproduction**: Genera una nuova soluzione applicando (crossover + mutation)  
sulle soluzioni ottenute dai suoi vicini
- 3) **Replacement**:
  - Rimpiazza la sua soluzione attuale con quella generata, se ha una migliore fitness function
  - Passa la nuova soluzione ai suoi vicini (tutti), i quali rimpiazzano le loro soluzioni con questa qua, se migliore per la loro obj function.

## Algorithm

Input:  $N, \lambda_1, \dots, \lambda_N$ , Neighborhood size ( $T$ )

- At each Generation, MOEA/D maintains two things:

1) A population of  $N$  solutions  $x_1, \dots, x_N$ , where  $x_i$  is the current solution to the  $i$ th subproblem, and  $F(x_1), \dots, F(x_N)$ .

2) Ideal point  $z = (z_1, \dots, z_m)$ , where  $z_j$  is the best value found so far for objective  $f_j$ ,  $\forall j = 1, \dots, m$ .

$\leftarrow z$  is the current ideal point

### • Step 1 (initialization)

- Step 1.1:** Compute the Euclidean distances between any two weight vectors and then determine the  $T$  closest weight vector to each  $\lambda_i$ . For each  $i = 1, \dots, N$ , set  $B(i) = \{i_1, \dots, i_T\}$ , where  $\lambda_j, \forall j \in B(i)$  are  $T$  closest vectors to  $\lambda_i$ .
- Step 1.2:** Randomly generate the initial population.
- Step 1.3:** Evaluate  $F(x_i) \quad \forall i = 1, \dots, N$ .
- Step 1.4:** Initialize  $z$ : the initial ideal point  $(z_1, \dots, z_m)$  according to the condition:  $z_j = \min_{1 \leq i \leq N} f_j(x_i) \quad \forall j = 1, \dots, m$ .

### • Step 2 (Update)

For  $i = 1, \dots, N$ , do

- Step 2.1 (Reproduction):** Randomly select two indices  $k$  and  $l$  from  $B(i)$ , and generate a child solution  $x_{\text{child}}$  from parents  $x_k$  and  $x_l$  by applying genetic operators.
- Step 2.2 (Repair):** Repair  $x_{\text{child}}$  by applying problem-specific repair/improvement heuristic.
- Step 2.3 (Function Evaluation):** Evaluate  $F(x_{\text{child}})$ .
- Step 2.4 (Update of  $z$ ):** For each  $j = 1, \dots, m$ , if  $z_j > f_j(x_{\text{child}})$  then set  $z_j = f_j(x_{\text{child}})$ .
- Step 2.5 (Replacement/Update of Solutions):** For each index  $j \in B(i)$ , if  $g^c(x_{\text{child}}|z_j, z) \leq g^c(x_j|z_j, z)$  then set  $x_j = x_{\text{child}}$  and  $F(x_j) = F(x_{\text{child}})$ .

← Repair is needed only in constrained optimization, when  $x_{\text{child}}$  violates some of the constraints

### • Step 3 (stopping criterion)

If termination criterion is satisfied, then obtain approximation Pareto Set solutions  $\{x_1, \dots, x_N\}$  and ES:  $\{F(x_1), \dots, F(x_N)\}$ , else go to Step 2.

Come comparare le performance di due MOEA?

Per comparare le performance possono essere utilizzati più modi:

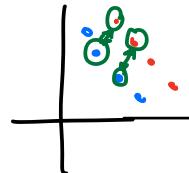
1) Hypervolume

2) Generational distance (GD) metric

È calcolata tramite la distanza euclidea tra i punti forniti dall'optimizer e il reference set

$$GD(A) = \frac{1}{n} \left( \sum_{i=1}^n d_i^p \right)^{\frac{1}{p}} \rightarrow \text{sum of the min distances}$$

↳ Avg distance



Essa necessita dei reference points, i quali è possibile ottenere dalla popolazione finale di un optimisation algorithm considerando solo le non-dominated solutions, e misura quanto le soluzioni fornite dall'optimizer in considerazione sono vicine ai reference points.

3) Inverted Generational Distance

Invece di considerare la distanza dalle soluzioni ai reference points, considera la distanza dai reference points alle soluzioni.

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

Essa misura quanto well-spread sono le soluzioni rispetto ai reference points (come sono distribuite le soluzioni nell' obj. space).

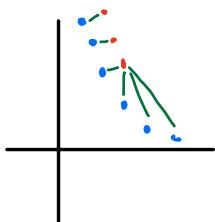
Naturalmente, anche questa metoda ha bisogno dei reference points.

#### 4) Delta+

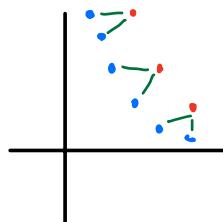
Il Delta+ è il massimo tra la GD e l' IGD.

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

Questa misura prende come riferimento la peggiora tra GD e IGD, quindi è un buon indicatore di confronto essendo che prende in considerazione entrambe.



Buono GD ma  
Pessima IGD



Pessimo GD, ma  
buon IGD

#### Many - Objective optimization

In many-obj problems sono problemi con un gran numero di obj functions.

##### Summary

- Multi-Objective: from 2 and 3 objectives
- Many-Objective: from 4 to 30
- Massive-Objective: >30

Essi necessitano di algoritmi specifici (NSGA-II non funziona) perché:

- 1) Le soluzioni sono per lo più tutte non-dominated, quindi il Non-dominated Front ranking non funziona (tutte le soluzioni nella prima frontiera)
- 2) Difficile usare crowding distance <sup>on hypervolume</sup> perché computazionalmente costoso (cause of dim.)

### 3) Visualizzazione efficient set difficile

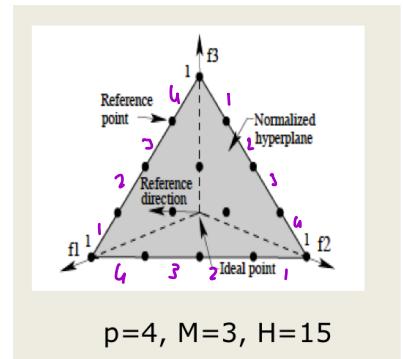
#### NSGA-III

Evoluzione NSGA-II per many-obj problems, basato su un insieme di reference points forniti dall'utente.

È un algoritmo composto da 4 fasi più uno step iniziale (step 0):

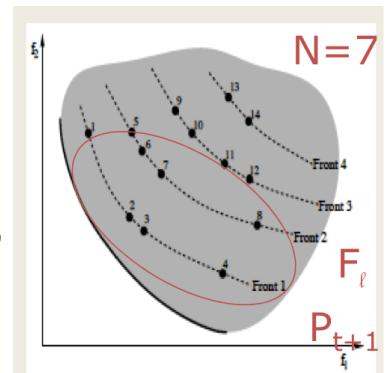
- Vengono usati i reference point dati dall'utente oppure viene usato il DAS and Dennis approach per crearli:

- Il numero di reference points è dato da:  $H = \binom{M+p-1}{p}$  dove  $H = n^o$  obj functions e  $p =$  numero di divisioni per ogni obj function axis.
- I punti vengono creati sul normalized hyperplane distribuiti uniformemente su esso.
- Ad ogni reference point è associata una reference direction (search direction)



- Vengono identificate le Non-dominated Frontiers

- $P_t$  e  $Q_t$  combinate in  $R_t$
- $R_t$  diviso in frontiere
- Presi gli elementi dalle frontiere con priorità ( $s_t$ ), per scegliere i rimanenti all'interno di una frontiera non viene usata la crowding-distance



OSS: In  $F_t$  nei many-obj problems ci sono più soluzioni rispetto ai multi-obj problems, come detto precedentemente

2) le obj function vengono normalizzate, rescale tra 0 e 1

- Calcola ideal point per ogni obj function dato  $S_t$ , cioè la soluzione in  $S_t$  che da il minimo valore per quella funzione
- Ogni soluzione in  $S_t$  viene aggiornata rispetto all'ideal point di ogni f. obiettivo, cioè  $f_j(s) - z_j$   $\forall s \in S_t$ , per rendere  $z_j$  l'origine rispetto a quella obj function.
- Trovo i punti estremi per ogni obj function dato  $S_t$ , cioè la soluzione in  $S_t$  che da il massimo valore per quella funzione
- Normalizzo i punti nella popolazione dividendo il loro valore per quella obj function rispetto alla differenza tra il suo extreme point e ideal point

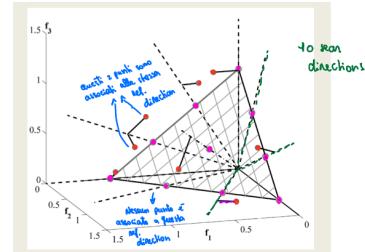
```

Algorithm 2  $f^* = \text{Normalize}(S_t, Z')$ 

Input :  $S_t, Z'$ 
Output :  $f^*$  // the reference points on normalized hyper-plane
1: for  $j=1$  to  $M$  do
2:   Compute ideal point:  $z_j^{\min} = \min_{s \in S_t} f_j(s)$ 
3:   Traslate objectives:  $f_j(s) = f_j(s) - z_j^{\min} \quad \forall s \in S_t$ 
4:   Compute extreme points:  $z^{j\max}, j=1,\dots,M$  of  $S_t$ 
5: endfor
6: Compute intercepts  $a_j$  for  $j=1,\dots,M$ 
7: Compute the normalized objectives  $f^*$  using the equation:

$$f_j^*(x) = \frac{f_j}{a_j} \quad \text{for } j=1,\dots,M$$


```



3) Associone ogni individuo in  $S_t$  alla reference direction più vicina.

- Ogni reference direction può avere zero, uno o più di un membro di  $S_t$  associato
- Conto per ogni reference direction quanti individui di  $S_t$  sono stati associati
- Quando devo scegliere quali punti prendere dalla frontiera, scelgo quelli la cui reference distance più vicina ha meno punti associati di  $S_t$ .  
Questo perché mi permette di avere individui che "cercano" in quella direzione.  
Se nessun punto può essere associato ad una search direction, l'idea è di rimuoverla.

OSS:

- Non necessita di parametri (a differenza ad esempio di MOEA/D che vuole  $T$ )
- Selection è randomica, ma se ci sono dei constraints viene reinhardtato il tournament selection operator

### Mixed Pareto-Lexicographical Problems (Multi-obj)

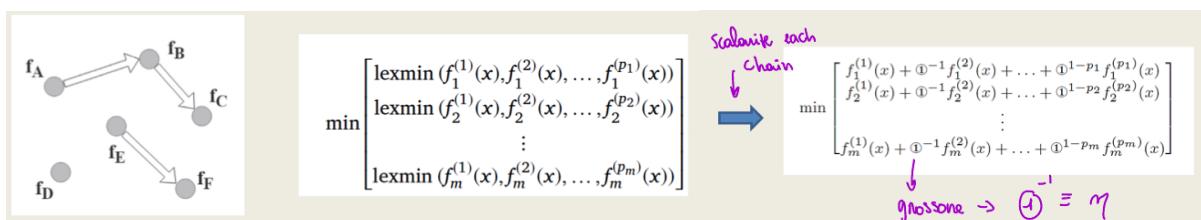
Siano: pure lexicographical multi-obj opt. problems così definiti:

$$\begin{array}{l} \text{Lexmin } f_1(x), f_2(x), \dots, f_M(x) \\ \text{Subject to } x \in \Omega \end{array} \iff \begin{array}{l} \min_{x \in \Omega} f_1(x) + \eta f_2(x) + \eta^2 f_3(x) \dots \end{array}$$

dove minimizzazione  $f_1(x)$  infinitamente più importante di minimizzazione  $f_2(x)$  ...

#### Priority chain (PC) case

Sono problemi dove sappiamo dell'esistenza di priorità tra alcune obj. functions ma non è chiara la priorità tra ognuna di esse (non c'è una singola catena di precedenze).



Per affrontare questo problema, bisogna definire la priority chain dominance.

Date due soluzioni  $A$  e  $B$ ,  $A$  domina  $B$  ( $A \leq B$ ) se

$$A \leq B \iff \begin{cases} f_i(x_A) \leq f_i(x_B) & \forall i = 1 \dots m \\ \exists j : f_j(x_A) < f_j(x_B) \end{cases}$$

## Illustrative problem: the knapsack problem

<i>Classical formulation, one objective</i>	$\max Vx$ $s.t.$ $Wx < C$ $x_i \in \{0,1\}$	
<i>Pure Paretian formulation, two objectives</i>	$\max V_1x, V_2x$ $s.t.$ $Wx < C$ $x_i \in \{0,1\}$	
<i>Pure Lexicographic formulation, two objectives</i>	$\text{lexmax } V_1x, V_2x$ $s.t.$ $Wx < C$ $x_i \in \{0,1\}$	$\max V_1x + V_2x \cdot \mathbb{D}^{-1}$ $s.t.$ $Wx < C$ $x_i \in \{0,1\}, \mathbb{D}^{-1} \text{ is an infinitesimal}$

## Illustrative problem (MPLO)

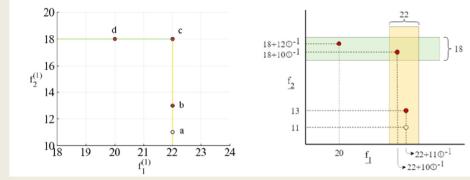
$\max \left[ \frac{f_1(x)}{f_2(x)} \right]$	This is a 0-1 Knapsack problem, with 5 variables and two main objectives (two chais). Each objective has one secondary objective:
$s.t.$ $Wx \leq C$	
$f_1(x) = V_1x$	<b>First chain →</b>
$f_2(x) = V_2x$	<b>Second chain →</b>
$V_1 = V_1^{(1)} + \mathbb{D}^{-1}V_1^{(2)}$	$V_1^{(1)} = [4, 5, 6, 7, 11]^T$
$V_2 = V_2^{(1)} + \mathbb{D}^{-1}V_2^{(2)}$	$V_2^{(1)} = [1, 2, 3, 4, 6]^T$
$V_1^{(1)} = [4, 5, 6, 7, 11]^T$	$V_2^{(1)} = [7, 8, 2, 1, 3]^T$
$V_2^{(1)} = [4, 3, 2, 1, 5]^T$	$V_1^{(2)} = [1, 2, 3, 4, 6]^T$
$W = [1, 2, 3, 4, 5]^T$	$V_2^{(2)} = [4, 3, 2, 1, 5]^T$
$C = 10$	$V_1^{(1)} \text{ might be the profit}$
$x_i \in \{0,1\}$	$V_1^{(2)} \text{ might be the portion of the profit coming from investments in renewable energies}$
	$V_2^{(1)} \text{ might be the (opposite of the) risk}$
	$V_2^{(2)} \text{ might be the (opposite of the) risk associated to non-government bonds}$
	$i = 1, \dots, 5$

## Feasible, unfeasible and saturating solutions

$x = [00000] w^T x = 0 \leq 10$	$x = [10000] w^T x = 1 \leq 10$
$x = [00001] w^T x = 5 \leq 10$	$x = [10001] w^T x = 6 \leq 10$
$x = [00010] w^T x = 4 \leq 10$	$x = [10010] w^T x = 5 \leq 10$
$x = [00011] w^T x = 9 \leq 10$	$x = [10011] w^T x = 10 \leq 10 \text{ (a)}$
$x = [00100] w^T x = 3 \leq 10$	$x = [10100] w^T x = 4 \leq 10$
$x = [00101] w^T x = 8 \leq 10$	$x = [10101] w^T x = 9 \leq 10$
$x = [00110] w^T x = 7 \leq 10$	$x = [10110] w^T x = 8 \leq 10$
$x = [00111] w^T x = 12 \leq 10$	$x = [10111] w^T x = 13 \leq 10$
$x = [01000] w^T x = 2 \leq 10$	$x = [11000] w^T x = 3 \leq 10$
$x = [01001] w^T x = 7 \leq 10$	$x = [11001] w^T x = 8 \leq 10 \text{ (d)}$
$x = [01010] w^T x = 6 \leq 10$	$x = [11010] w^T x = 7 \leq 10$
$x = [01011] w^T x = 11 \leq 10$	$x = [11011] w^T x = 12 \leq 10$
$x = [01100] w^T x = 5 \leq 10$	$x = [11100] w^T x = 6 \leq 10$
$\rightarrow x = [01101] w^T x = 10 \leq 10 \text{ (b)}$	$x = [11101] w^T x = 11 \leq 10$
$x = [01110] w^T x = 9 \leq 10$	$x = [11110] w^T x = 10 \leq 10 \text{ (c)}$
$x = [01111] w^T x = 14 \leq 10$	$x = [11111] w^T x = 15 \leq 10$

## Analysing 4 interesting solutions

	$x$	$f_1(x)$	$f_2(x)$	Pareto optimal
a	10011	$22 + 11\mathbb{D}^{-1}$	$11 + 10\mathbb{D}^{-1}$	no
b	01101	$22 + 11\mathbb{D}^{-1}$	$13 + 10\mathbb{D}^{-1}$	yes
c	11110	$22 + 10\mathbb{D}^{-1}$	$18 + 10\mathbb{D}^{-1}$	yes
d	11001	$20 + 9\mathbb{D}^{-1}$	$18 + 12\mathbb{D}^{-1}$	yes



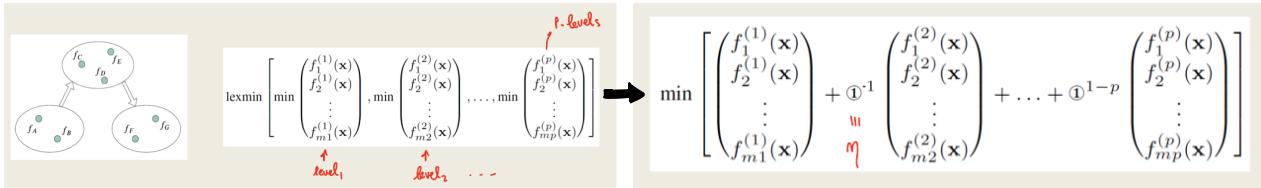
## Comments and the PC\_crowding\_distance

- Solutions (a) and (b) are equivalent for the first macro-objective  $f_1$ , but (b) is clearly more valuable for  $f_2$ , therefore (b) dominates (a)
- (b) is slightly better than (c) for  $f_1$  because its secondary objective is higher (the secondary is taken into account since they are equivalent for the primary). On the other hand, (c) is better than (b) on the second macro-objective. Thus (b) and (c) do not dominate each other
- Similarly, (c) and (d) happen to be nondominated. This time, the second secondary plays an active role
- (b) and (d) are not PC-dominated too, and secondaries do not even have to be considered (because primaries differ)

	$x$	$f_1$	$f_2$	PC.crowding.distance
a	10011	$22 + 11\mathbb{D}^{-1}$	$11 + 10\mathbb{D}^{-1}$	①
b	01101	$22 + 11\mathbb{D}^{-1}$	$13 + 10\mathbb{D}^{-1}$	$1 - 0.286\mathbb{D}^{-1}$
c	11110	$22 + 10\mathbb{D}^{-1}$	$18 + 10\mathbb{D}^{-1}$	$1.714 + 0.082\mathbb{D}^{-1}$
d	11001	$20 + 9\mathbb{D}^{-1}$	$18 + 12\mathbb{D}^{-1}$	①

## Priority Levels (PL) case

In questo caso le obj. function sono raggruppate in differenti livelli di priorità



Anche in questo caso dobbiamo definire la priority levels dominance.

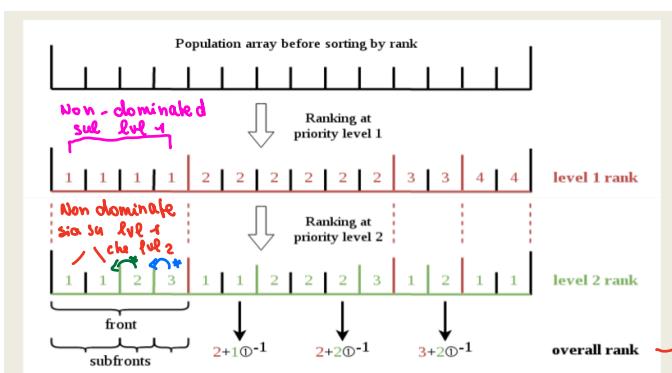
Date due soluzioni  $A$  e  $B$  che appartengono alle sub-fronts  $F_i$  e  $F_j$  rispettivamente,

$A$  PL-domina  $B$  ( $A \leq^* B$ ) se  $i < j$

Questa definizione può essere usata solo a posteriori, cioè non è possibile dire se  $A \leq^* B$  senza conoscere tutte le altre soluzioni nella popolazione corrente, perché per determinare il rank associato ad  $A$  e  $B$ , il quale è un non-archimedean number nel formato  $F_i = \text{main-rank} + m \cdot \text{second-rank} + \dots$ , abbiamo bisogno di confrontare quelle soluzioni con tutte le altre.

### PL non-dominated sort

- 1) Basato sulle obj. function del lvl 1 divido in frontiere le soluzioni
- 2) Successivamente divido le soluzioni nelle varie frontiere in sub-frontiers basate sulle obj. functions del lvl 2, e così via.



\* Dominate sulla obj. lvl 2

Somma dei ranks con infinitesimi,  
in questo caso rank<sub>1</sub> + m rank<sub>2</sub>

## PL crowding distance

Anche la crowding distance deve essere generalizzata per scegliere nelle subfrontiers.

$$cd = cd_I + \eta \cdot cd_{II} + \eta^2 \cdot cd_{III} \dots$$

↑ at lvl 1      ↑ at lvl 2      ↑ at lvl 3

## PL - NSGA-II

Sulla base di quanto definito, il PL - NSGA-II algorithm è il seguente:

### Algorithm 3 PL-NSGA-II algorithm.

```

1: /*  $P_0$  is the starting population,  $T$  is the total number of iterations */
2: /* The function returns the approximated Pareto front  $P_T$  */
3: procedure PL_NSGA-II ( $P_0, T$ )
4:   for  $t = 0 \dots T - 1$  do
5:      $Q_t = \text{make\_new\_pop}(P_t)$ 
6:      $R_t = P_t \cup Q_t$ 
7:     /*  $F$  is the set of all the non-dominated subfronts */
8:      $F = PL\_fast\_nondom\_sort(R_t, 0)$ 
9:      $P_{t+1} = \emptyset$ 
10:    /*  $i$  is a gross-index, a.k.a. gross-scalar */
11:     $\underline{i} = 1 \oplus^0$ 
12:    while  $|P_{t+1}| + |F_{\underline{i}}| \leq N$  do
13:      /* Crowding distance is computed within subfront  $F_{\underline{i}}$  */
14:      PL_crowding_dist_assignment( $F_{\underline{i}}$ )
15:       $P_{t+1} = P_{t+1} \cup F_{\underline{i}}$ 
16:      /* Move to the next subfront */
17:       $\underline{i} = \text{next\_index}(F, \underline{i})$ 
18:      Sort( $F_{\underline{i}}, \prec_n^*$ )
19:       $P_{t+1} = P_{t+1} \cup F_{\underline{i}}[1 : (N - |P_{t+1}|)]$ 
return  $P_T$ 

```

### Algorithm 2 Priority Levels crowding distance assignment.

```

1: /*  $F$  is a leaf-front in the hierarchy of population fronts partitioning */
2: procedure PL_CROWDING_DIST_ASSIGNMENT( $F$ )
3:    $n = |F|$ 
4:   for all  $i \in F$  do
5:      $F[i].dist = 0$ 
6:   /* For each level of priority  $q$ ;  $p$  is the index of the last level */
7:   for  $q = 1 \dots p$  do
8:     for  $j = 1 \dots m_q$  do
9:        $F = \text{sort}(F, f_j^{(q)})$ 
10:      /* +Inf means "full-scale" (IEEE 754 standard) */
11:       $F[1].dist += +\text{Inf}$ 
12:       $F[n].dist += +\text{Inf}$ 
13:      for  $i = 2 \dots (n - 1)$  do
14:        /* PL_crowd_dist_ass. has infinitesimal parts */
15:         $F[i].dist += \frac{\mathbb{1}^{1-q} f_j^{(q)}(F[i+1]) - f_j^{(q)}(F[i-1])}{f_j^{(q)max} - f_j^{(q)min}}$ 

```

### Algorithm 1 Priority Levels fast non-dominated sort.

```

1: /* This function is recursive */
2: /*  $P$  is the population,  $lvl$  is the current level to consider */
3: procedure PL_FAST_NONDominated_SORT( $P, lvl$ )
4:   /* Base case of recursion */
5:   if  $lvl < \text{min\_lvl}$  then return
6:   /* The first iteration also initializes all ranks to 0 */
7:   if  $lvl == 0$  then
8:     for all  $p \in P$  do
9:        $p.rank = 0$ 
10:      /* Ranking within the priority level to determine subfronts */
11:       $F^{(lvl)} = fast\_nondom\_sort\_in\_level(P, lvl)$ 
12:      /* Repeat for every subfront found */
13:      for all  $F_{\underline{i}} \in F^{(lvl)}$  do
14:        /* In the next priority level */
15:         $F_{\underline{i}} = PL\_fast\_nondominated\_sort(F_{\underline{i}}, lvl-1)$ 

```

First call  
with  
 $lvl = 1$

```

1: procedure FAST_NONDOM_SORT_IN_LEVEL( $P, lvl$ )
2:   for all  $p \in P$  do
3:      $S_p = \emptyset$ 
4:      $n_p = 0$ 
5:     for all  $q \in P$  do
6:       /* non-dominance at specified priority level */
7:       if  $p \prec_{\text{lvl}}^l q$  then
8:          $S_p = S_p \cup \{q\}$ 
9:       else if  $q \prec_{\text{lvl}}^l p$  then
10:         $n_p = n_p + 1$ 
11:      if  $n_p == 0$  then
12:        /* First subfront within  $P$  */
13:         $p.rank = p.rank + \mathbb{1}^{lvl}$ 
14:         $F_1 = F_1 \cup \{p\}$ 
15:      /* Start from the first subfront */
16:       $\underline{i} = \mathbb{1}^{lvl}$ 
17:      while  $F_{\underline{i}} \neq \emptyset$  do
18:         $Q = \emptyset$ 
19:        for all  $p \in F_{\underline{i}}$  do
20:          for all  $q \in S_p$  do
21:             $n_q = n_q - 1$ 
22:            if  $n_q == 0$  then
23:              /*  $q$  belongs to the  $i$ -th subfront */
24:               $q.rank = q.rank + \underline{i} + \mathbb{1}^{lvl}$ 
25:               $Q = Q \cup \{q\}$ 
26:            /* Index of the next subfront */
27:             $\underline{i} = \underline{i} + \mathbb{1}^{lvl}$ 
28:           $F_{\underline{i}} = Q$ 

```

Recursive

## Parallelisation of MOEA

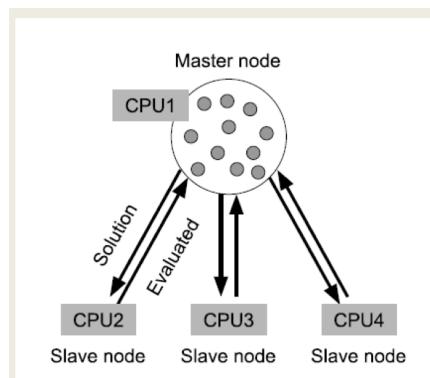
Y MOEA non hanno nei problemi "Embarrassing parallel problems", cioè è molto facile parallelizzarli.

### General Purpose Schemes

Sono schemi generali che funzionano anche nei single-obj problems e non utilizzano l'objective space.

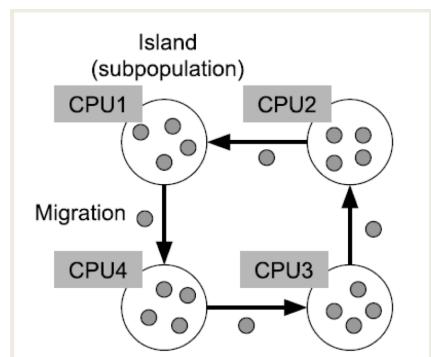
#### 1) Master - Slave

- YL master genera la offspring population e la distribuisce ai vari slave per calcolare le objectives
  - YL master riceve i valori delle objectives e genera la nuova popolazione (crossover + mutation)
- centrizzato



#### 2) Island - Model

- Ogni CPU gestisce una parte della popolazione
- Dopo un tot. di generazioni, esse migrano alcune delle migliori soluzioni da un'isola a un'altra. Questo aiuta a tenere informate le altre isole sui progressi fatti su una specifica isola.



La migrazione può essere random o predefinita.

L'appuccio è completamente decentralizzato (importante per la robustezza, se una CPU fallisce le altre possono continuare).

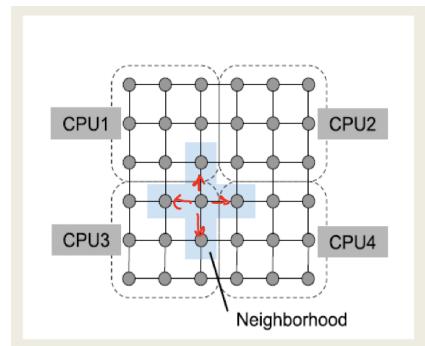
Utilizza alcuni parametri: # gen. per migrare, # individ. to migrate, how to choose indiv. to migrate

### 3) Cellular Model

La popolazione è suddivisa tra le varie CPU e gli individui sono positionati nello spazio in accordo a una topologia predefinita.

Le soluzioni in un nodo migrano solo nei nodi appartenenti nel vicinato, in accordo a un neighborhood scheme predefinito.

Questo modello permette di controllare il numero di comunicazioni tra i nodi e le comunicazioni avvengono solo localmente (nel vicinato, con una forma predefinita)

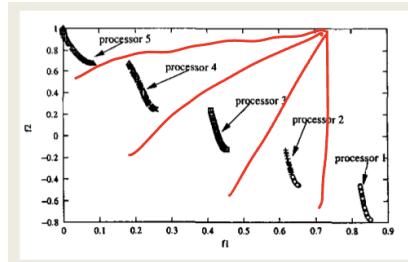
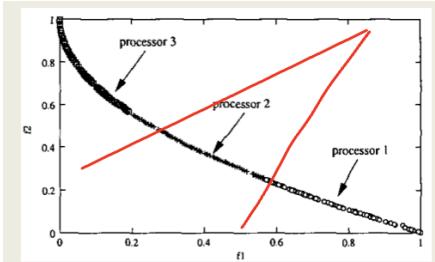


### MOEA specific techniques

Sono schemi specifici per i multi/many obj. evolutionary algorithms e risultano più efficienti perché tengono in considerazione l'obj. space.

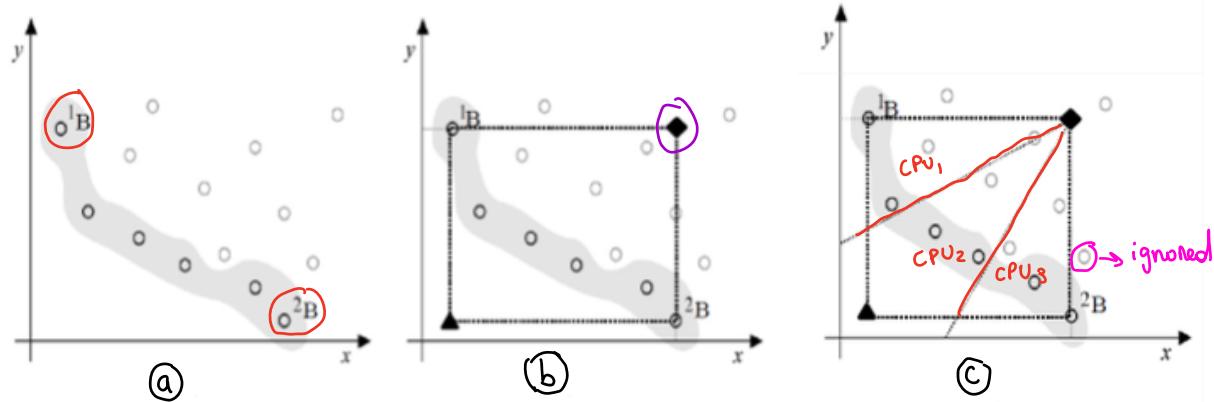
#### 1) Cone separation MEA (cspMEA)

L'idea di cspMEA è di dividere l'obj. space in n coni, dove n corrisponde al numero di cores, ed ogni CPU gestisce una specifica parte dello spazio. (specifica  
cono)



Bi - obj case :

- Identifica le migliori soluzioni per ognuna delle due obj. functions
- Computa il Nadir point
- Divide l'obj. space in equal cones e assegna ognuno ad una CPU.



Oss:

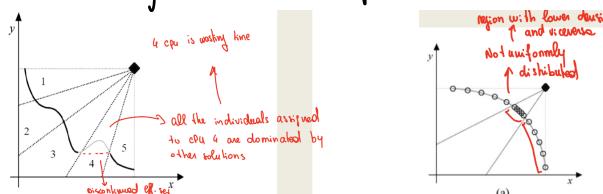
- Può succedere che un offspring cada al di fuori del cono dei parenti, quindi deve essere migrato alla CPU che fa riferimento a quella posizione dell'obj space
- Il Nadir point è dinamico, esso viene calcolato in modo centralizzato e anche i coni
- È facile implementare CSPMEA in una master-slave approach dove il master computa il nadir point e i coni, mentre gli slave svolgono le operazioni di crossover e mutation localmente, valutano obj functions per ogni punto nel loro cono e restituiscono i risultati al master.

PRO: È meglio del random assignment, cioè considera l'obj. space

CONS:

- I pareto front locali potrebbero non appartenere al pareto front globale

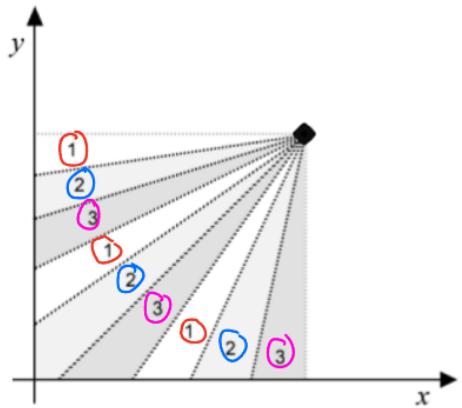
- le soluzioni possono essere distribuite non uniformemente
- Il migration rate potrebbe essere molto alto e unbounded



## 2) Microcones Separation parallel Note

Per risolvere alcuni dei problemi di cspMEA possiamo utilizzare l'idea dei microconi.

L'idea dei microconi è suddividere i coni in microconi ed assegnare ognuno di essi a una differente CPU.



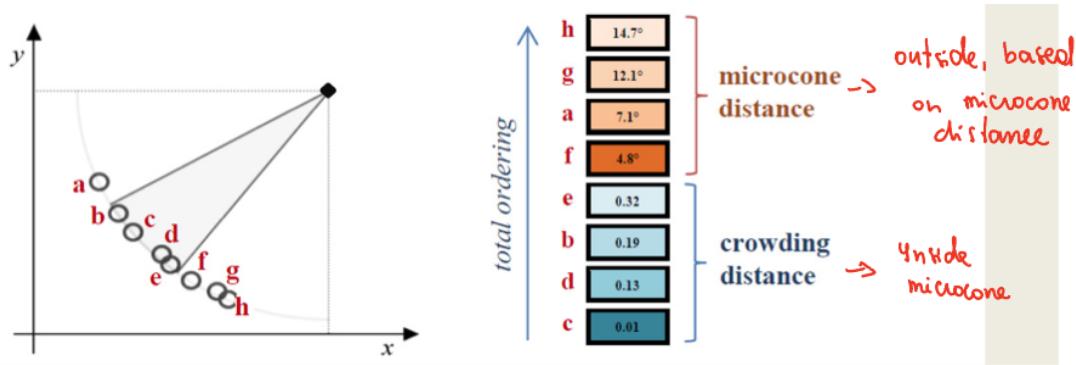
- 1) Dividere ogni cono in microconi
- 2) Assegnare tutti gli "1" alla CPU1, tutti i "2" alla CPU2, etc...

Questo fa sì che tutti i processori sono parzialmente informati sull'intero efficient set e permette di evitare di ottimizzare porzioni dell'efficient set che sono già dominate, e quindi risparmiare risorse.

Come determinare quali soluzioni scegliere per la prossima generazione?

Per ogni CPU dobbiamo scegliere pop-size solutions per la prossima generazione:

- 1) Iterativamente, viene scelta una soluzione da ogni microcone della CPU.
  - Se sono presenti soluzioni non ancora scelte, all'interno del microcone, viene presa quella con crowding distance minore.
  - Se tutte le soluzioni nel microcone sono state scelte, viene presa la soluzione al di fuori del microcone la quale ha la microcone distance minore, cioè la minor distanza angolare tra la soluzione e il microcone in considerazione (la più vicina).



Attenzione: La migrazione deterministica del csPHEA algorithm è stata sostituita da migrazione randomica (n° migrazioni può essere parametrizzato).