Multi-Objective Optimization (MOO) Approaches and Their Applications in Control Systems Design

Outline

Part I

- Optimization problems
- Pareto optimality
- Evolutionary approaches
  - Weighted sum method
  - Multi-objective approach
- Multi-Objective GA (MOGA)
  - NSGA II
  - Non-dominated sorting
  - Crowding distance
- Decision problem
- Performance improvements

Outline (cont.)

Part II

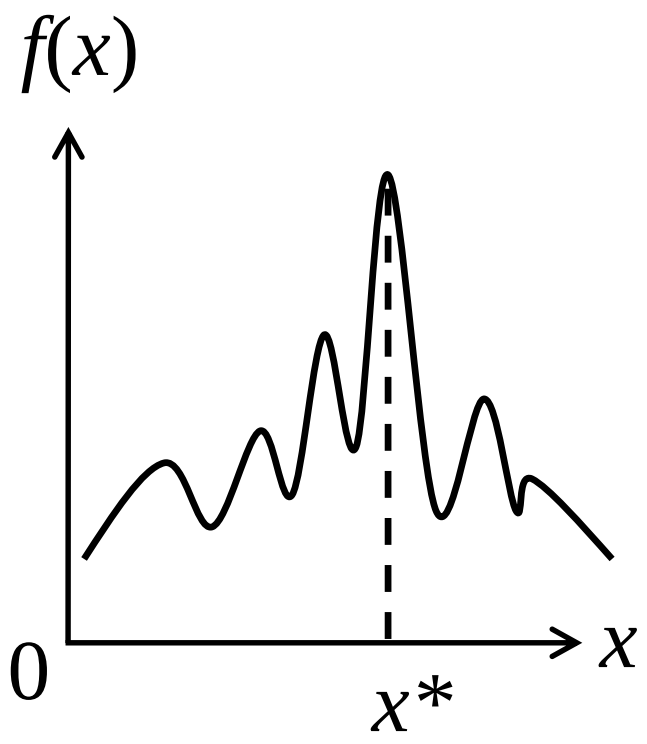
- Interval plants
- Design problem
- Generalized Kharitonov theorem
- Multi-objective evolutionary schemes
- Illustrated examples
- Conclusions

Mono-Objective Optimization

Maximize $f(x)$

subject to $x \in \mathbf{X}$

To find a single optimal solution x^* of a single objective function $f(x)$.



Multi-Objective Optimization Problems (MOOP)

- Involve more than one objective function that are to be minimized or maximized
- Answer is set of solutions that define the best tradeoff between competing objectives

General Form of MOOP

- Mathematically

$$\min/\max \quad f_m(\boldsymbol{x}), \qquad m = 1, 2, \cdots, M$$

$$\text{subject to } g_j(\boldsymbol{x}) \geq 0, \quad j = 1, 2, \cdots, J$$

$$h_k(\boldsymbol{x}) = 0, \quad k = 1, 2, \cdots, K$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \cdots, n$$

($x_i^{(L)}$: lower bound; $x_i^{(U)}$: upper bound)

A solution $\boldsymbol{X}$ is a vector of n decision variables: $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

Dominance

- In the single-objective optimization problem, the superiority of a solution over other solutions is easily determined by comparing their objective function values
- In multi-objective optimization problem, the goodness of a solution is determined by the **dominance**

Definition of Dominance

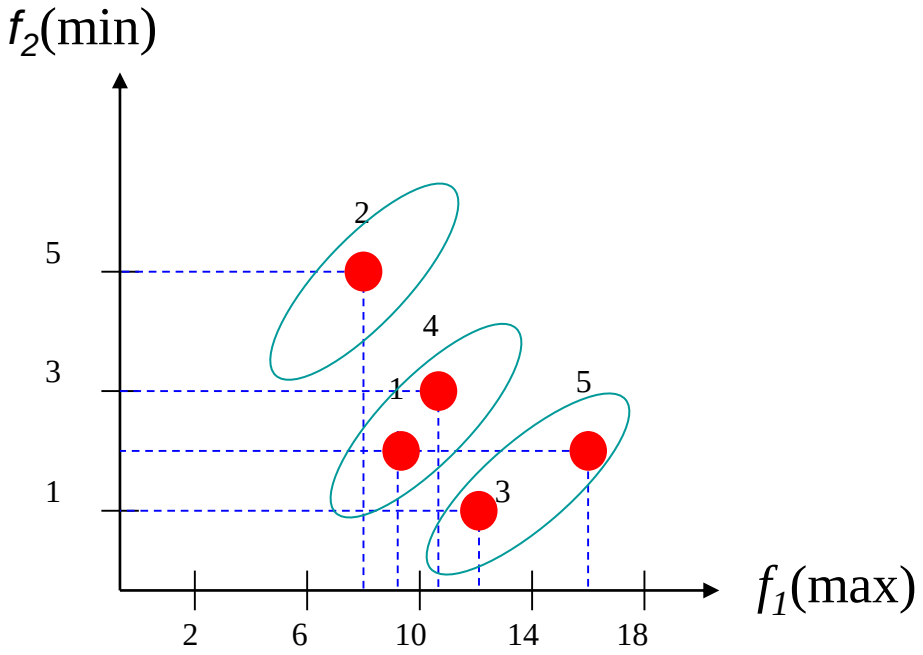
- x_1 dominates x_2, if
  - Solution x_1 is no worse than x_2 in all objectives
  - Solution x_1 is strictly better than x_2 in at least one objective
- x_1 dominates x_2 $\Longleftrightarrow$ x_2 is dominated by x_1

➔ $x_1 \preceq x_2$ OR ➔ $x_1 \succeq x_2$

Dominance

- x_1 dominates x_2
- x_2 is dominated by x_1
- x_1 is non-dominated by x_2
- x_1 is non-inferior to x_2

A Pareto optimal solution is a solution that is not dominated by any other solutions



X_1 dominates X_2

X_5 dominates X_1

X_3 dominates X_4

X_5 non-dominates X_3

Non-domination sets

Level 1: X_3 , X_5

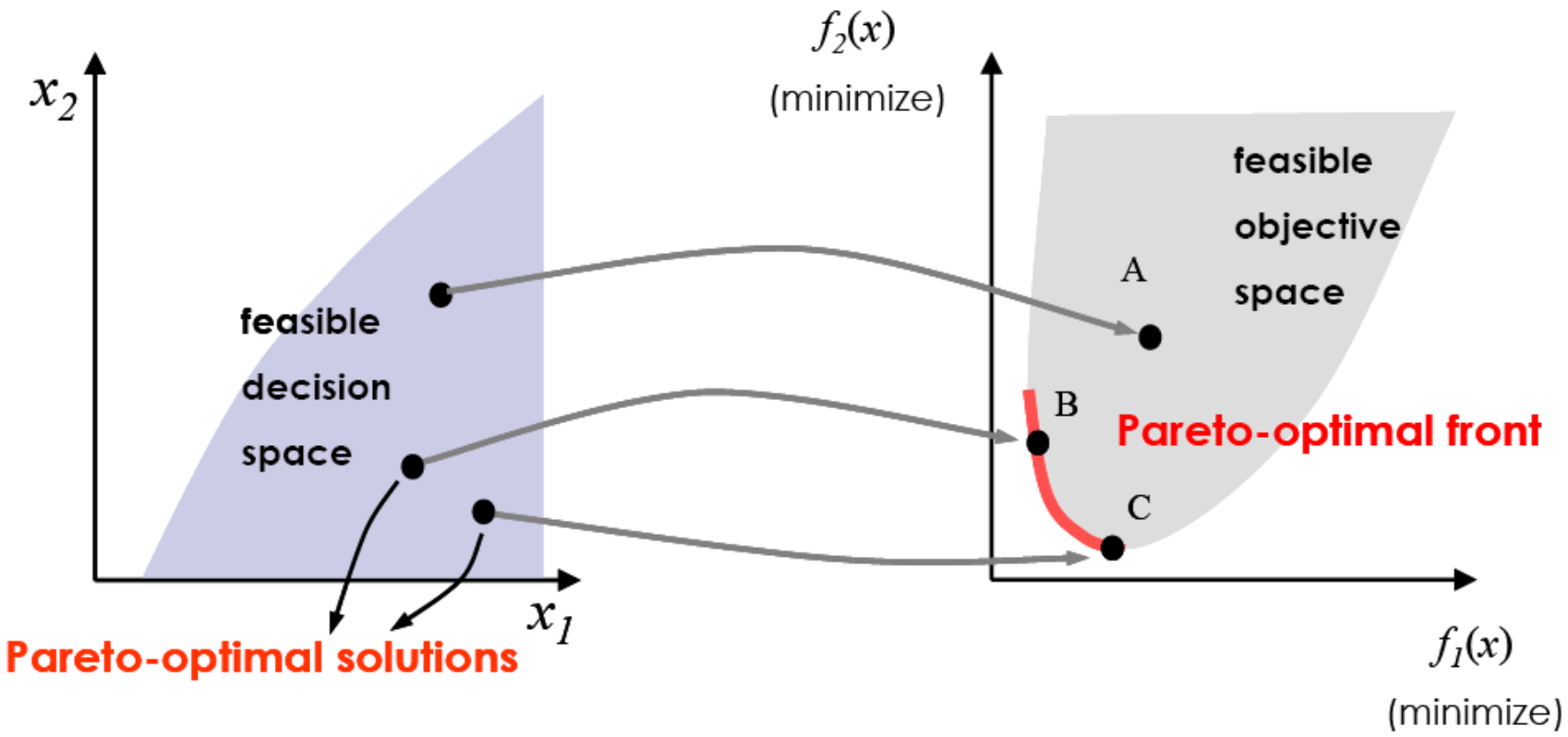
Level 2: X_1, X_4

Level 3: X_2

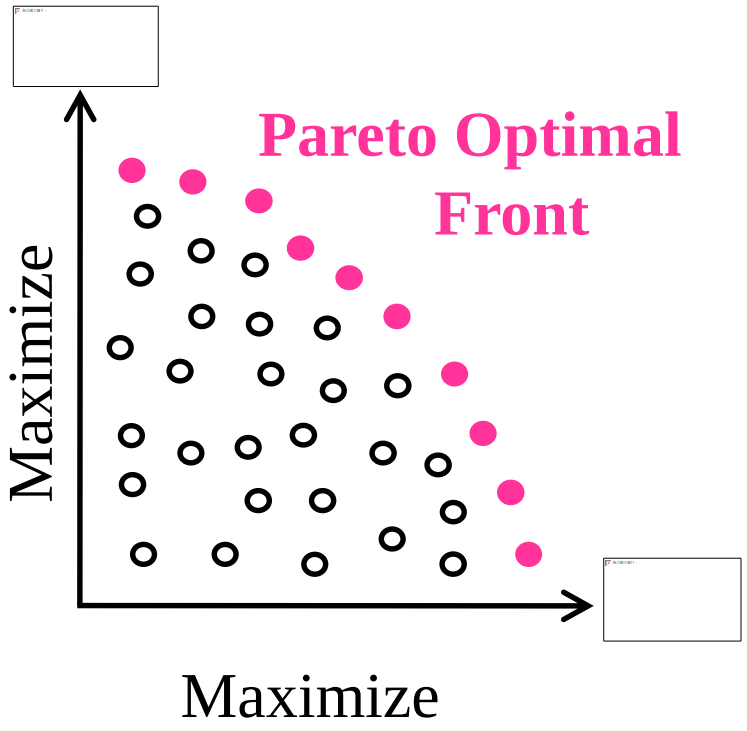
Pareto Optimal Solution

- **Non-dominated solution set** 非主宰解集合
  - Given a set of solutions, the non-dominated solution set is a set of all the solutions that are not dominated by any member of the solution set
- The non-dominated set of the entire feasible decision space is called the **Pareto-optimal set**
- The boundary defined by the set of all point mapped from the Pareto optimal set is called the **Pareto-optimal front**

Graphical Depiction of Pareto Optimal Solution

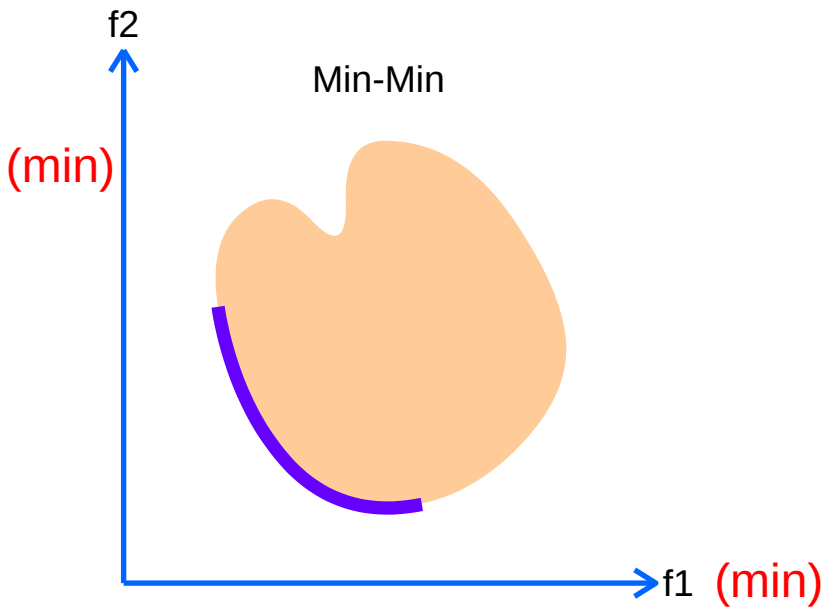


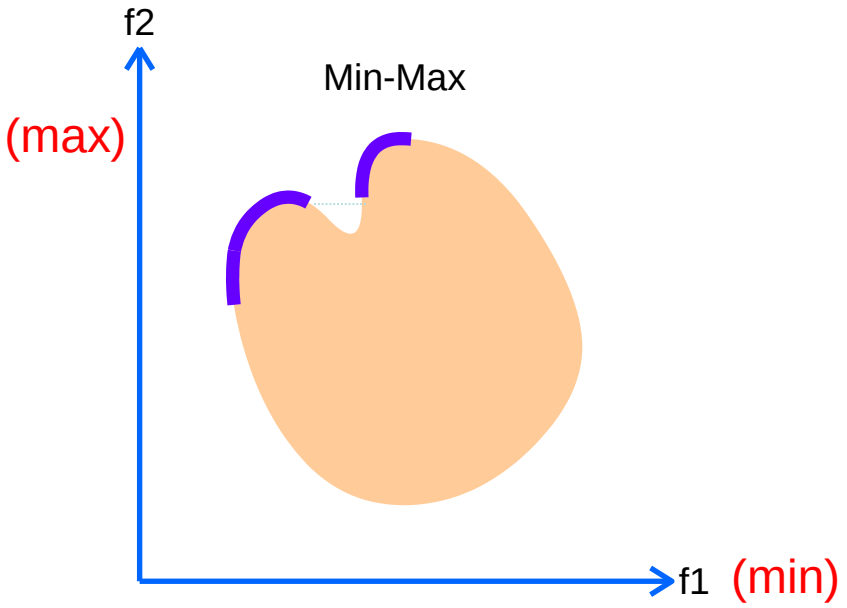
Pareto-optimal solutions

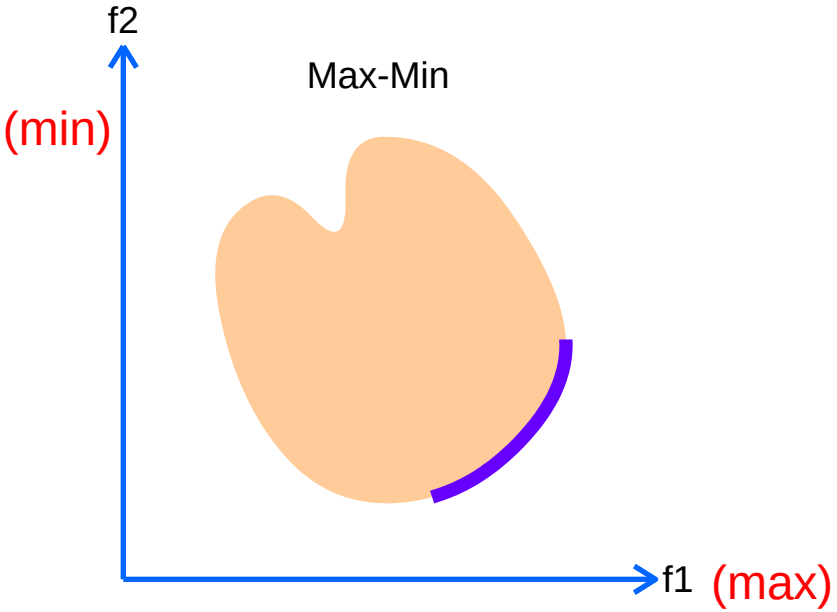


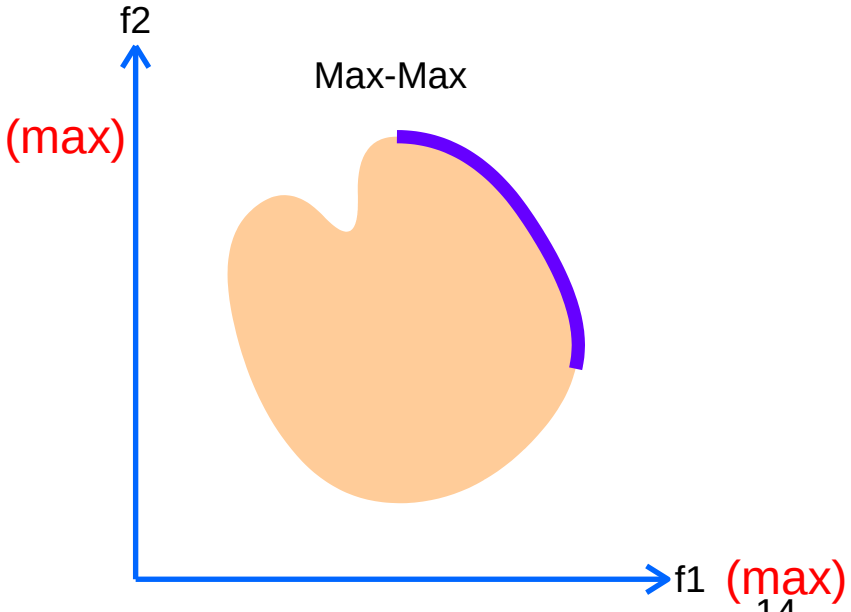
Pareto Optimal Solutions X*

Pareto Optimal Set X*


f2
Min-Min
(min)
f1
(min)


f2
Min-Max
(max)
f1
(min)


f2
Max-Min
(min)
f1
(max)


f2
Max-Max
(max)
f1
(max)

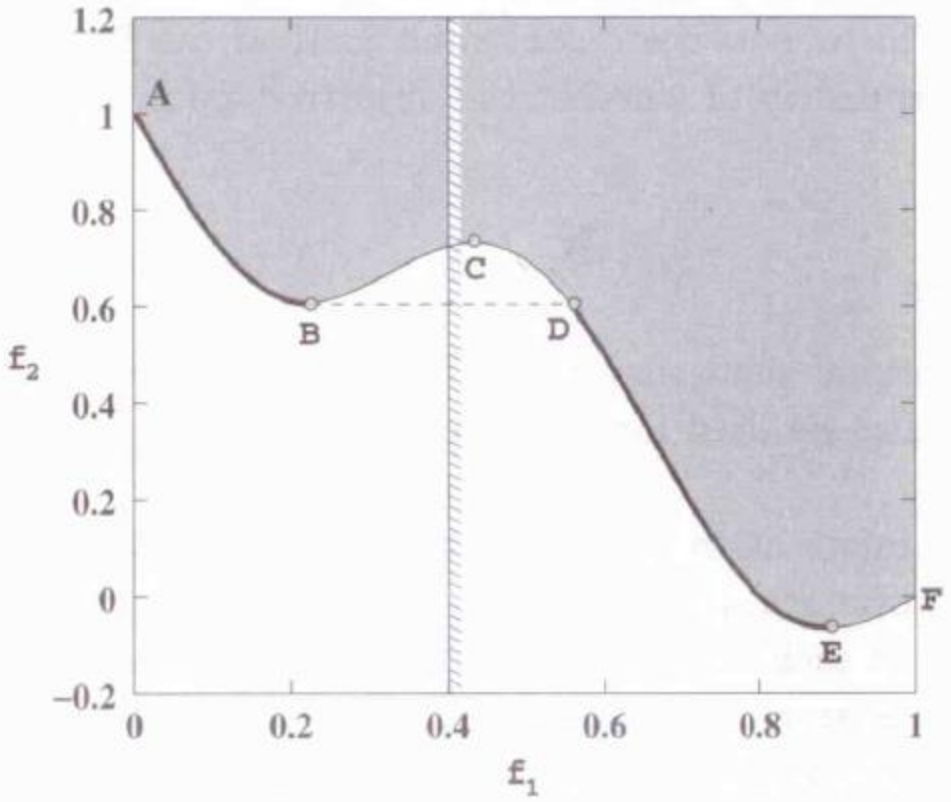


Figure 24 A two-objective problem with disjointed Pareto-optimal sets.

Practical Design Problems

- Examples:
  - Purchase of a car

    Comfort versus cost
  - Optical disk drive:

    $\mathrm{GM} > 6\mathrm{dB}, \mathrm{PM} > 45^{\circ}$
  - Coating of lens:

    Reflection versus no. of layers
  - Controller design:

    Tracking error versus disturbance rejection
- Conflicting objectives
  - Multi-objective optimization problems (MOOP)
- Multi-objective optimization (MOO)

How can we obtain the Pareto-optimal solutions?

- Weighted sum method or

  Scalarisation

  (aggregating objective functions)
- True multi-objective approach

Weighted Sum Method

- Scalarize a set of objectives into a single objective by adding each objective pre-multiplied by a user-supplied weight

$$\textbf{minimize} \quad F(\boldsymbol{x}) = \sum_{m=1}^{M} w_m f_m(\boldsymbol{x}),$$

$$\textbf{subject to} \quad g_j(\boldsymbol{x}) \geq 0, \qquad j = 1, 2, \cdots, J$$

$$h_k(\boldsymbol{x}) = 0, \qquad k = 1, 2, \cdots, K$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \qquad i = 1, 2, \cdots, n$$

- Weight of an objective is chosen in proportion to the relative importance of the objective

- Become a single-objective function!
- Common practice to choose weights such that:

$$\sum_{m=1}^{M} w_m = 1$$

Weighted Sum Method

- Advantage
  - Simple and most widely used
- Disadvantage
  - It is difficult to set the weight vectors to obtain a Pareto-optimal solution in a desired region in the objective space
  - It cannot find certain Pareto-optimal solutions in the case of a nonconvex objective space

What values of the weights must we choose?

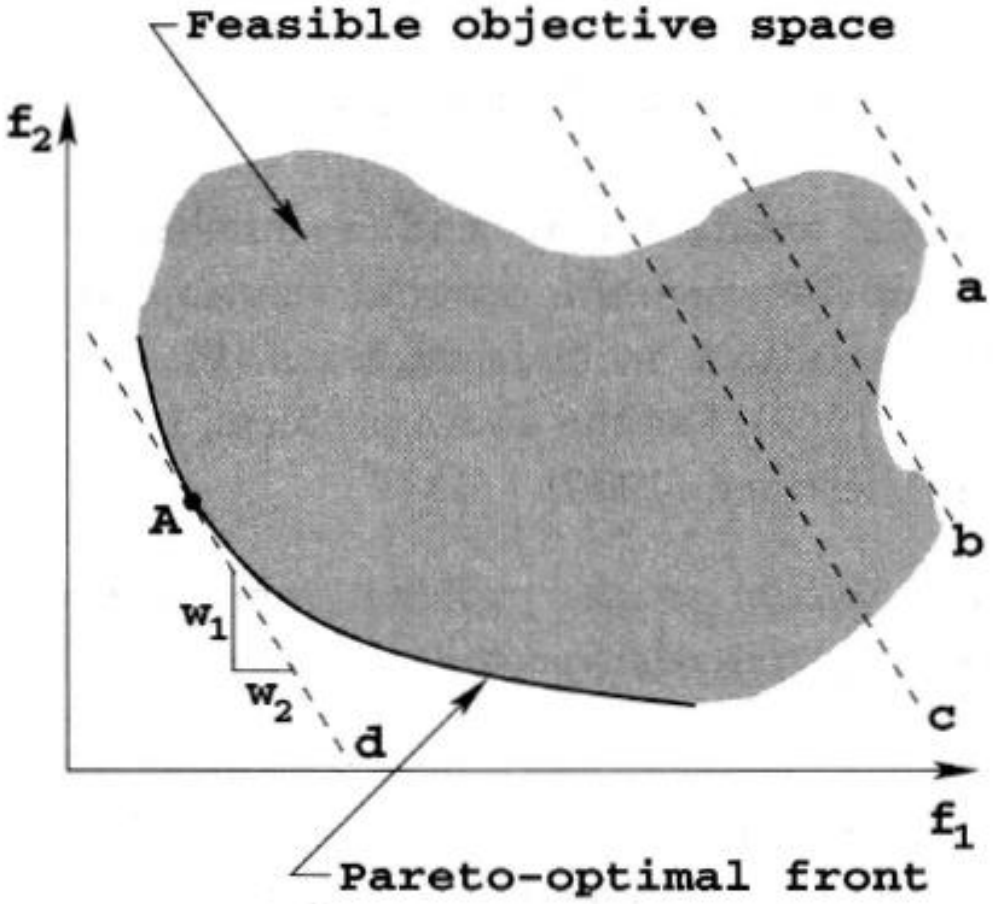
- Not-so-simple question!
- Relative importance of each objective
- Appropriate weight vector depends on the scaling of each objective functions, which might have different orders of magnitude
  - ➢ Dollars vs Comfort
  - ➢ ISE vs Disturbance Rejection
  - ➢ Distance vs Time

➔ Normalization of objectives is required

■ Theorem: the solution to the weighted sum function is Pareto-optimal if the weight is positive for all objectives.

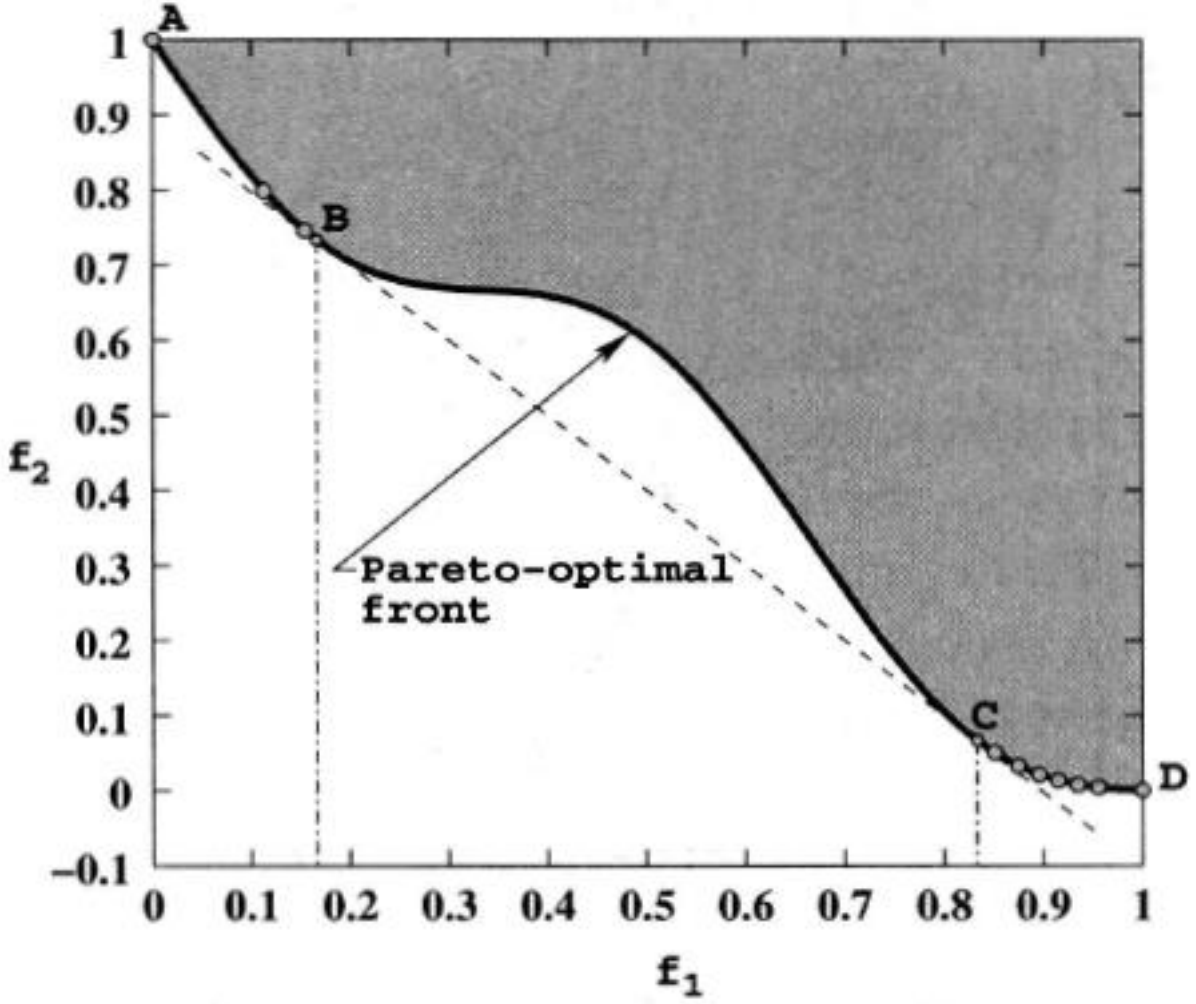
- Any Pareto-optimal solution can be obtained?
  - **No!**
- **Some Pareto-optimal solutions cannot be found.**

■ Theorem: If the solution X^* is a Pareto-optimal solution of a convex multi-objective optimization problem, then there exists a non-zero positive weight vector w such that X^* is a solution to the weighted sum function.



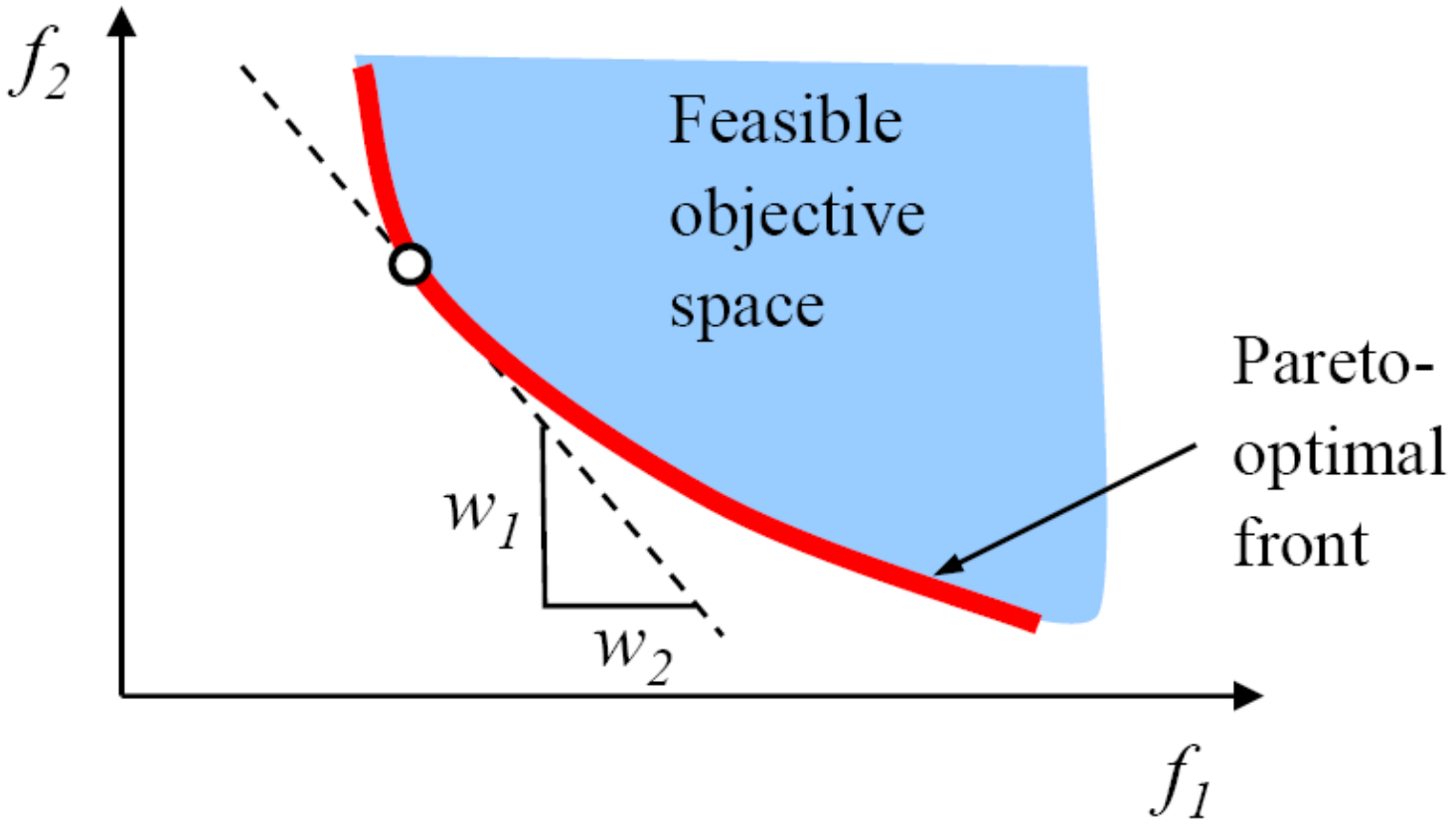
- *F* is a linear combination of both objectives f_1 and f_2
- A straight line as the contour line of *F* on the objective surface.
- The slope of the contour line is related to the choice of the weight vector *w.*
- The effect of lowering the contour line from '*a*' to '*b*' is in effect jumping from solutions of higher *F* values to a lower one, i.e. optimization process.

Weighted sum approach on a convex Pareto-front



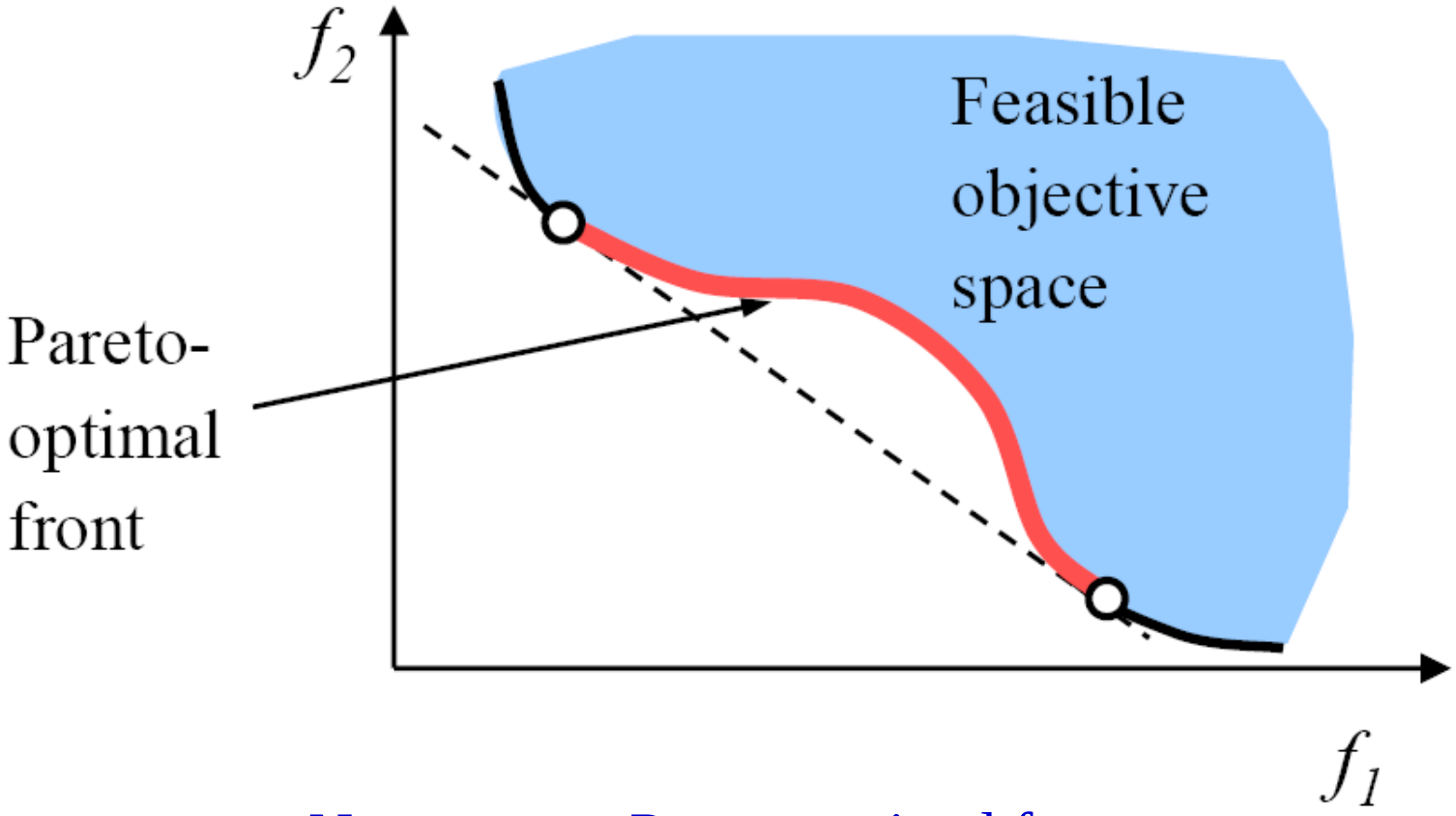
In the case of nonconvex objective space, Pareto optimal solutions in BC cannot be found by using the weighted sum approach, because no contour line will produce a tangent point with the feasible objective space in the region BC.

Weighted Sum Method (Convex Objective space)



Convex Pareto-optimal front

Weighted Sum Method (Non-Convex Objective space)



Non-convex Pareto-optimal front

Why evolutionary approaches?

- Conventional optimization methods
  - Gradient-based methods
  - NM simplex methods
  - Second-order methods

- Advantages:
  - Non-differentiable functions
  - Non-convexity
  - Complicated problems
  - Nature of Parallelism
  - Easy to apply to a wide variety of problems

- Disadvantages:
  - Time-consuming
  - not at all clear what is the best method for some class of practical problems

Evolutionary approaches

- Genetic algorithms (GA)
- Genetic programming (GP)
- Evolutionary strategies (ES)
- Co-evolutionary algorithms (COA)
- Evolutionary programming (EP)
- Particle swarm optimization (PSO)

Multi-Objective EAs (MOEAs)

- There are several different multi-objective evolutionary algorithms
- Depending on the usage of elitism, there are two types of multi-objective EAs

Multi-Objective MOEAs

Non-Elitist MOEAs

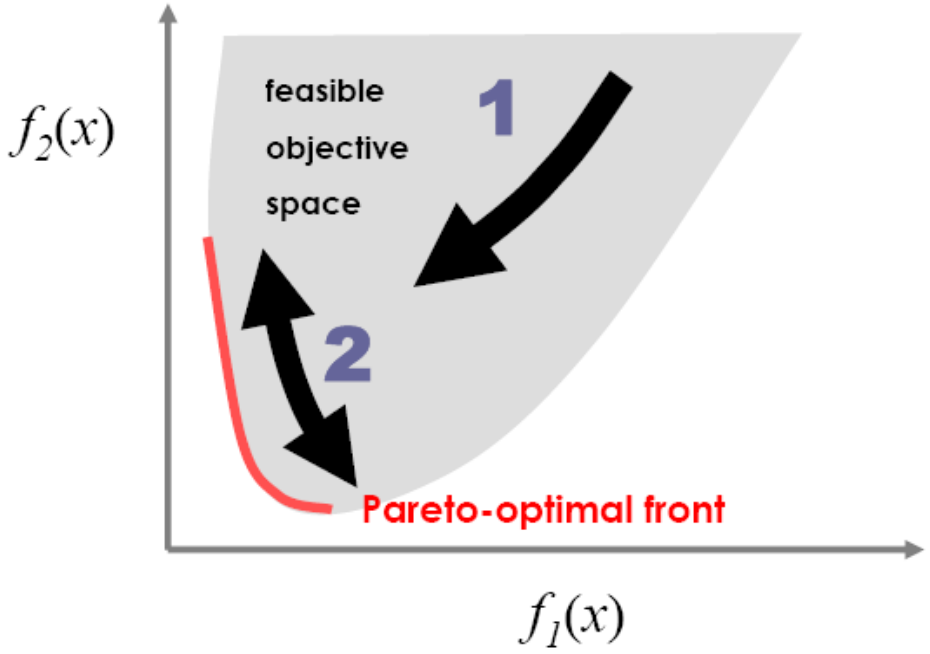
- Vector evaluated GA
- Vector optimized ES
- Weight based GA
- Random weighted GA
- Multiple-objective GA
- Non-dominated Sorting GA
- Niched Pareto GA

Elitist MOEAs

- Elitist Non-dominated Sorting GA
- Distance-based Pareto GA
- Strength Pareto GA
- Pareto-archived ES

Goals in MOO

- Find set of solutions as close as possible to Pareto-optimal front
- To find a set of solutions as diverse as possible



Identifying the Non-Dominated Set

- Naïve and slow approach
- Kung's et. al. Method
  - Computational the most efficient method known
  - Recursive
- Fast non-dominated sorting

Naïve and slow approach

Step 1 Set solution counter $i = 1$ and create an empty non-dominated set P'.

Step 2 For a solution $j \in P$ (but $j \neq i$), check if solution j dominates solution i. If yes, go to Step 4.

Step 3 If more solutions are left in P, increment j by one and go to Step 2; otherwise, set $P' = P' \cup \{i\}$.

Step 4 Increment i by one. If $i \leq N$, go to Step 2; otherwise stop and declare P' as the non-dominated set.

Kung's et. al. Method: Step 1

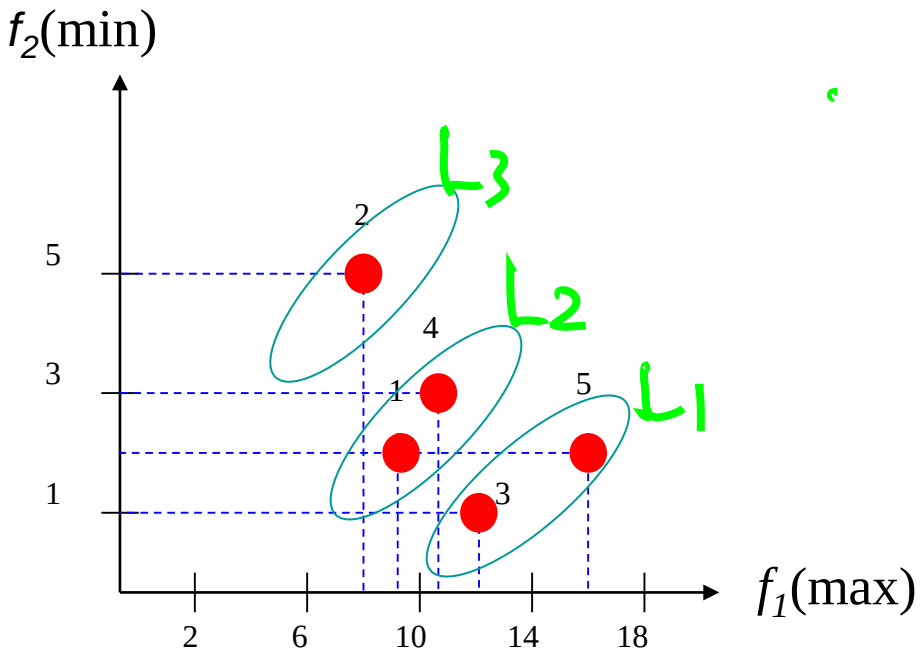
- Step 1: Sort population in descending order of importance of the first objective function and name population as P
- Step 2: Call recursive function Front(P)

Front(P)

```
IF |P| = 1,
  Return P
ELSE
  T = Front ( P(1: [ |P|/2 ]) )
  B = Front ( P( [ |P|/2 + 1 ] : |P|) )
   IF the i-th non-dominated solution of
  B is not dominated by any non-
  dominated solution of T,  M=T ∪{i}
     Return M
  END
END
```

Notes on Front (P)

- |•| is the number of the elements
- P(a : b) means all the elements of P from index a to b,
- [•] is an operator gives the nearest smaller integer value



X_1 dominates X_2

X_5 dominates X_1

X_3 dominates X_4

X_5 non-dominates X_3

Non-domination sets

Level 1: X_3 , X_5

Level 2: X_1, X_4

Level 3: X_2

Step 1 Since the first objective is to be maximized in this problem, we sort the population according to the decreasing value of the first objective function. We obtain the following sequence:

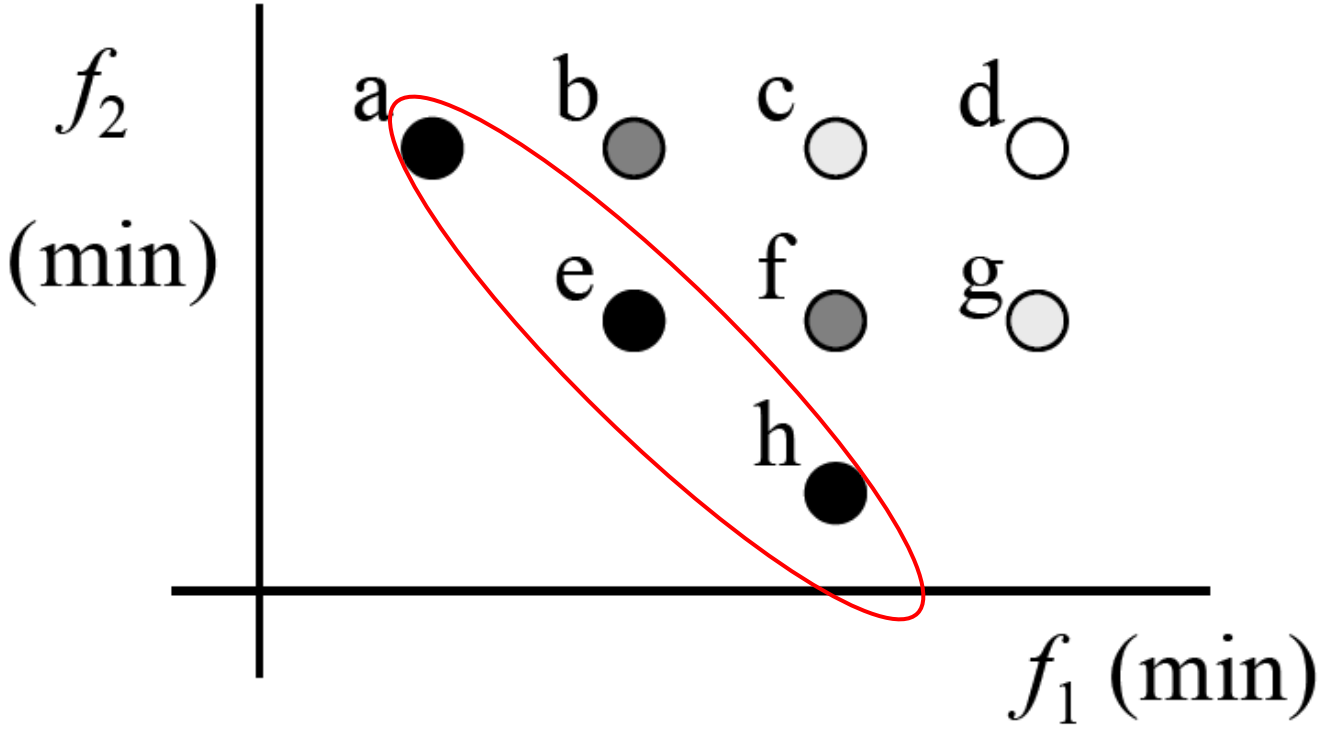
$$P = \{5, 3, 4, 1, 2\}$$

Step 2 When the entire set P enters Front( ), the size of P is not one. Thus, we set $T = \text{Front}(\{5, 3\})$ and $B = \text{Front}(\{4, 1, 2\})$. Recursively, the Front($\{5, 3\}$) would divide the set $\{5, 3\}$ into two halves and make Front($\{5\}$) and Front($\{3\}$) as the next inner T and B sets. Since the sizes of these two inner sets are one each, these two solutions are sent as output to the above Front( ) calls. Moving up a step to Front($\{5, 3\}$) with $T = \{5\}$ and $B = \{3\}$, the domination check and merging operation are then executed. We observe that solution 3 is not dominated by solution 5. Therefore, the merged set contains both solutions, or $M = \{5, 3\}$. Thus, the outcome of the $T = \text{Front}(\{5, 3\})$ is the set $\{5, 3\}$.

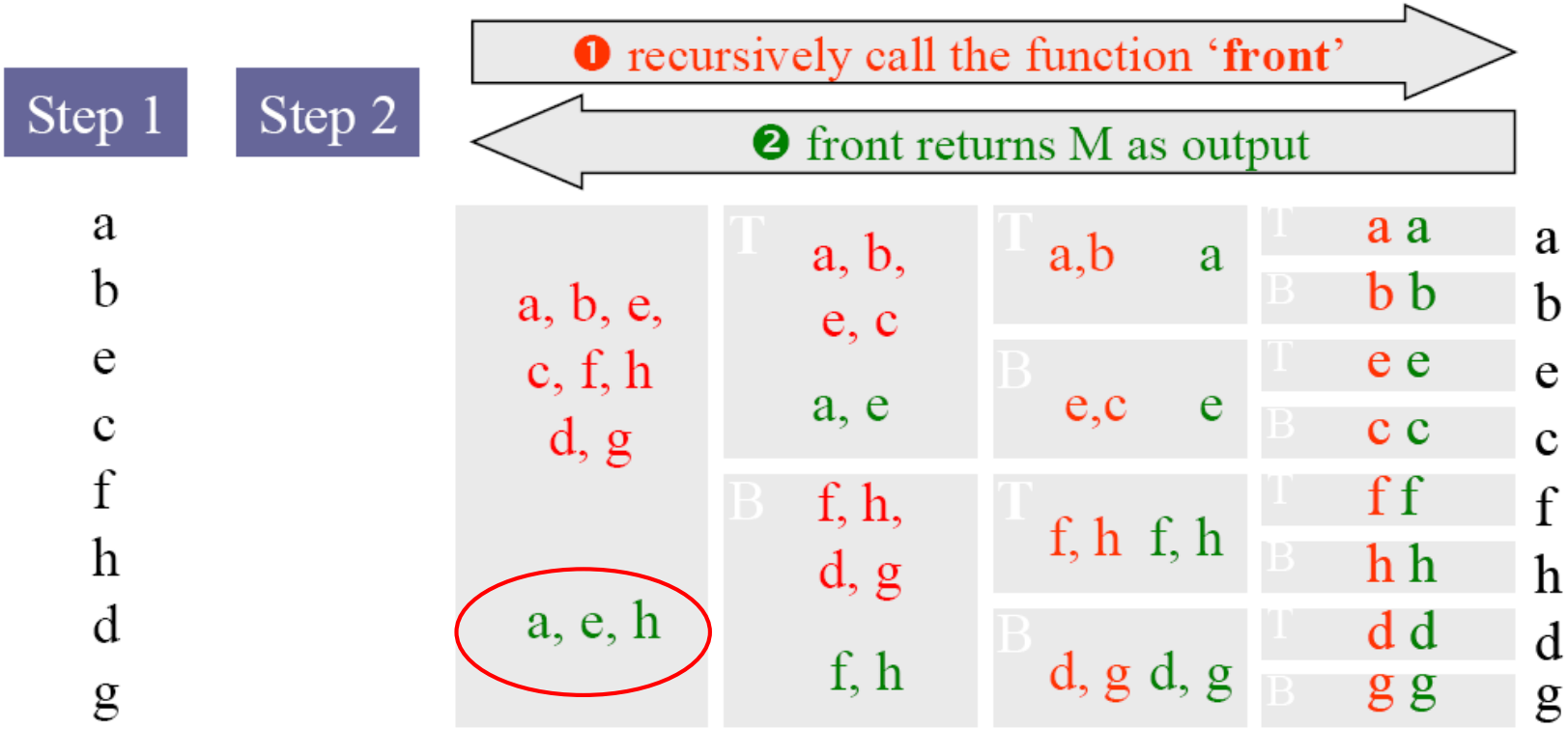
Similarly, the Front($\{4, 1, 2\}$) operation makes two calls, i.e. Front($\{4\}$) and Front($\{1, 2\}$). The output from the former is $\{4\}$. The latter makes two more calls, i.e. Front($\{1\}$) and Front($\{2\}$), with outputs as $\{1\}$ and $\{2\}$, respectively. Now, to complete the Front($\{1, 2\}$) operation, we need to check whether solution 2 is indeed dominated by solution 1. As we already know that solution 2 is dominated by solution 1, we have $M = \{1\}$ as the output of Front($\{1, 2\}$). Now, to complete the Front($\{4, 1, 2\}$) operation, we need to check the domination of solutions 4 (T) and 1 (B). Since solution 1 is not dominated by solution 4, the merged set is $M = \{4, 1\}$, which is also the output of the Front($\{4, 1, 2\}$) call.

Now, for the overall call to Front(P), the sets are $\{5, 3\}$ and $\{4, 1\}$, respectively. Checking the domination of the solutions in B with the solutions of T, we observe that both solutions 4 and 1 are dominated by solution 5. Thus, the output of the Front(P) call is the set $\{5, 3\}$, and this is the non-dominated set.

Example of Kung's Method



Example of Kung's Method



Elitist MOEAs

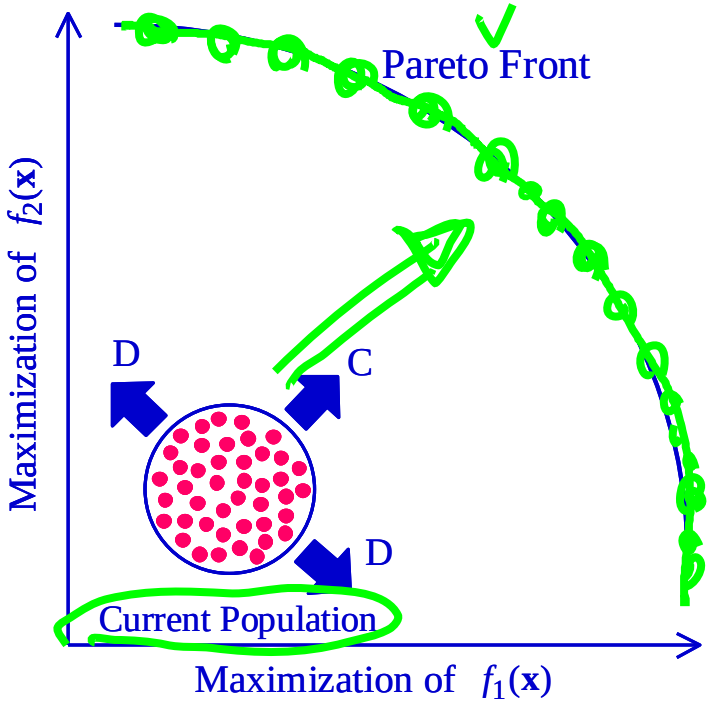
- Elite-preserving operator carries elites of a population to the next generation
- Rudolph(1996) proved GAs converge to the global optimal solution of some functions in the presence of elitism
- Elitist MOEAs two methods are often used
  - Elitist Non-dominated Sorting GA (NSGA II)
  - Strength Pareto EA

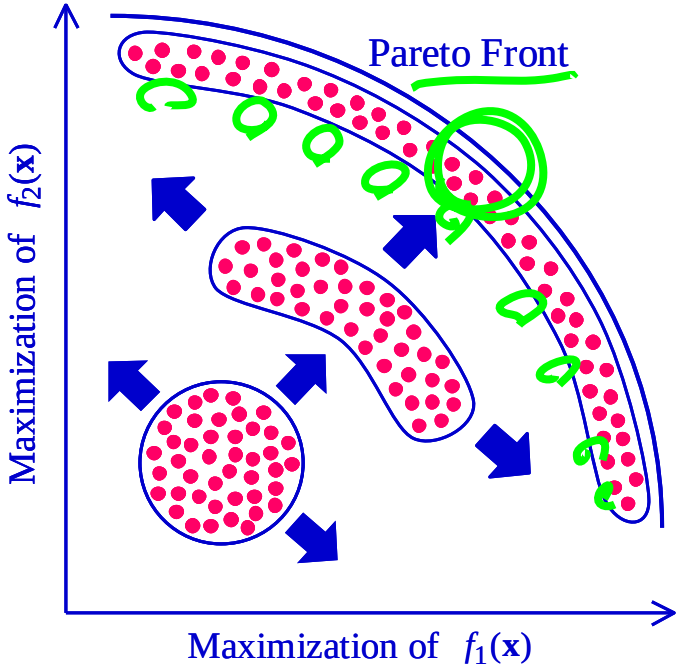
* **Reference**: G. Rudolph, *Convergence of evolutionary algorithms in general search spaces,* In Proceedings of the Third IEEE conference of Evolutionary Computation, 1996, p.50-54

Elitist Non-Dominated Sorting GA
(Deb et al., 2000)

- Use an explicit diversity-preserving strategy together with an elite-preservation strategy

Diversity and Convergence

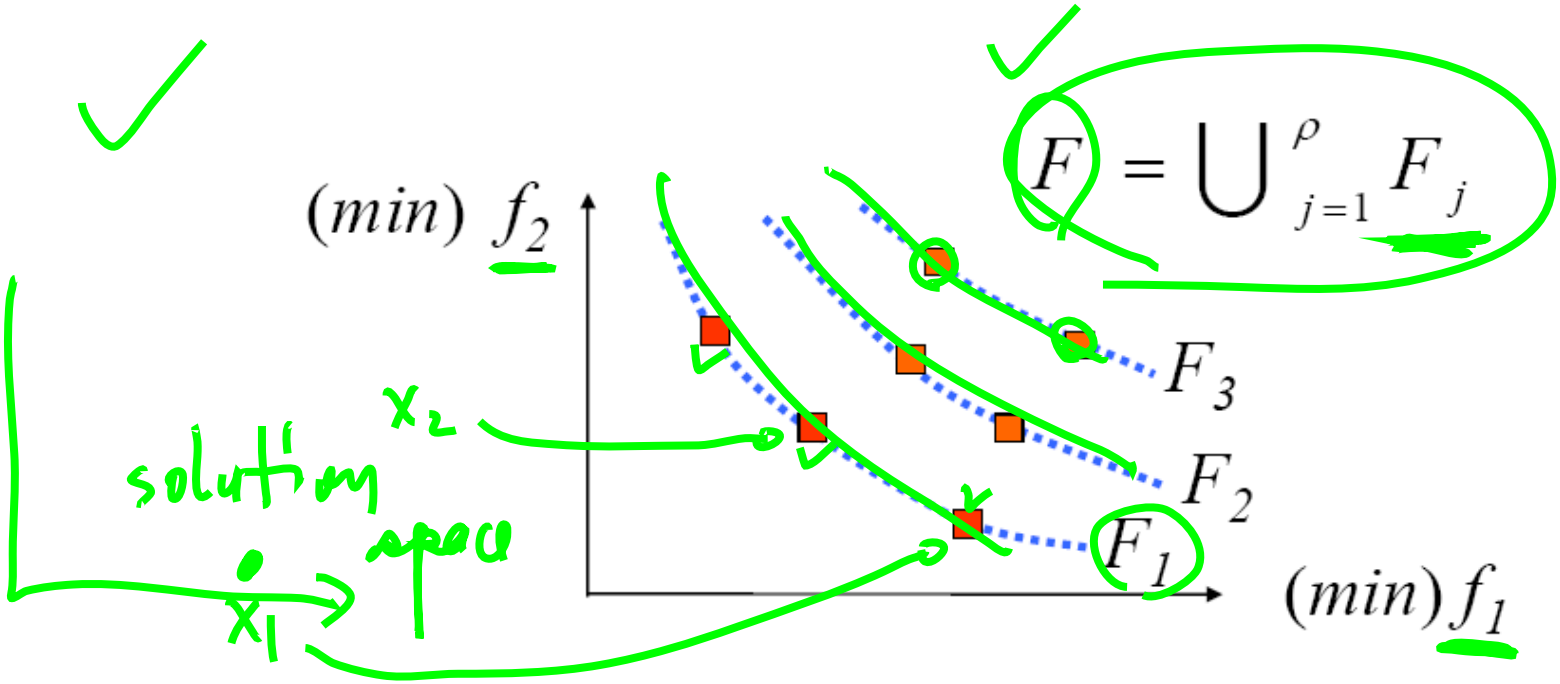




Crowding + Pareto Ranking & Elitist Strategy

Elitist Non-Dominated Sorting GA

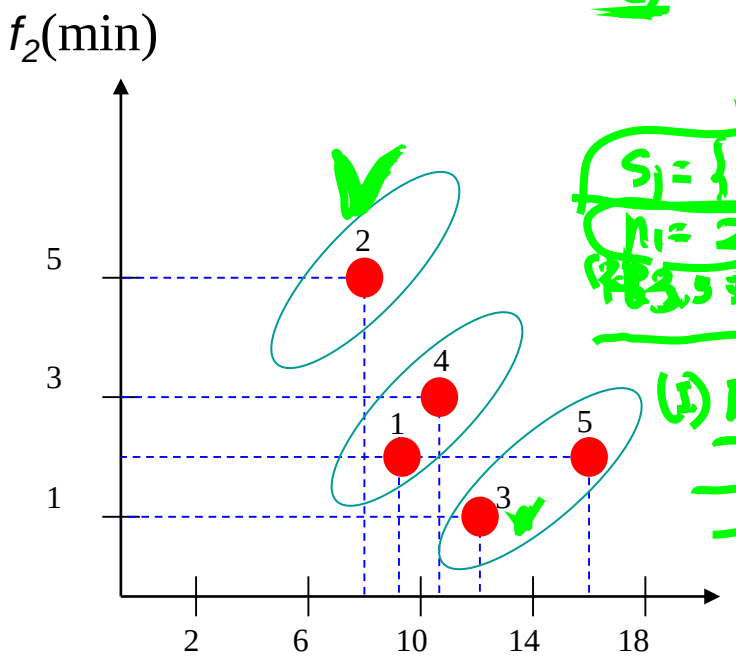
- Non-Dominated Sorting
  - Classify the solutions into a number of mutually exclusive equivalent non-dominated sets



Fast non-dominated sort

```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    If p dominates q
         S_p = S_p ∪ {q}                 Add q to the set of solutions dominated by p
      else if (q ≺ 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_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Initialize the front counter
while F_i ≠ ∅
   Q = ∅                                 Used to store the members of the next front
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q belongs to the next front
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Hand calculation



$F_3 = \emptyset = \{2\}$

algorithm 4.8

Non-Dominated Sorting Algorithm [i, j]

Step 1 For each $i \in P$, $n_i = 0$ and initialize $S_i = \emptyset$. For all $j \neq i$ and $j \in P$, perform Step 2 and then proceed to Step 3.

Step 2 If $i \preceq j$, update $S_i = S_i \cup \{j\}$. Otherwise, if $j \preceq i$, set $n_i = n_i + 1$.

Step 3 If $n_i = 0$, keep i in the first non-dominated front P_1 (we called this set P' in the above paragraph). Set a front counter $k = 1$.

Step 4 While $P_k \neq \emptyset$, perform the following steps.

Step 5 Initialize $Q = \emptyset$ for storing next non-dominated solutions. For each $i \in P_k$ and for each $j \in S_i$,

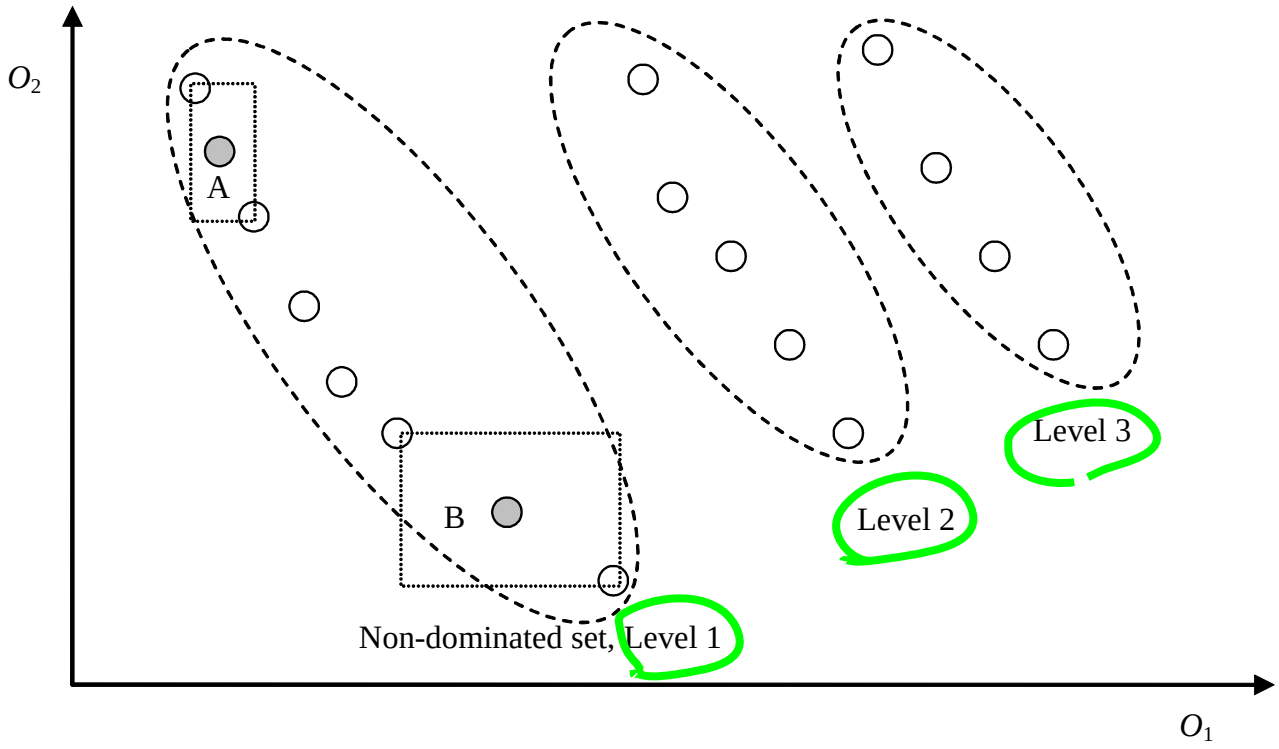
Step 5a Update $n_j = n_j - 1$.

Step 5b If $n_j = 0$, keep j in Q, or perform $Q = Q \cup \{j\}$.

Step 6 Set $k = k + 1$ and $P_k = Q$. Go to Step 4.

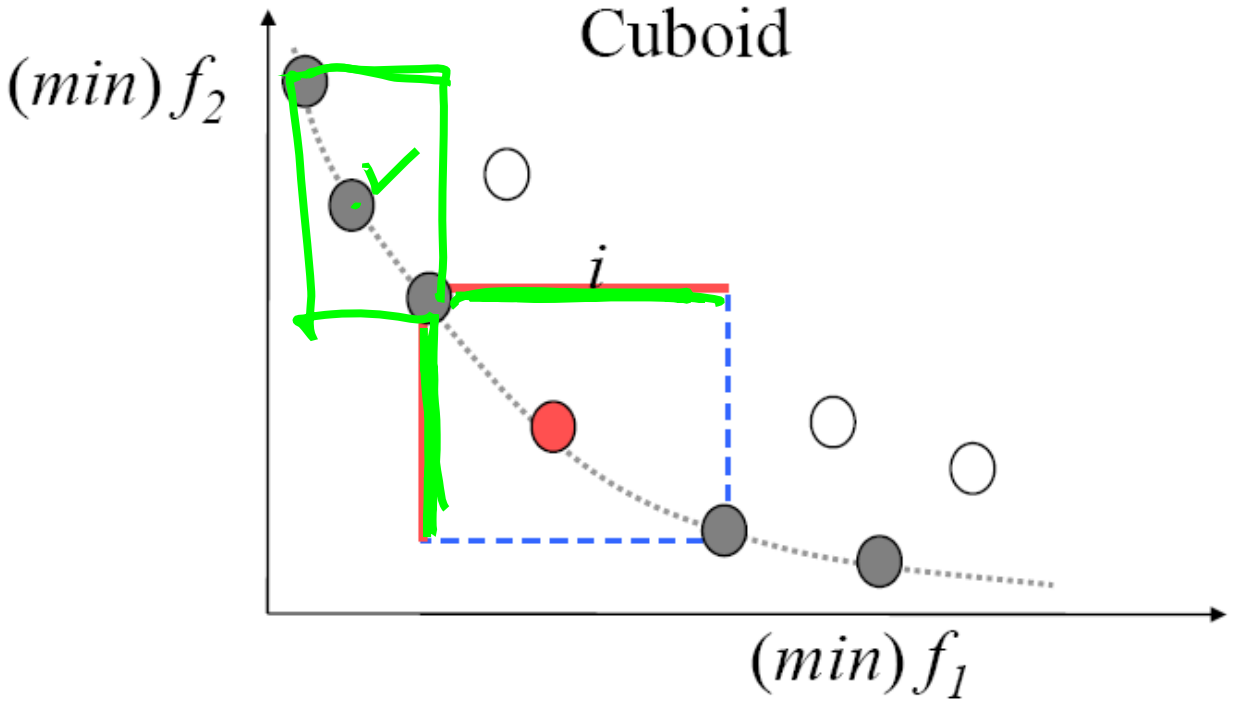
crowding distance

Crowding Techniques

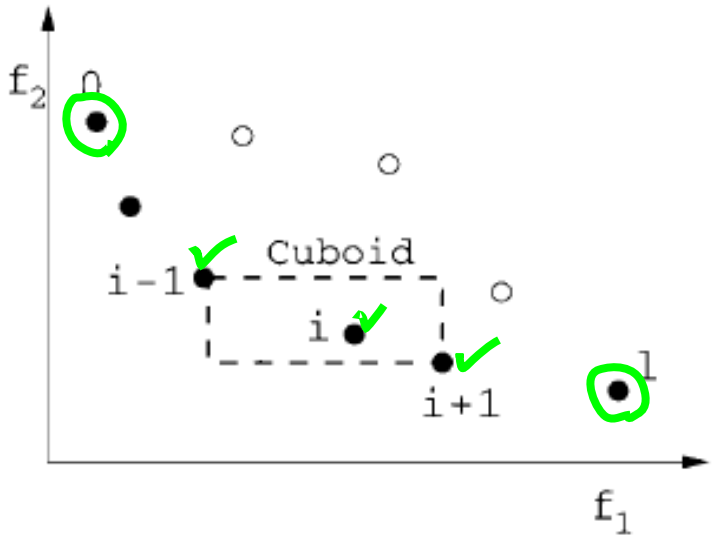


Elitist Non-Dominated Sorting GA

- Determine Crowding Distance for each solution
  - Denotes half of the perimeter of the enclosing cuboid with the nearest neighboring solutions in the same front



Crowding Distance



$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\max} - f_m^{\min}}$$

m=1,2, . . .

Crowding Distance Assignment Procedure: `Crowding-sort`$(\mathcal{F}, <_c)$

Step C1 Call the number of solutions in $\mathcal{F}$ as $l = |\mathcal{F}|$. For each i in the set, first assign $d_i = 0$.

Step C2 For each objective function $m = 1, 2, \ldots, M$, sort the set in worse order of f_m or, find the sorted indices vector: $I^m = \text{sort}(f_m, >)$.

Step C3 For $m = 1, 2, \ldots, M$, assign a large distance to the boundary solutions, or $d_{I_1^m} = d_{I_l^m} = \infty$, and for all other solutions $j = 2$ to $(l-1)$, assign:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\max} - f_m^{\min}}.$$

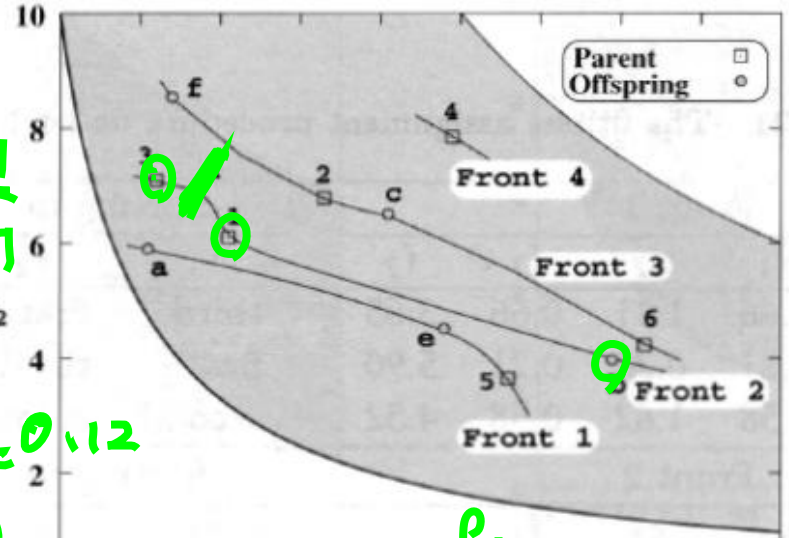
Table 21 The fitness assignment procedure under NSGA-II.

Front 1					Sorting in		Distance
Solution	x_1	x_2	f_1	f_2	f_1	f_2	
5	0.66	1.41	0.66	3.65	third	first	∞
a	0.21	0.24	0.21	5.90	first	third	∞
e	0.58	1.62	0.58	4.52	second	second	0.54

Front 2					Sorting in		Distance
Solution	x_1	x_2	f_1	f_2	f_1	f_2	
1	0.31	0.89	0.31	6.10	third	second	0.63
3	0.22	0.56	0.22	7.09	first	fourth	∞
b	0.79	2.14	0.79	3.97	fourth	first	∞
d	0.27	0.87	0.27	6.93	second	third	0.12

$$d = \phi + \frac{f_1^{(i+1)} - f_1^{(i-1)}}{f_1^{max} - f_1^{min}} = \frac{0.31 - 0.22}{1 - 0.1} = 0.1$$

$$d = 0.1 + \frac{f_2^{(i+1)} - f_2^{(i-1)}}{f_2^{max} - f_2^{min}} = 0.1 + \frac{4.09 - 6.10}{50 - 0} = 0.12$$



Step C1 We notice that $l = 4$ and set $d_1 = d_3 = d_b = d_d = 0$. We also set $f_1^{max} = 1$, $f_1^{min} = 0.1$ $f_2^{max} = 60$ and $f_2^{min} = 0$.

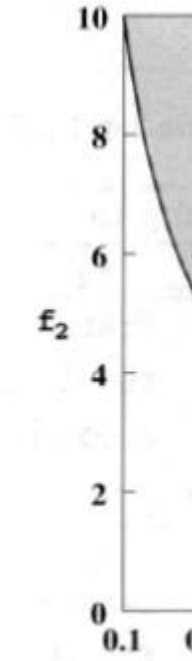
Step C2 For the first objective function, the sorting of these solutions is shown in Table 21 and is as follows: $I^1 = \{3, d, 1, b\}$.

Step C3 Since solutions 3 and b are boundary solutions for f_1, we set $d_3 = d_b = \infty$. For the other two solutions, we obtain:

f_1	f_2
0.31	6.10
0.22	7.09
0.79	3.97
0.27	6.93

$$d_d = 0 + \frac{f_1^{(1)} - f_1^{(3)}}{f_1^{max} - f_1^{min}} = 0 + \frac{0.31 - 0.22}{1 - 0.1} = 0.10.$$

$$d_1 = 0 + \frac{f_1^{(b)} - f_1^{(d)}}{f_1^{max} - f_1^{min}} = 0 + \frac{0.79 - 0.27}{1 - 0.1} = 0.58.$$



Now, we turn to the second objective function and update the above distances. First, the sorting on this objective yields $I^2 = \{b, 1, d, 3\}$. Thus, $d_b = d_3 = \infty$ and the other two distances are as follows:

$$d_1 = d_1 + \frac{f_2^{(d)} - f_2^{(b)}}{f_2^{max} - f_2^{min}} = 0.58 + \frac{6.93 - 3.97}{60 - 0} = 0.63.$$

$$d_d = d_d + \frac{f_1^{(3)} - f_2^{(1)}}{f_2^{max} - f_2^{min}} = 0.10 + \frac{7.09 - 6.10}{60 - 0} = 0.12.$$

The overall crowded distances of the four solutions are:

$$d_1 = 0.63, \quad d_3 = \infty, \quad d_b = \infty, \quad d_d = 0.12.$$

Elitist Non-Dominated Sorting GA

- Crowding tournament selection
  - Assume that every solution has a non-domination rank and a local crowding distance
  - A solution i wins a tournament with another solution j
    1. if the solution i has a better rank
    2. They have the same rank but solution i has a better crowing distance than solution j

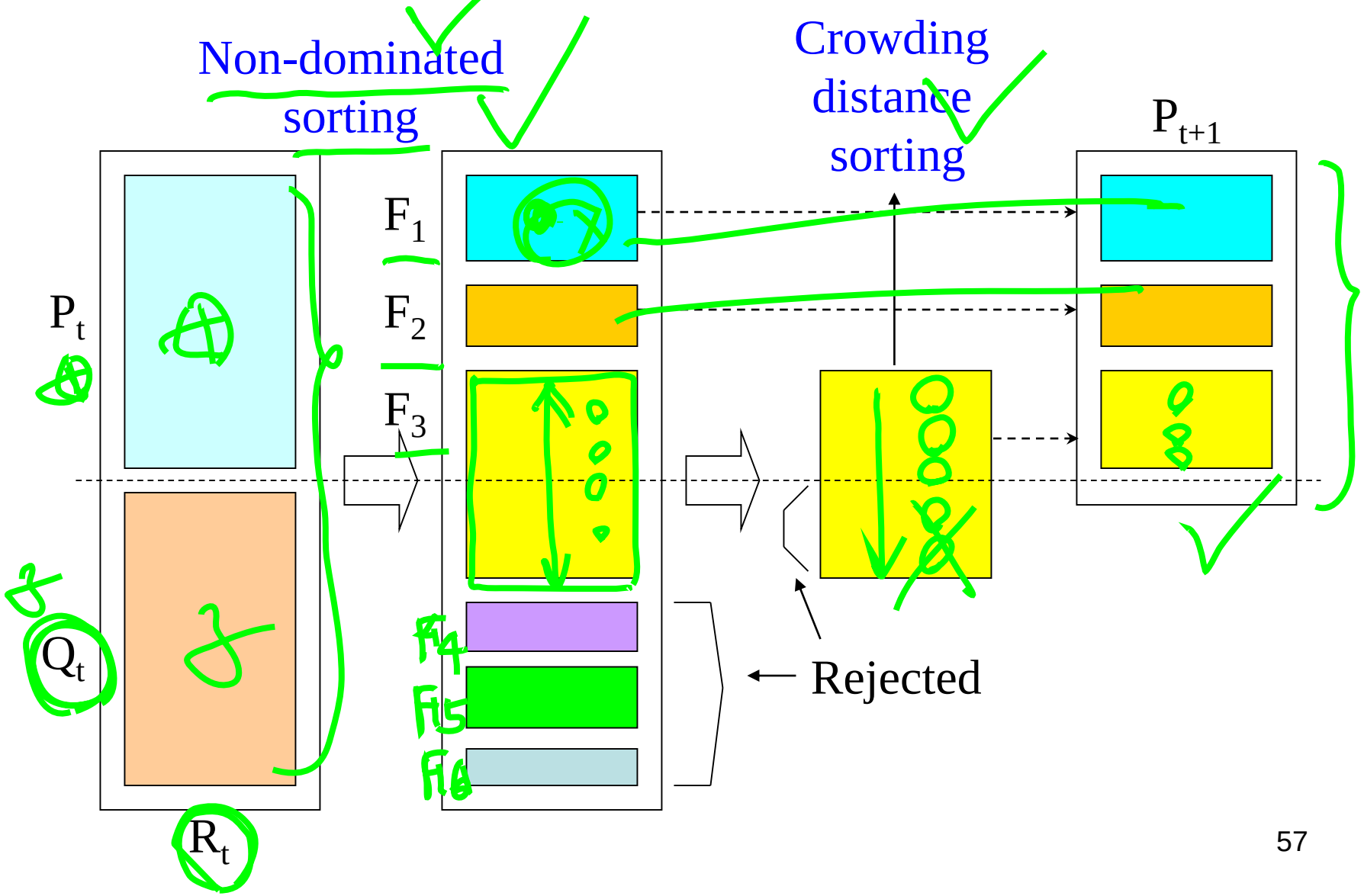
Elitist Non-Dominated Sorting GA

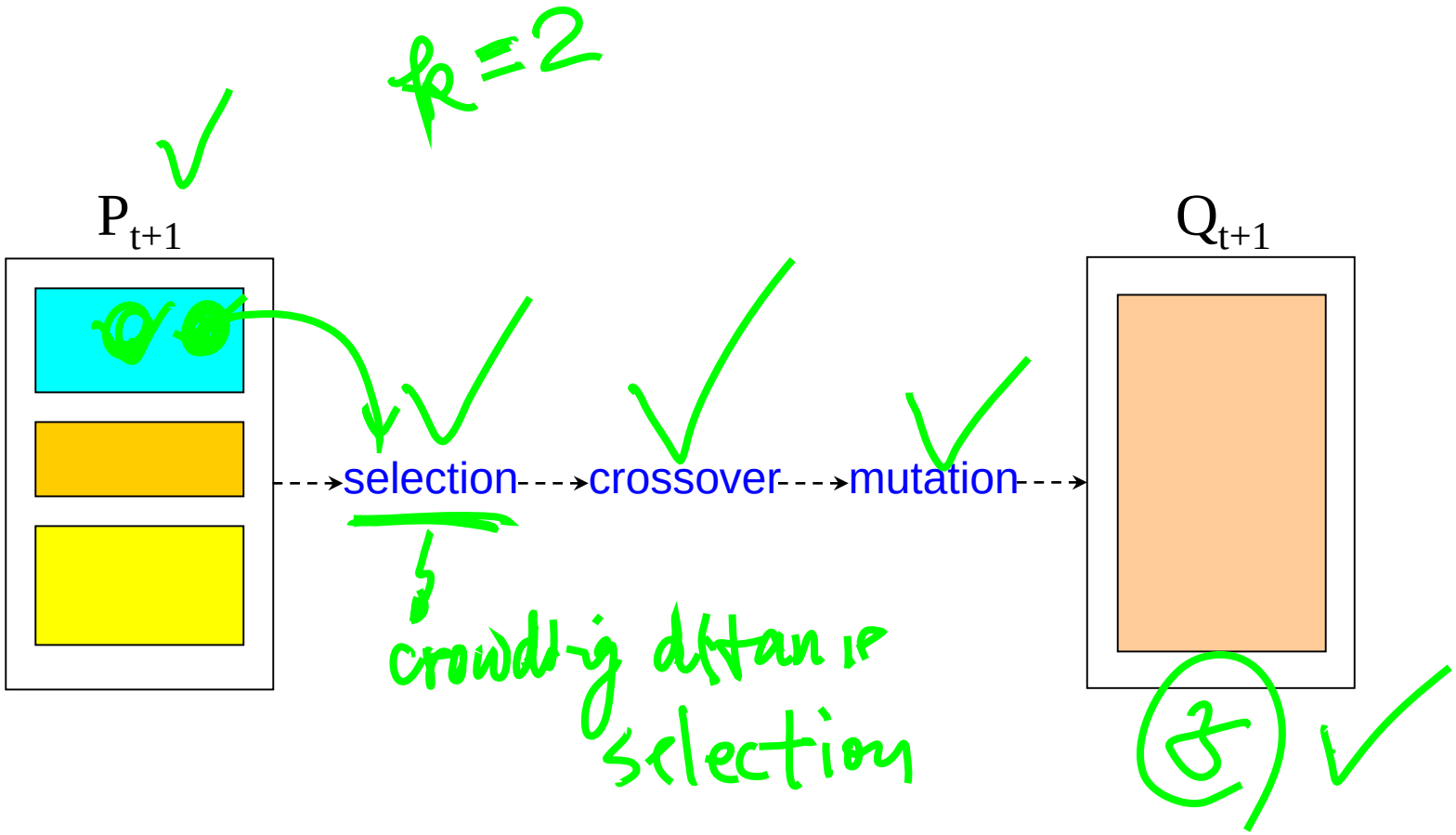
- **Step 1**
  - Combine parent P_t and offspring Q_t populations $R_t = P_t \cup Q_t$
  - Perform a **non-dominated sorting** to R_t and find different fronts F_i
- **Step 2**
  - Set new population $P_{t+1} = \varnothing$ and set $i = 1$
  - Until $|P_{t+1}| + |F_i| < N$, perform $P_{t+1} = P_{t+1} \cup F_i$ and increase i

Elitist Non-dominated Sorting GA

- **Step 3**
  - Include the most widely spread solutions (N-$|P_{t+1}|$) of F_i in P_{t+1} using the **crowding distance** values
- **Step 4**
  - Create offspring population Q_{t+1} from P_{t+1} by using the **crowded tournament selection**, crossover and mutation operators

How MOGA works?




P_{t+1}
selection
crossover
mutation
Q_{t+1}
crowding distance selection

$R_t = P_t \cup Q_t$ — combine parent and offspring population

$\mathcal{F} =$ `fast-non-dominated-sort`(R_t) — $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all nondominated fronts of R_t

$P_{t+1} = \emptyset$ and $i = 1$

until $|P_{t+1}| + |\mathcal{F}_i| \leq N$ — until the parent population is filled

  `crowding-distance-assignment`$(\mathcal{F}_i)$ — calculate crowding-distance in $\mathcal{F}_i$

  $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$ — include ith nondominated front in the parent pop

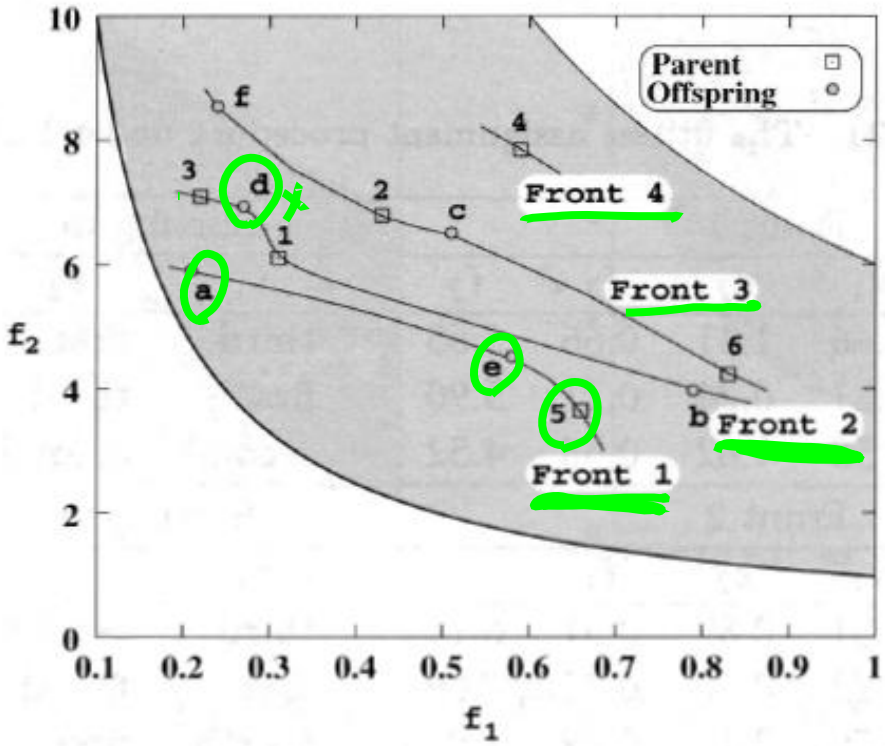
  $i = i + 1$ — check the next front for inclusion

Sort$(\mathcal{F}_i, \prec_n)$ ➔Crowding sort — sort in descending order using $\prec_n$

$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$ — choose the first $(N - |P_{t+1}|)$ elements of $\mathcal{F}_i$

$Q_{t+1} =$ `make-new-pop`(P_{t+1}) — use selection, crossover and mutation to create a new population Q_{t+1}

$t = t + 1$ — increment the generation counter



Step 1 We first combine the populations P_t and Q_t and form $R_t = \{1, 2, 3, 4, 5, 6, a, b, c, d, e, f\}$. Next, we perform a non-dominated sorting on R_t. We obtain the following non-dominated fronts:

$$\mathcal{F}_1 = \{5, a, e\},$$
$$\mathcal{F}_2 = \{1, 3, b, d\},$$
$$\mathcal{F}_3 = \{2, 6, c, f\},$$
$$\mathcal{F}_4 = \{4\}.$$

Step 2 We set $P_{t+1} = \emptyset$ and $i = 1$. Next, we observe that $|P_{t+1}| + |\mathcal{F}_1| = 0 + 3 = 3$. Since this is less than the population size N $(= 6)$, we include this front in P_{t+1}. We set $P_{t+1} = \{5, a, e\}$. With these three solutions, we now need three more solutions to fill up the new parent population.

Now, with the inclusion of the second front, the size of $|P_{t+1}| + |\mathcal{F}_2|$ is $(3 + 4)$ or 7. Since this is greater than 6, we stop including any more fronts into the population. Note that if fronts 3 and 4 had not been classified earlier, we could have saved these computations.

Step 3 Next, we consider solutions of the second front only and observe that three (of the four) solutions must be chosen to fill up three remaining slots in the new population. This requires that we first sort this subpopulation (solutions 1, 3, b, and d) by using the $<_c$ operator (defined above in Section 6.2.1). We calculate the crowded distance values of these solutions in the front by using the step-by-step procedure.

Step 3 A sorting according to the descending order of these crowding distance values yields the sorted set $\{3, b, 1, d\}$. We choose the first three solutions.

Step 4 The new population is $P_{t+1} = \{5, a, e\}$ $\{3, b, 1\}$. These population members are shown in Figure 141 by joining them with dashed lines. It is important to note that this population is formed by choosing solutions from the better non-dominated fronts.

The offspring population Q_{t+1} has to be created next by using this parent population. We realize that the exact offspring population will depend on the chosen pair of solutions participating in a tournament and the chosen crossover and mutation operators. Let us say that we pair solutions $(5, e)$, $(a, 3)$, $(1, b)$, $(a, 1)$, (e, b) and $(3, 5)$, so that each solution participates in exactly two

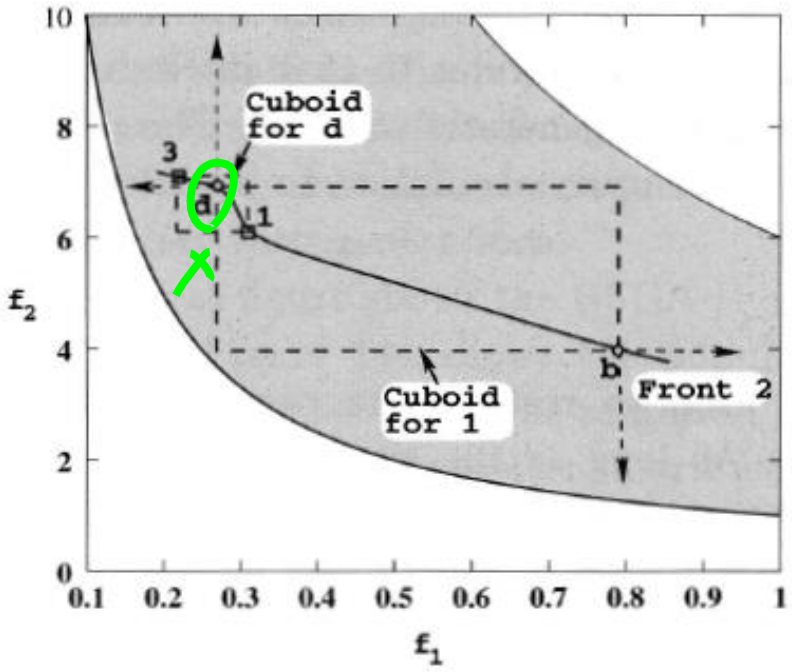


Figure 140 The cuboids of solutions 1 and d. The cuboids for solutions 3 and b extend to infinity.

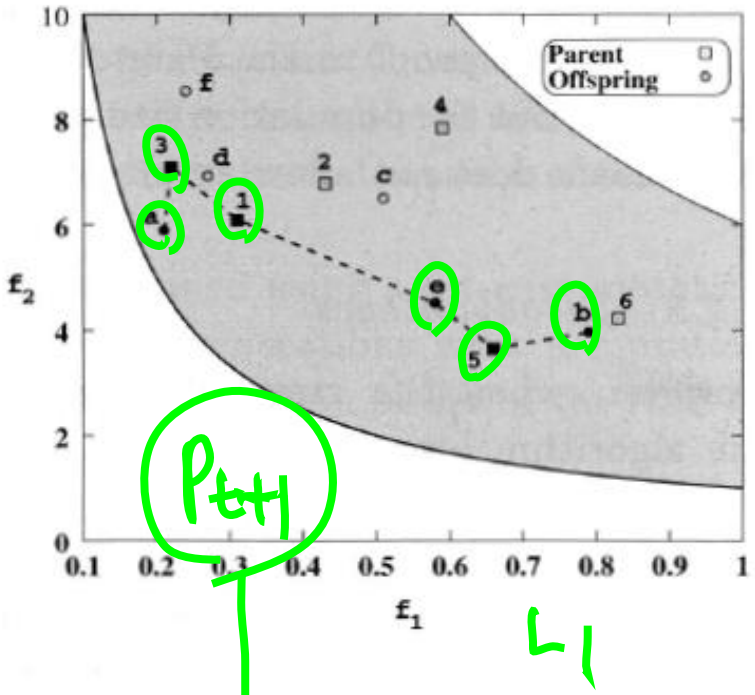
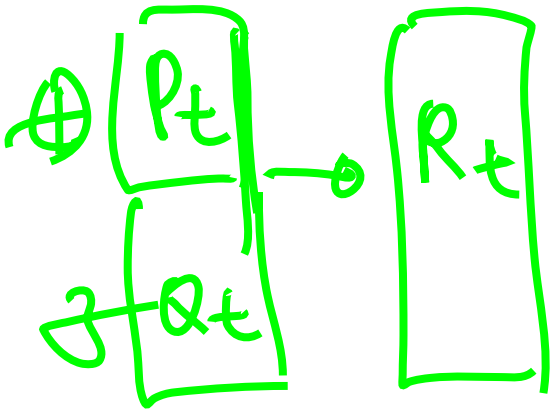


Figure 141 The new parent population P_{t+1} joined by dashed lines.

NSGA II

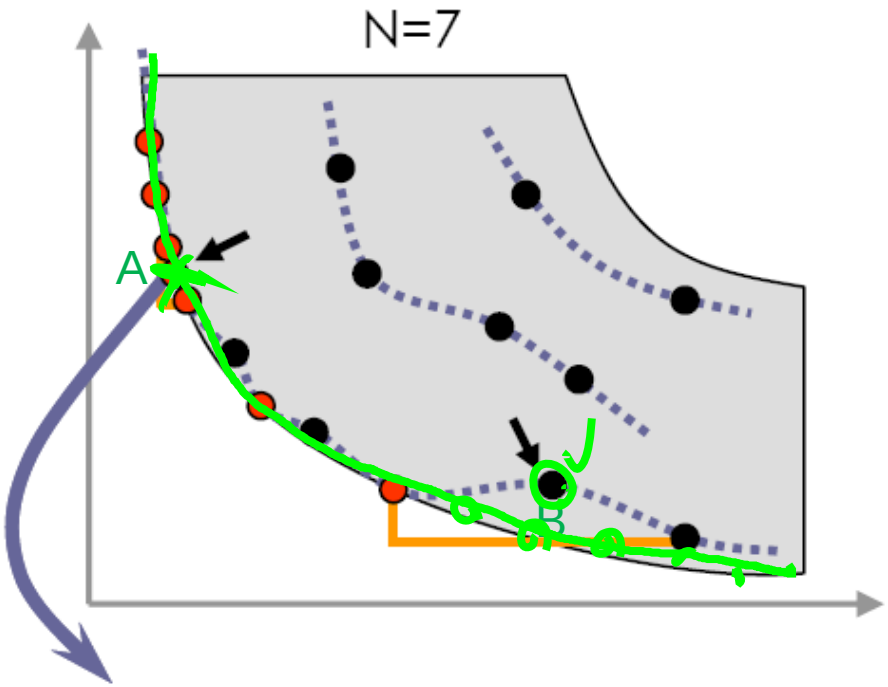
Elitist Non-dominated Sorting GA

- Advantages
  - The diversity among non-dominated solutions is maintained using the crowding procedure: No extra diversity control is needed
  - Elitism protects an already found Pareto-optimal solution from being deleted



Elitist Non-dominated Sorting GA

- Disadvantage
  - When there are more than N members in the first non-dominated set, some Pareto-optimal solutions may give their places to other non-Pareto-optimal solutions



A Pareto-optimal solution is discarded