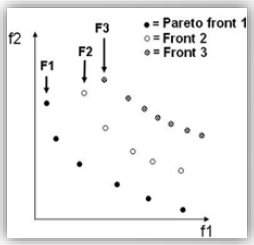
Non-dominated Sorting Genetic Algorithm II (NSGA-II)

CPE 354
OPTIMIZATION DESIGN AND EVOLUTIONARY COMPUTING

What is NSGA-II ?

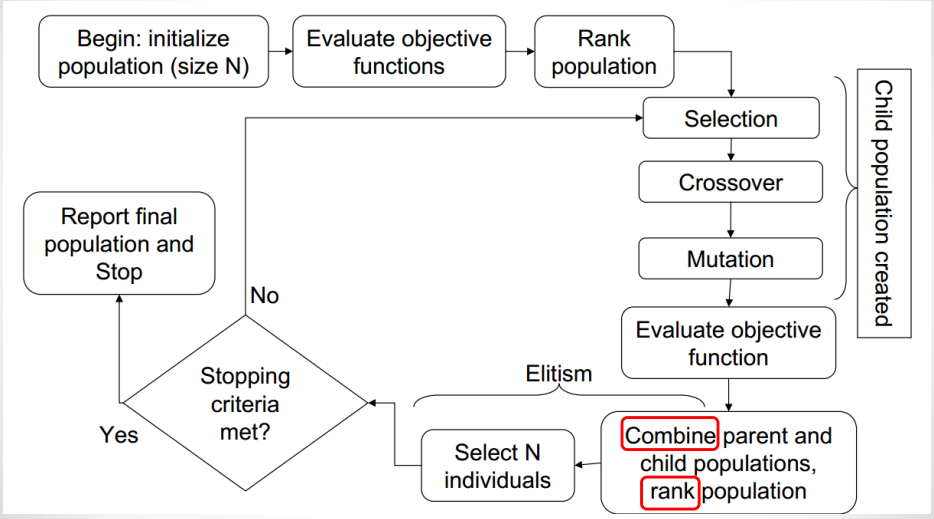
(Non-dominated sorting genetic algorithm II [1])

- A searching algorithm for finding **non-dominated solutions** or PF of multi-objective optimization problems.



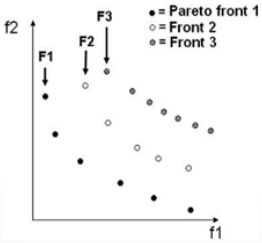
- **Three Main Features**
  - A **non-dominated sorting** where all the individuals are sorted according to the level of non-domination.
  - An **elitism** which stores all non-dominated solutions, that enhancing fast convergence properties.
  - A **crowding distance** that emphasize on less crowded solutions to maintain the diversity and spread of the solutions.

Flowchart of NSGA-II [2]



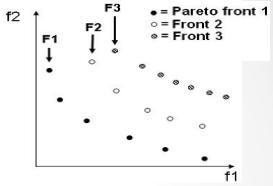
The Procedure of NSGA-II [1]

1. Randomly generate the first generation with size *N*.
2. Calculate objective functions of the first generation.
3. Sort population by **non-dominated sorting** approach.
   - Sorted the population into different fronts (*F1*, *F2*, *F3*, etc. 1 is the best level, 2 is the next-best level, and so on) according to non-domination levels.



Non-dominated Sorting [1]

```
fast-nondominated-sort(P)
for each p ∈ P
   for each q ∈ P
      if (p ≺ q) then                  if p dominates q then
         S_p = S_p ∪ {q}               include q in S_p
      else if (q ≺ p) then             if p is dominated by q then
         n_p = n_p + 1                 increment n_p
   if n_p = 0 then                     if no solution dominates p then
      F_1 = F_1 ∪ {p}                  p is a member of the first front
i = 1
while F_i ≠ ∅
   H = ∅
   for each p ∈ F_i                    for each member p in F_i
      for each q ∈ S_p                 modify each member from the set S_p
         n_q = n_q − 1                 decrement n_q by one
         if n_q = 0 then H = H ∪ {q}   if n_q is zero, q is a member of a list H
   i = i + 1
   F_i = H                             current front is formed with all members of H
```

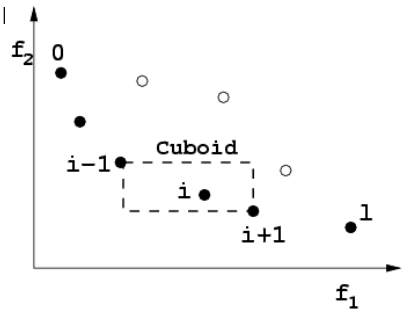


The Procedure of NSGA-II [2]

4. Used binary tournament selection to generate the population of parents in the current population.
   - The binary tournament selection **randomly selects two** solutions from the current population then selects the **better one** with respect to the non-domination rank.
   - Solutions at the **same non-domination front** are compared by a **crowding distance** (which is a measure of the density of the solutions at the neighborhood of that solution).

Crowding Distance

- Estimation of the density of **solutions surrounding** each mei



Crowding-distance calculation.
Points marked in filled circles are solutions of the same non-dominated front. [1]

Crowding Distance

```
1: procedure CROWDINGDISTANCE(F)
2:     N = |F|
3:     for i = 1...N do
4:         F[i]_dist = 0
5:     end for
6:     for m = 1,...,M do
7:         SORT(F, m)
8:         F[1]_dist = F[N]_dist = ∞
9:         for i = 2...N − 1 do
10:            F[i]_dist = F[i]_dist + (F[i+1].m − F[i−1].m) / (f_m^max − f_m^min)
11:        end for
12:    end for
13: end procedure
```

Crowding-distance computation algorithm. [5]

- The members of the front is sorted
  - According to each objective function from the lowest value to the highest value.

$(0,5)$
$(0,5)$
$(2,2)$
$(3,1)$
$(3,1)$
$(3,1)$
$(5,0)$

- The **extreme solutions** (the **smallest** and **largest** value) for each objective function are assigned a very **large distance.**
  - To guarantee that they will be selected in the next generation.

Note

F: a Pareto front composed of N individuals.

F[i].m : the m^{th} objective of the i^{th} individual in front ***F***.

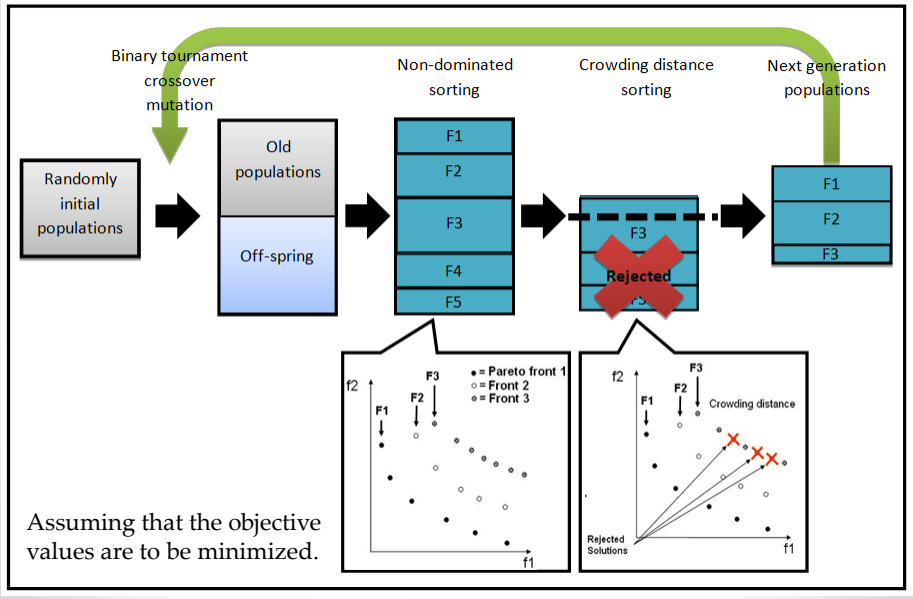
M: number of objectives

f_{min}_m, f_{max}_m: the minimum and maximum values for objective m.

The Procedure of NSGA-II (3)

4. Used the **genetic operators** (i.e. crossover and mutation) to generate an offspring population with size *N*
5. Calculate objective functions of the offspring population.
7. Combined population of parents and off-springs with size 2*N*.
8. Sorted population by **non-dominated sorting** approach.

How to create the next generation



The Procedure of NSGA-II [4]

9. Create the next generation by copying the best solutions (**elitism**) or the first *N* individuals from the mixed population of parents and off-springs. The solutions that are not copied are rejected.
   - The selection criteria are first the **non-domination rank** and then the **crowding distance**.
   - If the size of *F1* is smaller than size *N*, then choose all members of the set *F1* for the new population. The remaining members of the population are chosen from the ranking of non-dominated fronts. Thus, the solutions from the rank *F2* are chosen next, followed by solutions from the rank *F3*, and so on. This procedure is continued until the last rank exceeds the empty slot of new population.
   - Calculate the **crowding distance** for each potential solution from the last rank.
10. The procedure is terminated when a user-defined maximum number of generations is reached. Otherwise, increase generation number and continue with Step 4.

NSGA-II Implementation

- NSGA-II requires parameter tunings including
  - Population size
  - Mutation probability
  - Crossover probability
  - Maximum generation
- Implement NSGA-II from jMetal framework [3].
  - http://jmetal.sourceforge.net/
  - Java, C#, C++
- Implement NSGA-II from NGPM in Matlab v1.4
  - http://www.mathworks.com/matlabcentral/fileexchange/31166-ngpm-a-nsga-ii-program-in-matlab-v1-4
  - NGPM manual v1.4.pdf: http://goo.gl/M1puCx

Pseudo code for NSGA-II [4]

```
Input: Population_size, ProblemSize, P_crossover, P_mutation
Output: Children
Population ← InitializePopulation(Population_size, ProblemSize)
EvaluateAgainstObjectiveFunctions(Population)
FastNondominatedSort(Population)
Selected ← SelectParentsByRank(Population, Population_size)
Children ← CrossoverAndMutation(Selected, P_crossover, P_mutation)
While (¬StopCondition())
  EvaluateAgainstObjectiveFunctions(Children)
  Union ← Merge(Population, Children)
  Fronts ← FastNondominatedSort(Union)
  Parents ← ∅
  Front_L ← ∅
  For (Front_i ∈ Fronts)
    CrowdingDistanceAssignment(Front_i)
    If (Size(Parents)+Size(Front_i) > Population_size)
      Front_L ← i
      Break()
    Else
      Parents ← Merge(Parents, Front_i)
    End
  End
  If (Size(Parents)<Population_size)
    Front_L ← SortByRankAndDistance(Front_L)
    For (P_1 To P_Population_size−SizeFront_L)
      Parents ← Pi
    End
  End
  Selected ← SelectParentsByRankAndDistance(Parents, Population_size)
  Population ← Children
  Children ← CrossoverAndMutation(Selected, P_crossover, P_mutation)
End
Return (Children)
```

References

1. Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000). A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182-197.
2. Tushar Goel, Elitist Non-dominated Sorting Genetic Algorithm: NSGA-II, http://www2.mae.ufl.edu/haftka/stropt/Lectures/multi_objective_GA.pdf
3. Durillo, J.J., Nebro, A.J.: jMetal: A java framework for multiobjective optimization. Adv. Eng. Softw. 42, 760–771 (2011)
4. J. Brownlee, Clever Algorithms: Nature-Inspired Programming Recipes, Clever Algorithms: Nature-Inspired Programming Recipes, Lulu, 2011
5. Fortin, Félix-Antoine, and Marc Parizeau. "Revisiting the NSGA-II crowding-distance computation." Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, 2013.