Multi-objective Optimization

Jugal K. Kalita

Single vs. Multi-objective Optimization

- **Single Objective Optimization:** When an optimization problem involves only one objective function, the task of finding the optimal solution is called single-objective optimization.
- *Example:* Find a CAR for me with Minimum cost.
- **Multi-objective Optimization:** When an optimization problem involves more than one objective function, the task of finding one or more optimal solutions is known as multi-objective optimization.
- *Example:* Find a CAR for me with minimum cost and maximum comfort.

Multi-objective Problems

- Problems with more than one objective function, typically conflicting objectives
- Cars Example: Maximize luxury vs. Minimize price
- Another example: Maximiize $f_1(x) = x^2$ and $f_2(x) = (x-2)^2$ when x is in the range $[-10^3, 10^3]$

Multi-objective Problems

- A more complex example: Maximize the following two objective functions when the variables are within the range $[-\pi, \pi]$:

$$f_1(x) = \left[1 + (A_1 - B_1)^2 + (A_1 - B_2)^2\right]$$

$$f_2(x) = \left[(x_1 + 3)^2 + (x_2 + 1)^2\right]$$

$$A_1 = 0.5sin(1) - 2cos(1) + sin(2) - 1.5cos(2)$$

$$A_2 = 1.5sin(1) - cos(1) + 2sin(2) - 0.5cos(2)$$

$$B_1 = 0.5sin(x_1) - 2cos(x_1) + sin(x_2) - 1.5cos(x_2)$$

$$B_2 = 1.5sin(x_1) - cos(x_1) + 2sin(x_2) - 0.5cos(x_2)$$

General Formulation of a Multi-objective Problem

- Minimiize $\mathbf{F}(X)$

where

$$\mathbf{F}(X) = \{f_i : \forall i = 1, M\}$$
$$\mathbf{X} = \{X_j : \forall j = 1, N\}$$

subject to

$$\mathbf{C}(X) \leq 0 \quad \text{where} \quad \mathbf{C} = \{C_k : \forall k = 1, P\}$$
$$\mathbf{H}(X) = 0 \quad \text{where} \quad \mathbf{H} = \{H_k : \forall k = 1, Q\}$$

Multi-objective Optimization

- Multi-objective optimization (MOO) is the optimization of conflicting objectives.
- In some problems, it is possible to find a way of combining the objectives into a single objective.
- But, in some other problems, it is not possible to do so.
- Sometimes the differences are qualitative and the relative importance of these objectives cannot be numerically quantified.
- Frequently, there is no unique solutions, rather there is a trade-off among optimal solutions.

Conceptual Example

- Suppose you need to fly on a long trip: Should you choose the cheapest ticket (more connections) or shortest flying time (more expensive)?
- It is impossible to put a value on time, so these two objectives cannot be linked.
- Also, the relative importance will vary.
  - There may be a business emergency you need to go fix quickly.
  - Or, maybe you are on a very tight budget.

Pareto-Optimal Solutions

- A MOO problem with constraints will have many solutions in the feasible region.
- Even though we may not be able to assign numerical relative importance to the multiple objectives, we can still classify some possible solutions as better than others.

Example of Pareto-Optimal Solutions

- Suppose in our airplane-trip example from earlier, we find the following tickets.

Ticket	Travel Time (hrs)	Price ($)
A	10	1700
B	9	2000
C	8	1800
D	7.5	2300
E	6	2200

Comparison of Solutions

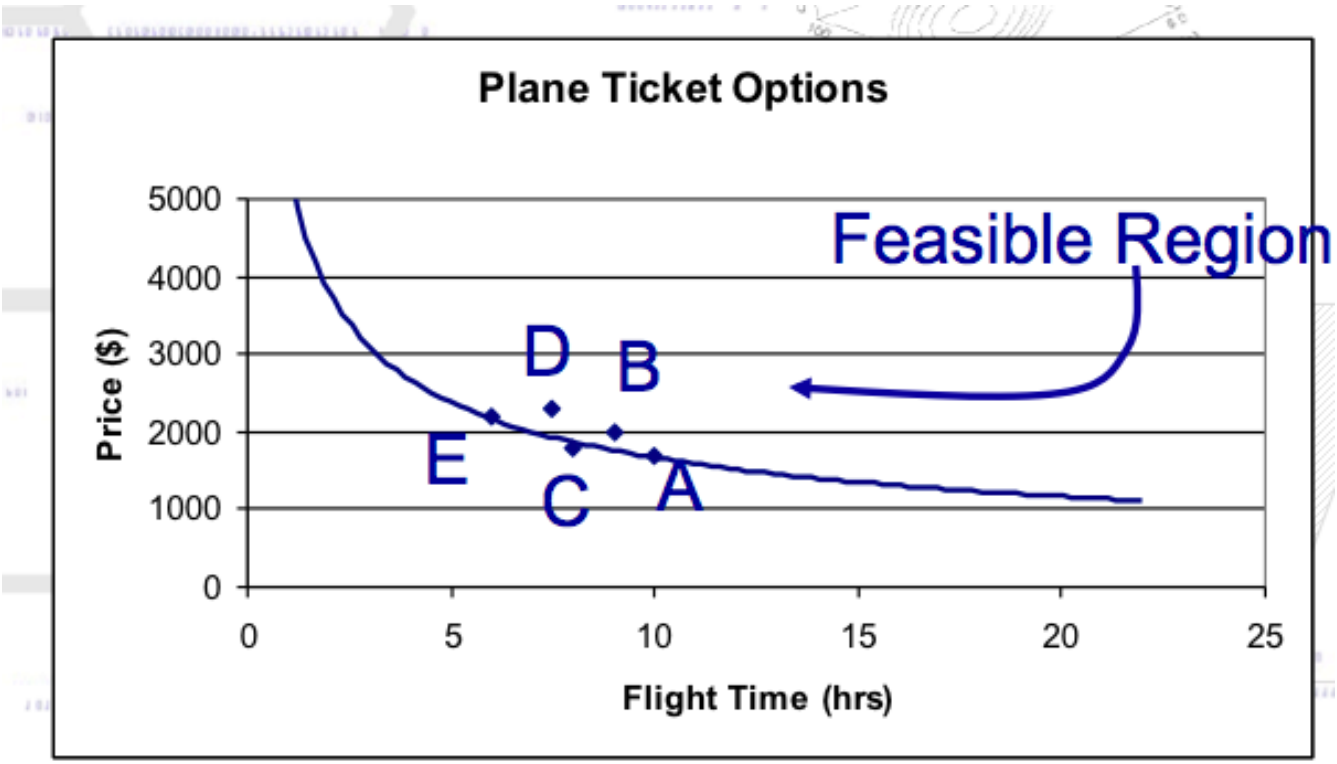
- If we compare tickets A and B, we cannot say that either is superior without knowing the relative importance of Travel Time vs. Price.
- However, comparing tickets B and C shows that C is better than B in **both** objectives, so we can say that C "dominates" B.
- So, as long as C is a feasible option, there is no reason we would choose B.

Comparison of Solutions

- If we finish the comparisons, we also see that D is dominated by E.
- The rest of the options (A, C, and E) have a trade-off associated with Time vs. Price, so none is clearly superior to the others.
- We call $\{A, C, E\}$ the *non-dominated* set of solutions because none of the solutions are dominated.

Graph of Solutions

- Usually, solutions of this type form a typical shape, shown in the graph below.



Types of Solutions

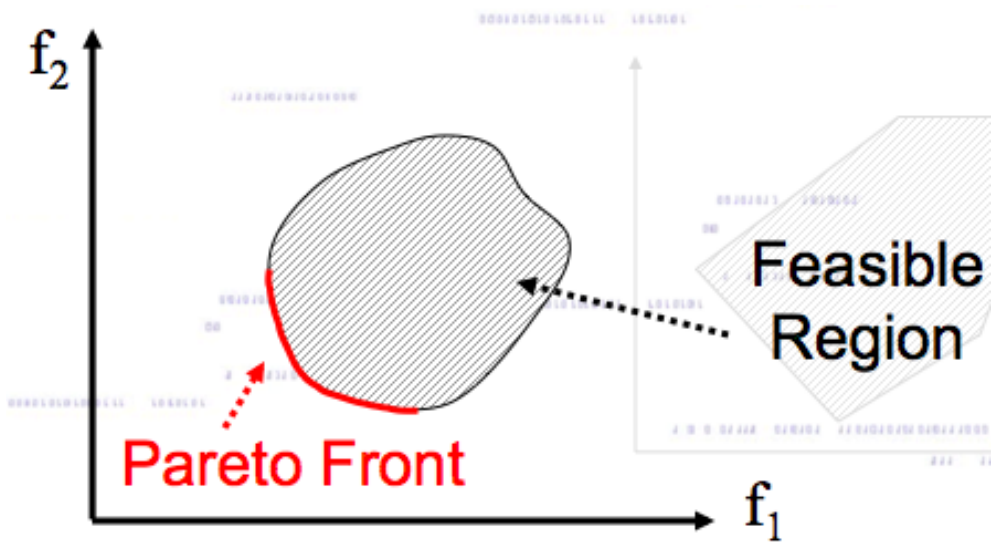
- Solutions that lie along the line are non-dominated solutions while those that lie inside the line are dominated because there is always another solution on the line that has at least one objective that is better.

Pareto-optimal Solutions

- The line is called the Pareto front and solutions on it are called Pareto-optimal.
- All Pareto-optimal solutions are non-dominated.
- Thus, it is important in MOO to find the solutions as close as possible to the Pareto front and as far along it as possible.

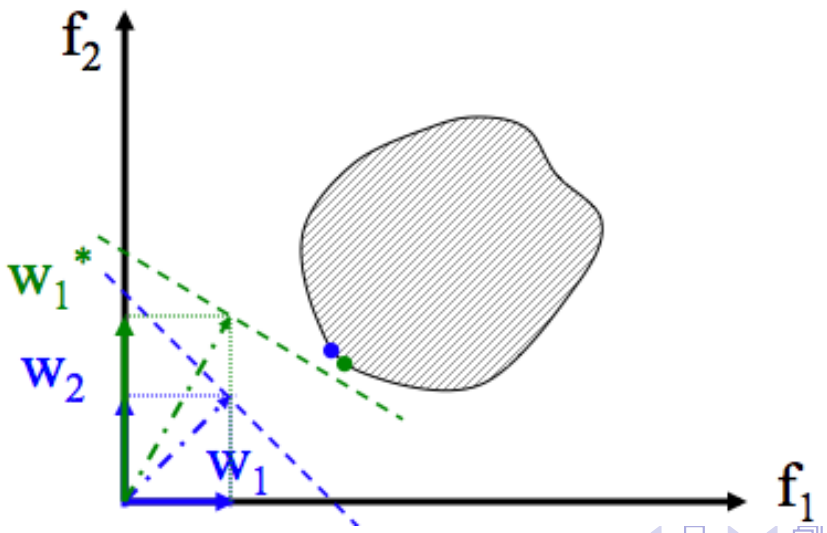
Finding the Pareto Front

- For the following feasible region with objectives f_1 and f_2 where both f_1 and f_2 are minimized.



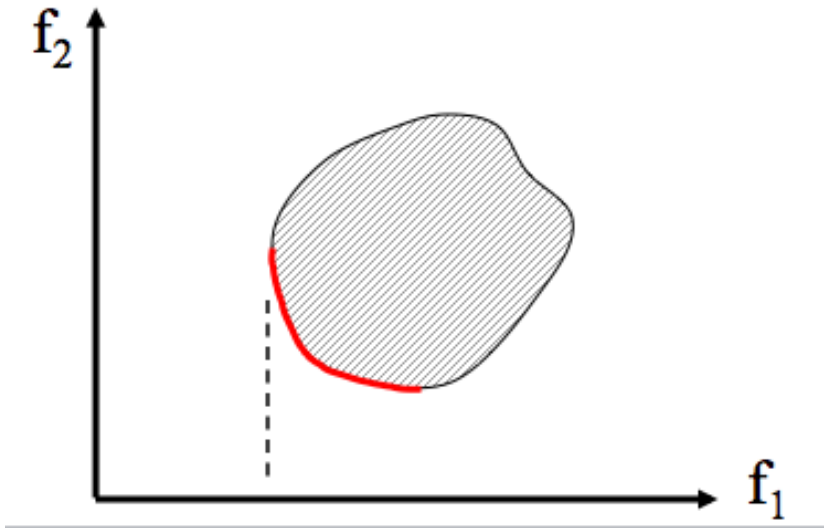
Graphical Example

- One way to imagine finding points on the Pareto front is by using a combination of numerical weights for the two objectives.
- Here, we have a weight of w_1 on f_1 and weight w_2 on f_2. The equation of a slanted line is $w_1 f_1 + w_2 f_2$ for various values of w_1 and w_2 such that $w_1 + w_2 = 1$.



Finding the Pareto Front

- If this is done for a 90° span of lines, all the points on the Pareto front will be found.



Realistic Procedures

- There are different methods used in practice, but one is to use a genetic algorithm to enumerate points along the Pareto front over several iterations, then use some method to rank the quality of the trade-offs based on the particular application being modeled.

Practicality of this Procedure

- Actually, this is not the procedure that is used in practice, but it is a good illustration of the concept.
- This procedure would require finding all possible points in the feasible region and then using many combinations of weights.
- For more than two objectives, the complexities and the number of combinations make this impractical.

Finding a Point on the Pareto Front

- It should be remembered that each point on the Pareto front is found by solving an optimization problem.
- If the problem is nonlinear or very complex, the simple step of getting just one solution may not be trivial.
- This is the case whether we a genetic algorithm or some other method. We have to find points on the Pareto Front.

Development of Multiobjective GAs

- *The 1st generation:* Vector Evaluation approach Vector Evaluated Genetic Algorithm (VEGA)
  - J. D. Schaffer: "Multiple objective optimization with vector evaluated genetic algorithms", *Proc. of 1st Inter. Conf. on GAs and Their Applic.*, 1985.
- *The 2nd generation*: Pareto ranking + Diversity
  - Multiobjective Genetic Algorithm (MOGA)
    Fonseca, C. M. and Fleming, P. J: "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization", Proc. of the 5th Inter. Conf. on GAs, 1993.
  - *Non-dominated Sorting Genetic Algorithm* (NSGA)
    Srinivas, N. and Deb, K.: "Multiobjective function optimization using nondominated sorting genetic algorithms", Evolutionary Computation, 1995.

Development of Multiobjective GAs

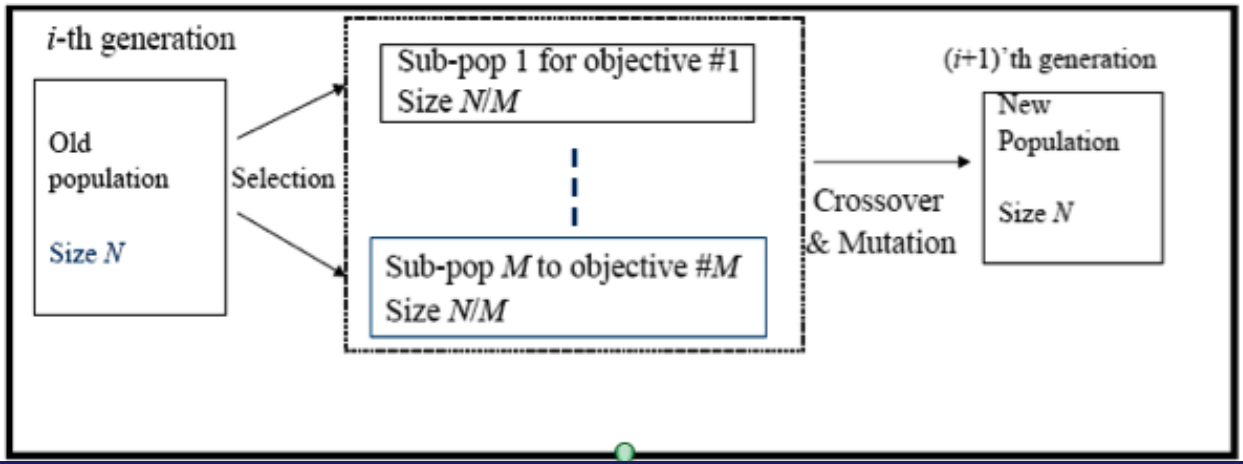
- *The 3rd generation: Weighted Sum + Elitist Preserve*
  - Random-weight Approach (RWA): Ishibuchi, H and T. Murata: “A multiobjective genetic local search algorithm and its application to flowshop scheduling”, IEEE Trans. on Sys., Man & Cyber., 1998.
  - Strength Pareto Evolutionary Algorithm (SPEA): E. Zitzler and L. Thiele: “Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach”, IEEE Trans. on Evolutionary Comput., 1999.
  - Adaptive Weight Approach (AWA): Gen M,. and Cheng, R. : Genetic Algorithms and Engineering Optimization, John Wiley & Sons, New York, 2000.
  - Non-dominated Sorting Genetic Algorithm II (NSGAII): Deb, K., A. Pratap, S. Agarwal & T. Meyarivan: “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II” IEEE Trans. on Evolutionary Comput., 2002

Vector Evaluated Genetic Algorithms (Schaffer, 1985)

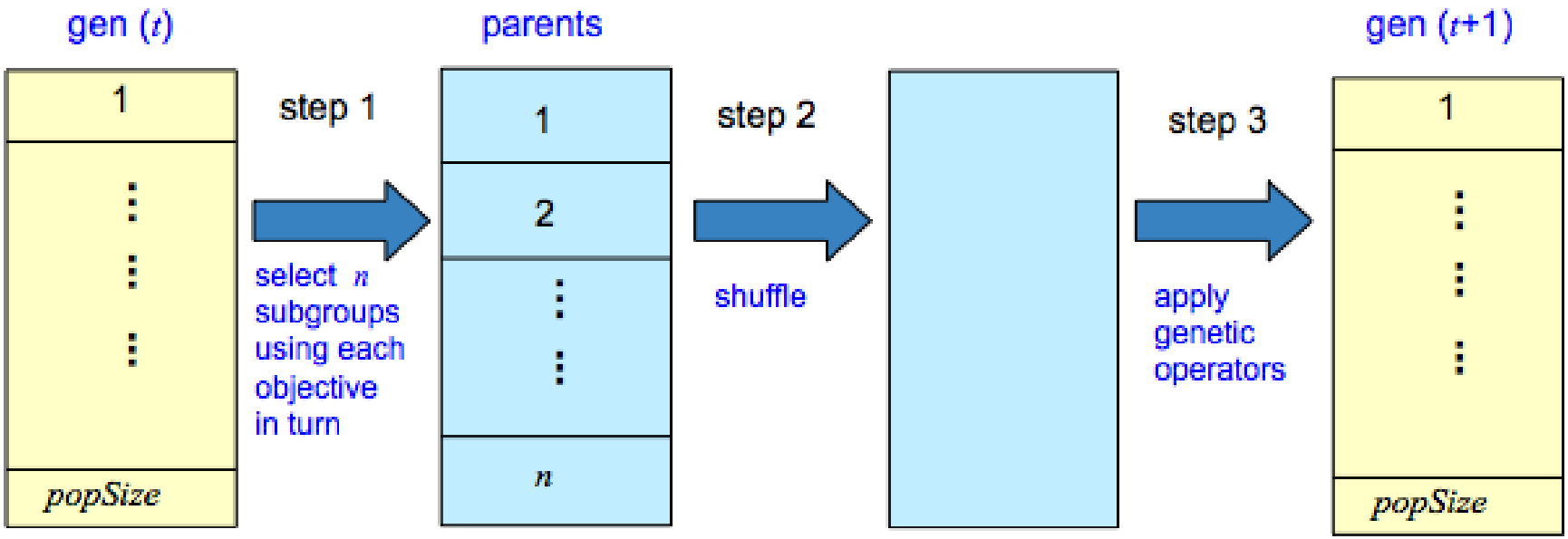
- VEGA is the first notable work to solve MOO problems in which it uses a vector fitness measure to create the next generation.
- The selection step in each generation becomes a loop. Each time through the loop the appropriate fraction of the next generation, or subpopulation, is selected on the basis of each objective.
- The entire population is shuffled thoroughly to apply crossover and mutation operators. This is performed to achieve the mating of individuals of different subpopulations.

VEGA 1985

- Divide population into M equal blocks for M objectives at every generation.
- Reproduce each block with one objective function.
- Complete population participates in crossover and mutation.
- Use proportionate selection operator.
- To reduce the positional bias in the population, shuffle the population before partition.



VEGA 1985


gen (t)
1
popSize
step 1
select n subgroups using each objective in turn
parents
1
2
n
step 2
shuffle
step 3
apply genetic operators
gen (t+1)
1
popSize

Pareto Ranking Approach

- Pareto ranking approach consists of assigning fitness ranks to chromosomes based on the level of non-domination.
- Best ranked chromosomes have better chance at reproduction.
- A Pareto ranking-based fitness assignment method was first suggested by
  - Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

Pareto Ranking (Goldberg 1989)

- *Step 1:* Assign rank 1 to the nondominated chromosomes
- *Step 2:* Remove rank 1 chromosomes from contention
- *Step 3:* Find the next nondominated chromosomes, remove them from contention, assign rank 2 to them.
- *Step 4:* Continue until the entire population is ranked.

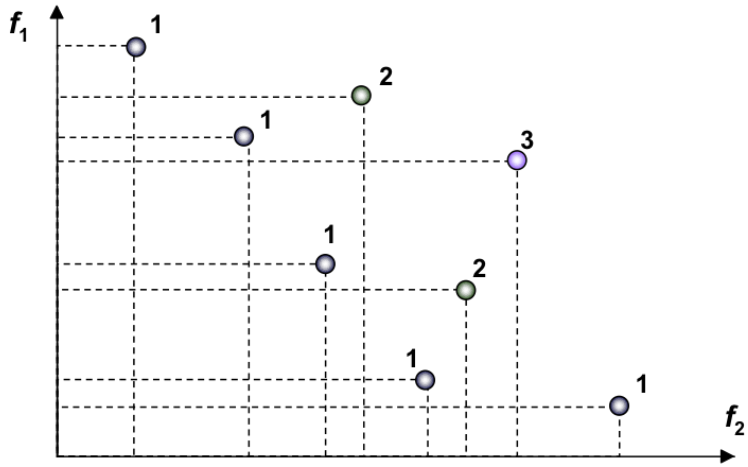


Figure: Simple case with two objectives to be minimized using Goldberg's Ranking

Multi-objective Genetic Algorithm (Fonseca & Fleming 1993)

- *Step 1:* Assign rank 1 to the nondominated chromosomes
- *Step 2:* Remove rank 1 chromosomes from contention
- *Step 3:* Find the next nondominated chromosomes, remove them from contention, assign rank equal to the number of a chromosome's dominating individuals +1.
- *Step 4:* Continue until the entire population is ranked.

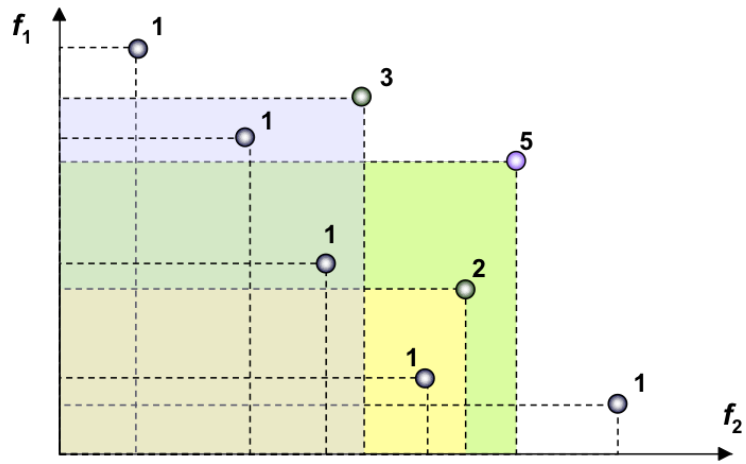


Figure: Simple case with two objectives to be minimized using MOGA

Non-dominated Sorting Genetic Algorithm (Deb 1995)

- *Step 1:* Identify nondominated individuals, and assign a large dummy fitness value.
- *Step 2:* To maintain diversity in the population, share these individuals with their dummy fitness values.
- *Step 3:* After sharing, ignore these nondominated individuals temporarily.
- *Step 4:* Identify the second nondominated front in the rest of the population and assign a dummy fitness value smaller than the minimum shared dummy fitness of the previous front.
- *Step 5*: Continue this process until the entire population is classified into several fronts.

Salient Points about NSGA

- Assign fitness first to the non-dominated set and successively to dominated sets.
- A solution i of the first (or best) non-dominated set is assigned a fitness value $F_i = N$ where N is the population size.
- Assigning higher fitness to solutions belonging to a better non-dominated set ensures selection pressure toward the Pareto-optimal front.
- Maintain diversity of solutions in a specific front by reducing the assigned fitness based on the crowdedness (the number of neighboring solutions) of a solution. This is done using a *sharing function*.
- Degrading the fitness of solutions in a crowded area in a certain front emphasizes solutions in less crowded regions.

NSGA...

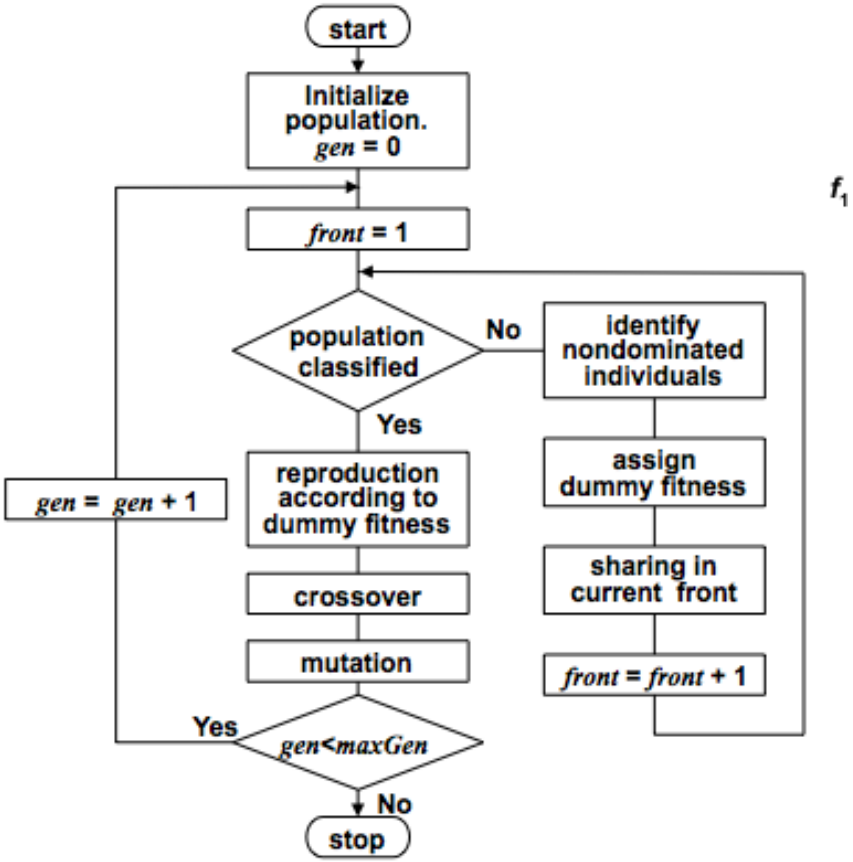


Figure: Flow chart of NSGA

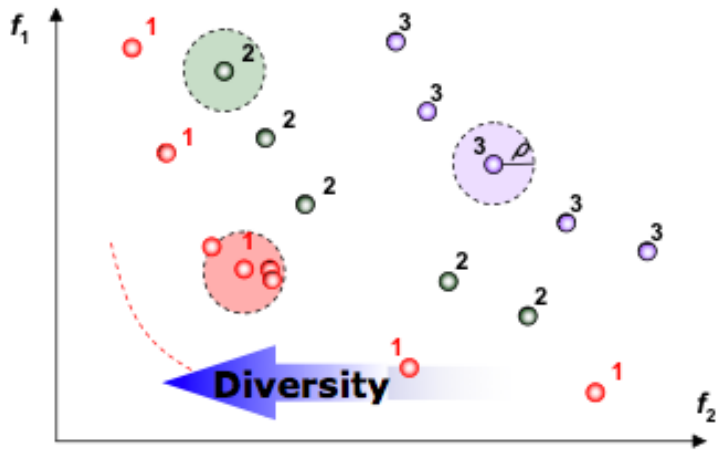


Figure: Simple case with two objectives to be minimized using NSGA

NSGA Analysis

- It is relatively easy to implement.
- It needs a parameter called the *sharing factor* to decide how the fitnesses of crowded solutions in a specific front should be shared.
- The solutions it obtains to a problem seem to be very sensitive to the sharing factor.

Non-Dominated Sorting Genetic Algorithm NSGA-II

- Deb et al. (2000, 2002) proposed a new version of NSGA called NSGA-II
- NSGA-II is more efficient computationally speaking.
- It uses elitism and doesn't need a sharing factor or parameter.

Innovations in NSGA-II

- There are three innovations.
  - A Fast non-dominated sorting algorithm for solutions
  - A way to efficiently compute what's called *crowding distance*.
  - A simple crowded distance comparison
- We will look at all three of these. First, let's look at the main algorithm.

Non-Dominated Sorting Genetic Algorithm NSGA-II

- Steps before main loop begins
  - Initialize population P_0
  - Sort P_0 on the basis of nondomination
  - Fitness of an individual is equal to an individual's nondomination level
  - Binary Tournament Selection
  - Mutation and recombination create an offspring population Q_0

Non-Dominated Sorting Genetic Algorithm NSGA-II

- Primary loop
  - $R_t = P_t + Q_t$ (t= generation)
  - Sort R_t on the basis of nondomination
  - Create P_{t+1} by adding individuals from each level of R_t until P_{t+1} is of size N
  - Create Q_{t+1} by applying Binary Tournament Selection, Mutation, and Recombination to P_{t+1}
  - Increment t

NSGA Flowchart

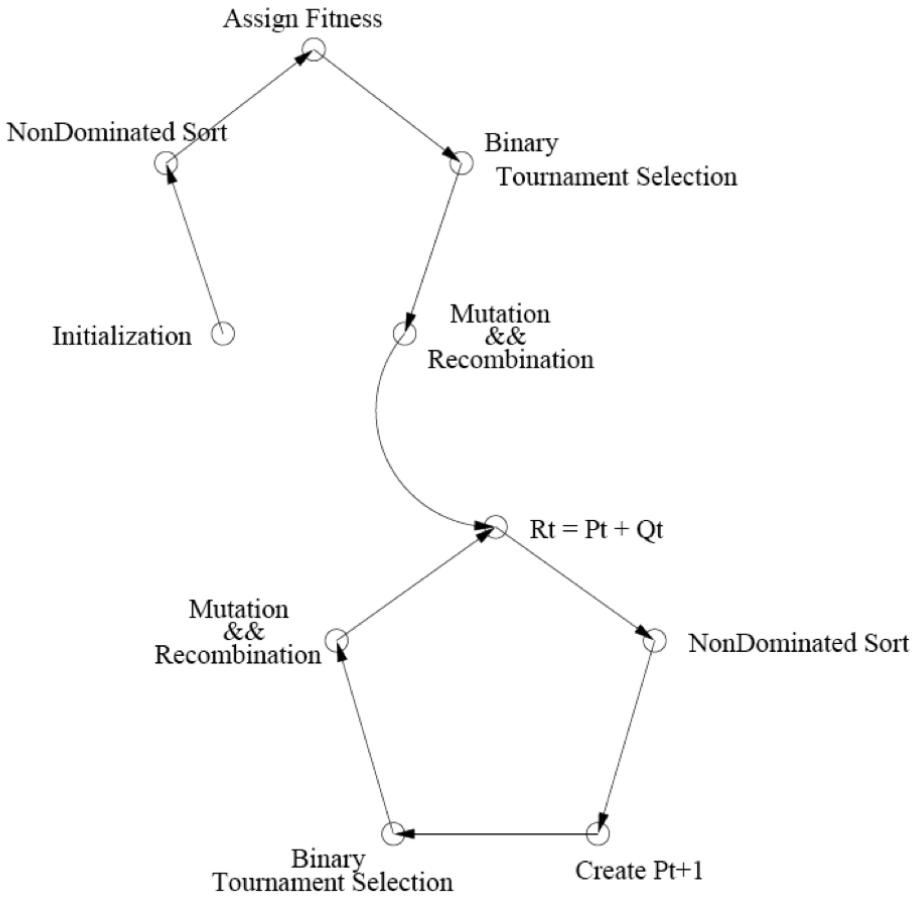


Figure: Flow chart of NSGA II

NSGA Flow Diagram

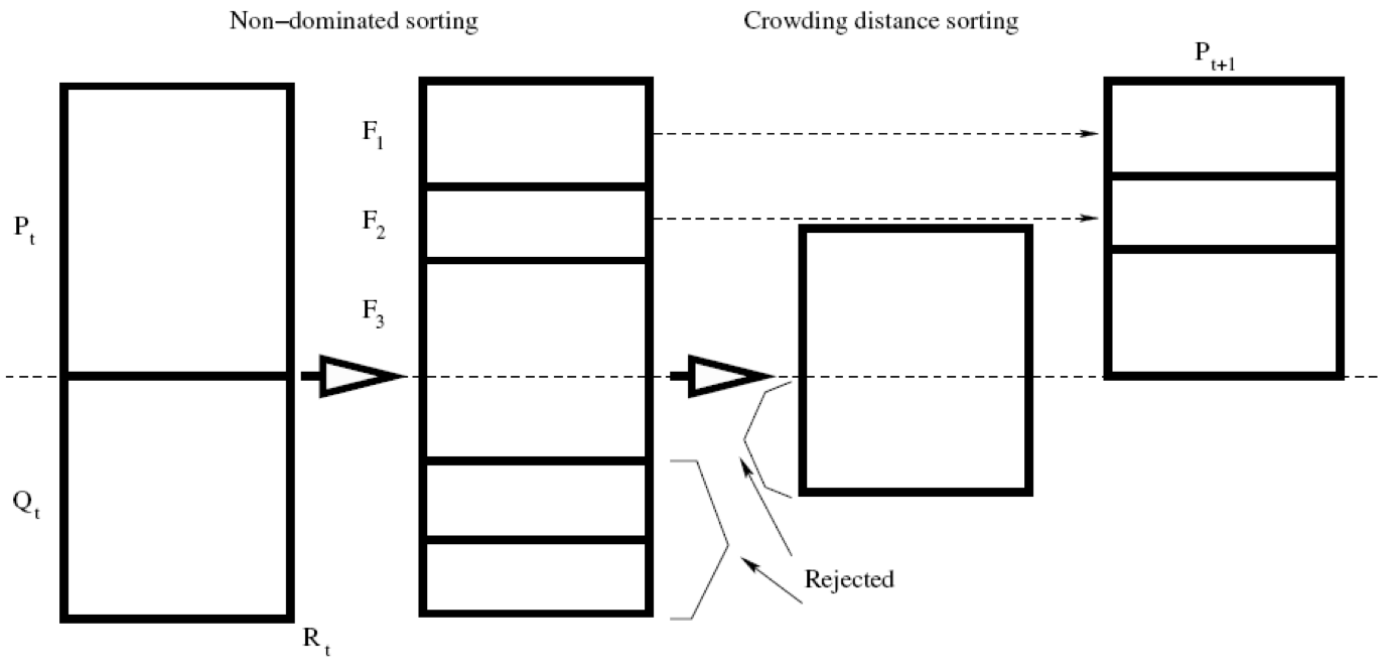


Figure: P_t is the parent population and Q_t is the offspring population for generation t. F_1 are the best solutions from the combined population, parents and offspring. F_2 are the next best solutions, etc.

Non-dominated Soring

- For each solution, calculate two entities: 1) domination count n_p, the number of solutions that dominate the solution p, and 2) S_p, a set of solutions that the solution p dominates.
- All solutions in the first non-dominated front have their domination count as 0.
- For each solution p with $n_p = 0$, we visit each member q of its set S_p and reduce its domination count by 1. In doing so, if for any member q, the domination count becomes 0, we put it in a separate list Q. Q is the second non-dominated front.
- Continue the above procedure with each member of Q and the third front is identified.
- Continue till all fronts are identified.

Non-dominated Sorting

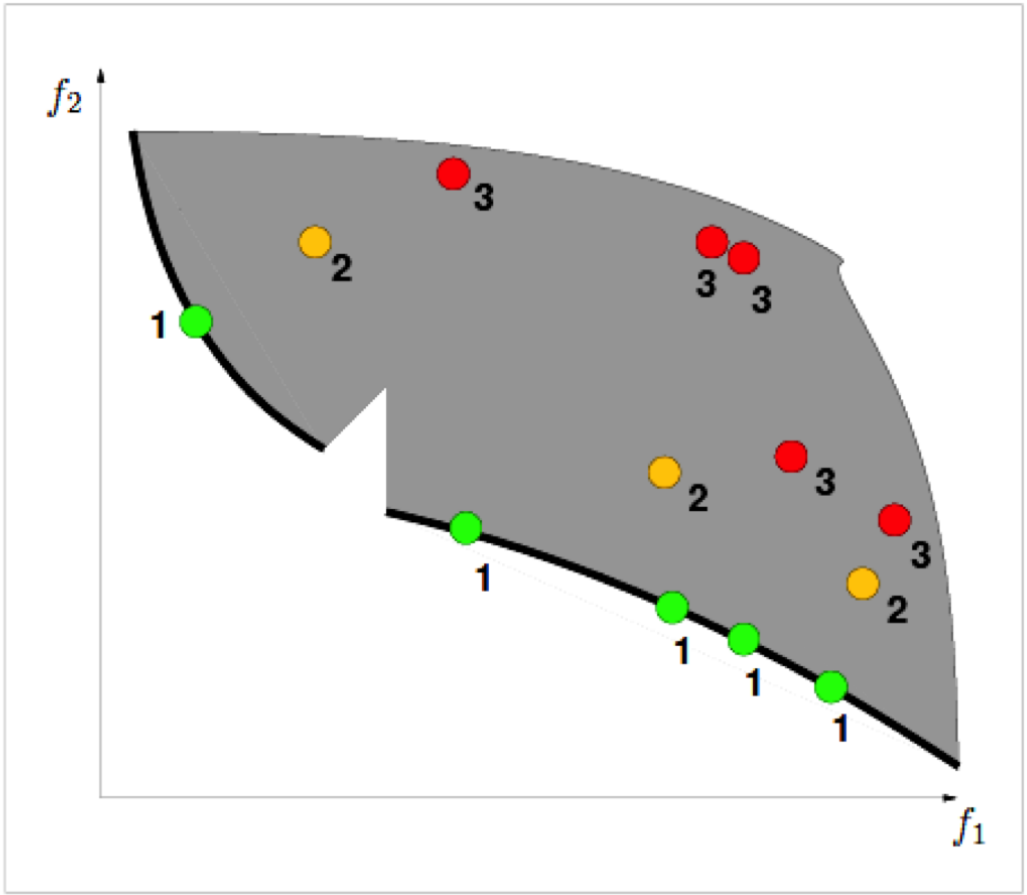


Figure: Result of non-dominated sorting

Non-dominated sorting

1: k=0
2: **repeat**
3: $k \longleftarrow k + 1$
4: $R.k \longleftarrow$ non-dominated solutions in P
5: $P \longleftarrow P - R.k$
6: **until** $P = \Phi$

Here $R_1 \cdots R_k$ denote partitions of rank $i = 1 \cdots k$

Diversity Preservation

- It is desired that there is a good spread of solutions.
- This is done by estimating the density of the solutions and by defining a crowded-comparison operator.
- To estimate the density of solutions surrounding a particular solution in the population, we sort all solutions in each objective dimension and calculate the average distance of two points on either side of this point along each of the objectives.
- See algorithm at the bottom of page 185 to see how each solution is given a crowding distance along each objective dimension.

Crowding Distance Metric

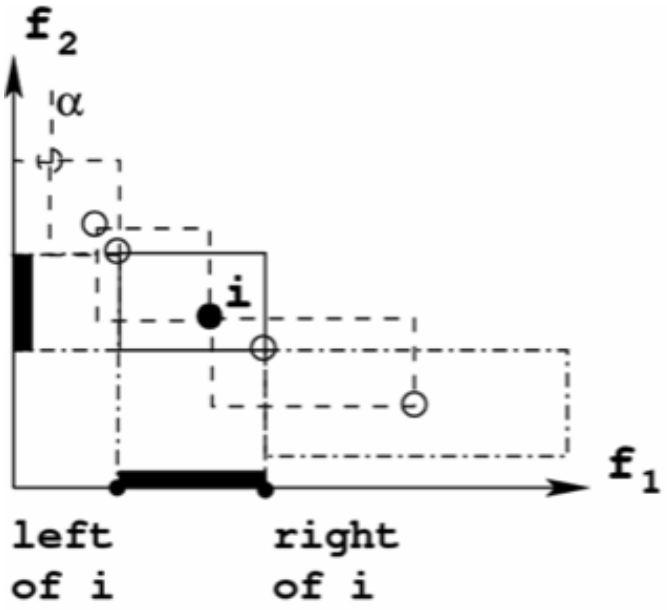


Figure: Crowding distance calculation. For solution *i* find the average distance from the two solutions immediately before and immediately after, along each objective dimension.

Crowded-Comparsion Operator

- The crowded-comparison operator ($\prec_n$) is used to select solutions at various stages of the algorithm toward a uniformly spread-out Pareto-optimal front.
- Assume that every individual i in the population has two attributes:
  1. Non-domination rank i_{rank}
  2. crowding distance $i_{distance}$
- We now define a partial order $\prec_n$ as $i \prec_n$ if $(I_{rank} < j_{rank})$ or $((i_{rank} = j_{rank})$ and $(i_{distance} > j_{distance}))$