Optimization Techniques

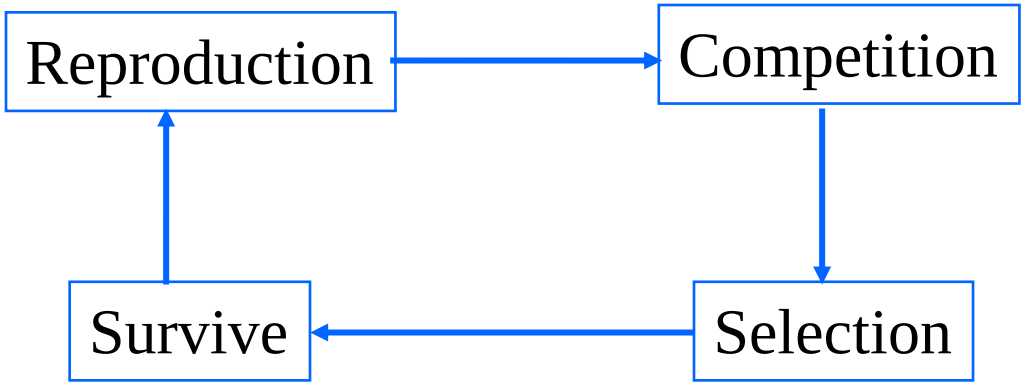
Genetic Algorithms

Optimization Techniques

- Mathematical Programming
- Network Analysis
- Branch & Bound
- Genetic Algorithm
- Simulated Annealing
- Tabu Search

Genetic Algorithm

- Based on Darwinian Paradigm



- Intrinsically a robust search and optimization mechanism

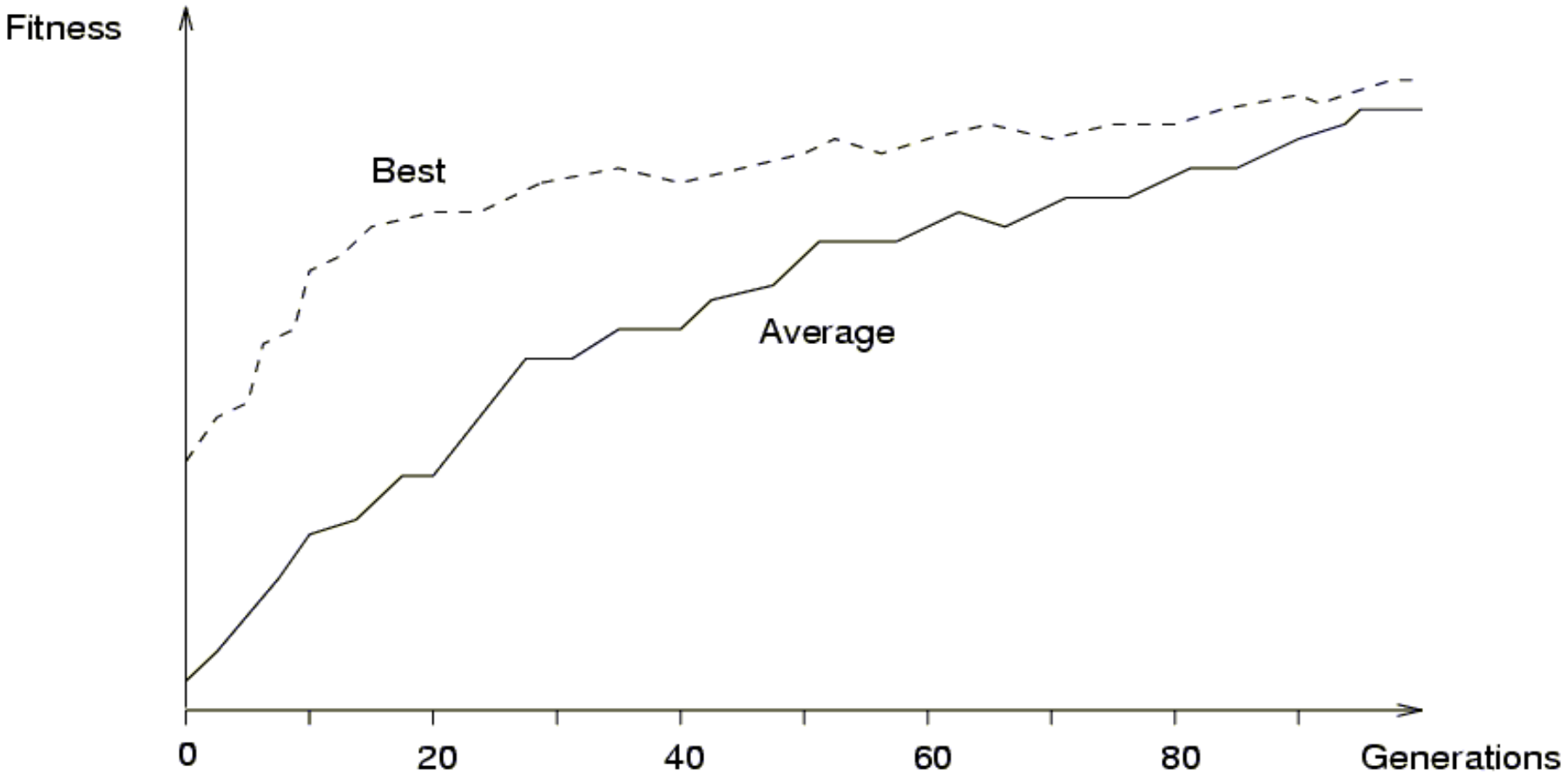
Genetic Algorithm Introduction 1

- Inspired by natural evolution
- Population of individuals
  - Individual is feasible solution to problem
- Each individual is characterized by a Fitness function
  - Higher fitness is better solution
- Based on their fitness, parents are selected to reproduce offspring for a new generation
  - Fitter individuals have more chance to reproduce
  - New generation has same size as old generation; old generation dies
- Offspring has combination of properties of two parents
- If well designed, population will converge to optimal solution

Algorithm

```
BEGIN
  Generate initial population;
  Compute fitness of each individual;
  REPEAT /* New generation /*
    FOR population_size / 2 DO
      Select two parents from old generation;
       /* biased to the fitter ones */
      Recombine parents for two offspring;
      Compute fitness of offspring;
      Insert offspring in new generation
    END FOR
  UNTIL population has converged
END
```

Example of convergence



Introduction 2

- Reproduction mechanisms have no knowledge of the problem to be solved

- Link between genetic algorithm and problem:
  - Coding
  - Fitness function

Basic principles 1

- Coding or Representation
  - String with all parameters
- Fitness function
  - Parent selection
- Reproduction
  - Crossover
  - Mutation
- Convergence
  - When to stop

Basic principles 2

- An individual is characterized by a set of parameters: Genes
- The genes are joined into a string: Chromosome

- The chromosome forms the genotype
- The genotype contains all information to construct an organism: the phenotype

---

- Reproduction is a "dumb" process on the chromosome of the genotype
- Fitness is measured in the real world ('struggle for life') of the phenotype

Coding

- Parameters of the solution (genes) are concatenated to form a string (chromosome)
- All kind of alphabets can be used for a chromosome (numbers, characters), but generally a binary alphabet is used
- Order of genes on chromosome can be important
- Generally many different codings for the parameters of a solution are possible
- Good coding is probably the most important factor for the performance of a GA
- In many cases many possible chromosomes do not code for feasible solutions

Genetic Algorithm

- Encoding
- Fitness Evaluation
- Reproduction
- Survivor Selection

Encoding

- Design alternative → individual (chromosome)
- Single design choice → gene
- Design objectives → fitness

Example

- Problem
  - Schedule *n* jobs on *m* processors such that the maximum span is minimized.

Design alternative: job i (i=1,2,…n) is assigned to processor j (j=1,2,…,m)

Individual: A n-vector **x** such that $x_i = 1, \ldots, or\ m$

Design objective: minimize the maximal span

Fitness: the maximal span for each processor

$$span_{j\in\{1,2,..m\}} = \sum_{i=1}^{n} C_j M_{ij}$$

$$, where C_j \text{ is the job length and } M_{ij} = 1 \text{ if } i_{th} \text{ job is allocated to } j_{th} \text{ processor}$$

Example of coding for TSP

Travelling Salesman Problem

- Binary
  - Cities are binary coded; chromosome is string of bits
    - Most chromosomes code for illegal tour
    - Several chromosomes code for the same tour
- Path
  - Cities are numbered; chromosome is string of integers
    - Most chromosomes code for illegal tour
    - Several chromosomes code for the same tour
- Ordinal
  - Cities are numbered, but code is complex
  - All possible chromosomes are legal and only one chromosome for each tour
- Several others

Reproduction

- Reproduction operators
  - Crossover
  - Mutation

Reproduction

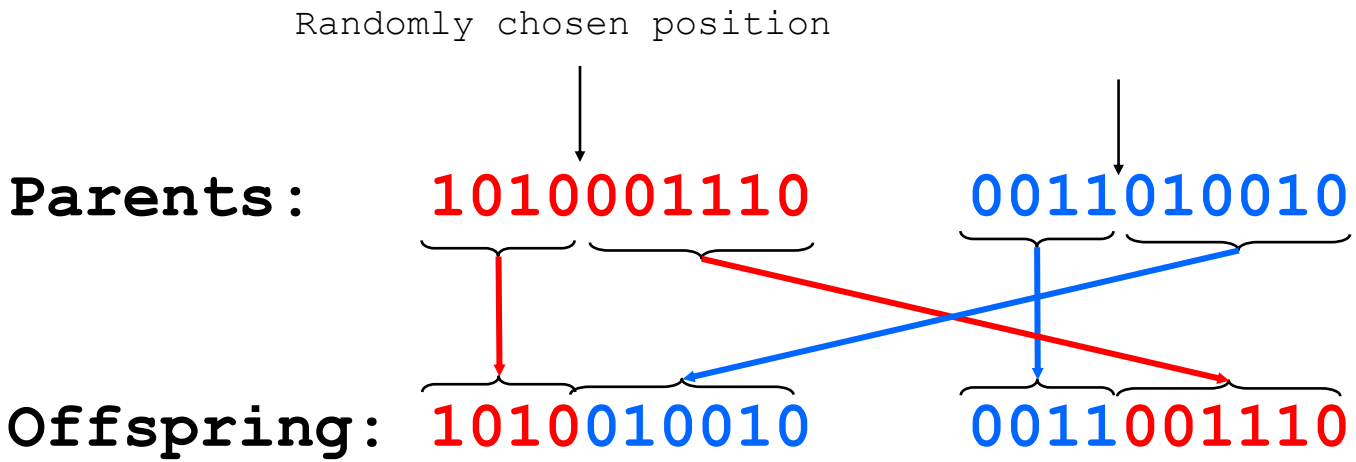
- Crossover
  - Two parents produce two offspring
  - There is a chance that the chromosomes of the two parents are copied unmodified as offspring
  - There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
  - Generally the chance of crossover is between 0.6 and 1.0
- Mutation
  - There is a chance that a gene of a child is changed randomly
  - Generally the chance of mutation is low (e.g. 0.001)

Reproduction Operators

- Crossover
  - Generating offspring from two selected parents
    - Single point crossover
    - Two point crossover (Multi point crossover)
    - Uniform crossover

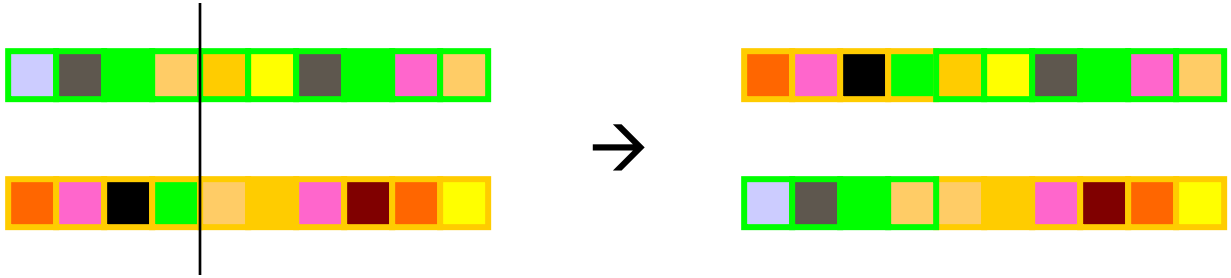
One-point crossover 1

- Randomly one position in the chromosomes is chosen
- Child 1 is head of chromosome of parent 1 with tail of chromosome of parent 2
- Child 2 is head of 2 with tail of 1



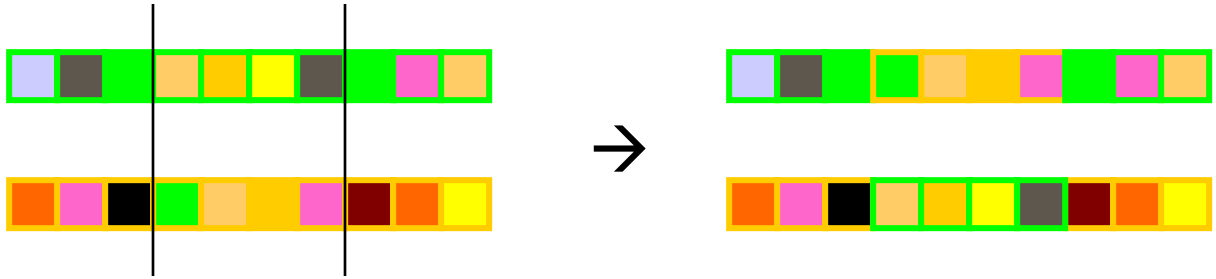
Reproduction Operators comparison

- Single point crossover

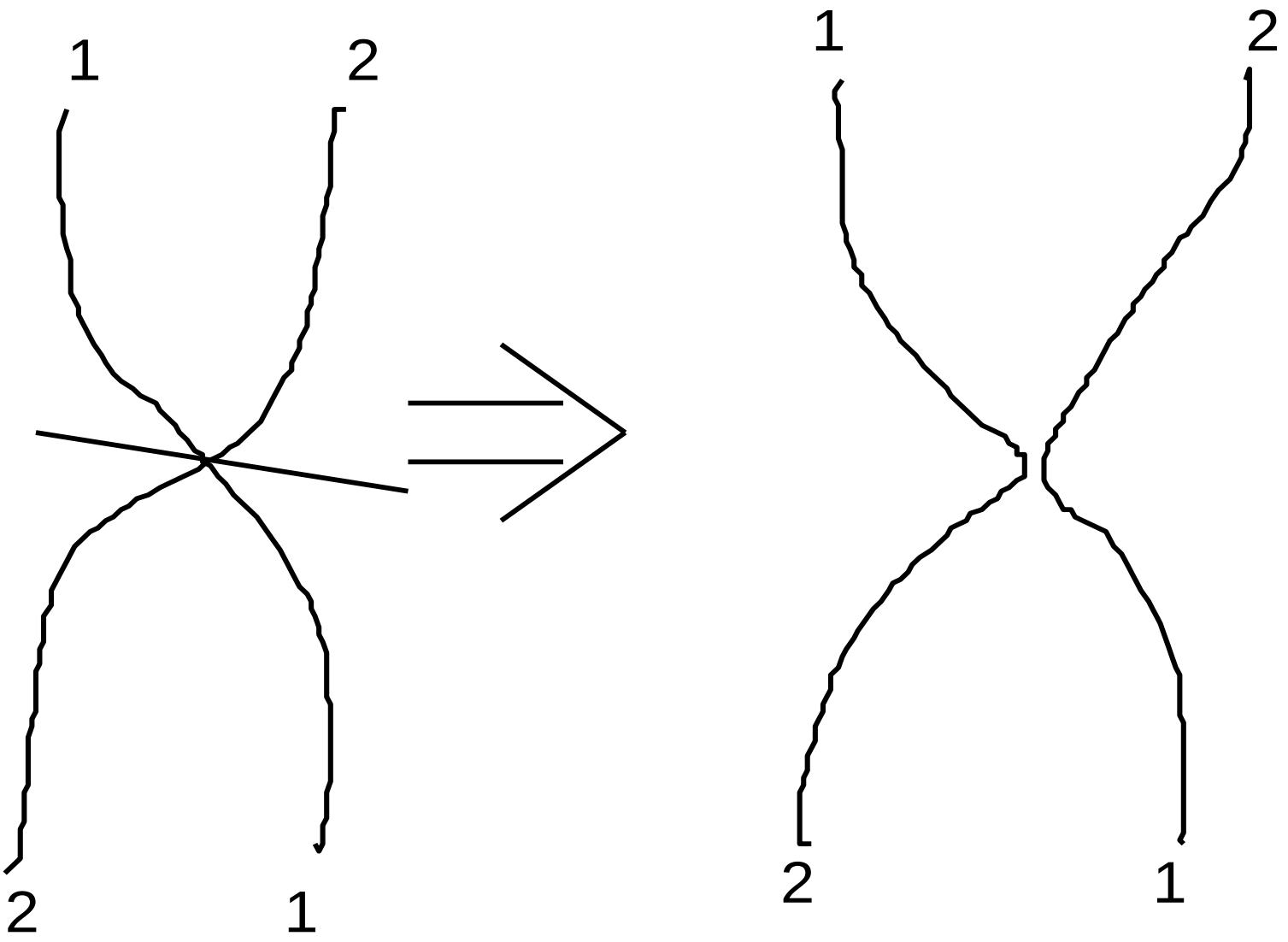


Cross point

- Two point crossover (Multi point crossover)

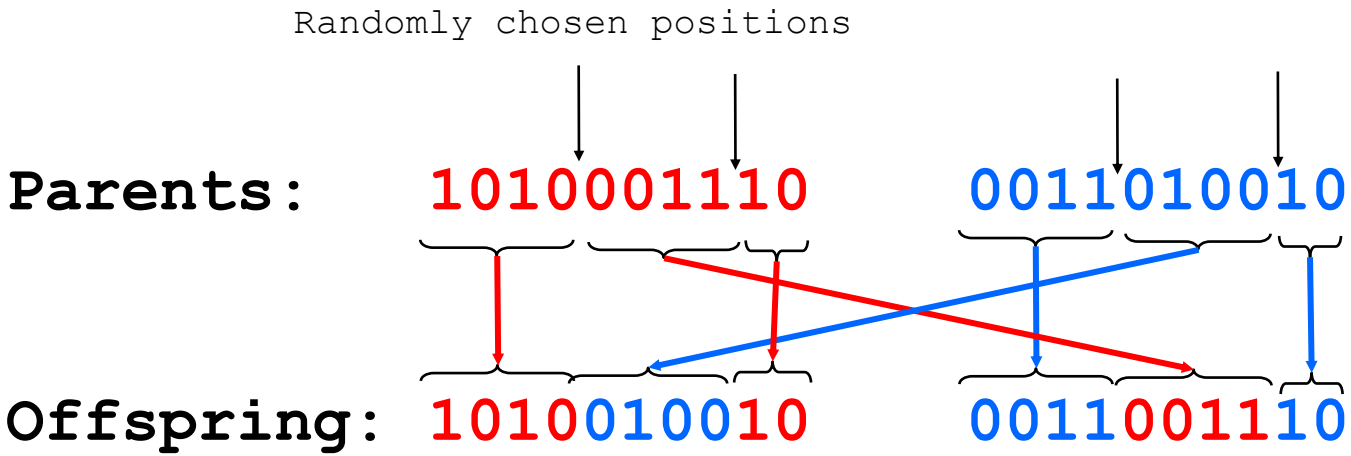


One-point crossover - Nature



Two-point crossover

- Randomly two positions in the chromosomes are chosen
- Avoids that genes at the head and genes at the tail of a chromosome are always split when recombined



Uniform crossover

- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent
- Bit density in mask determines how much material is taken from the other parent (takeover parameter)

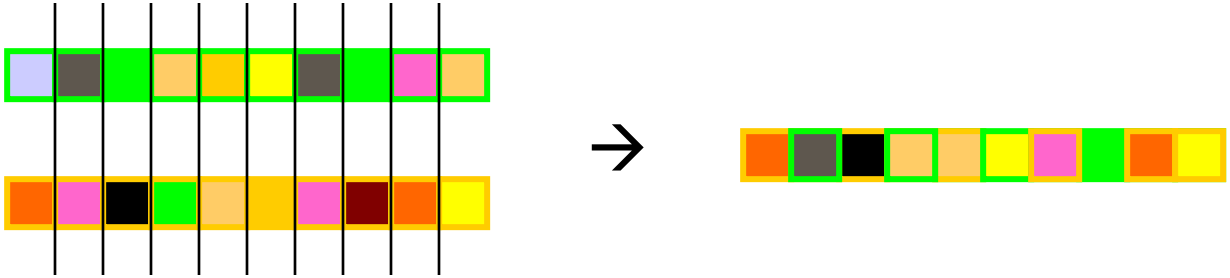
```
Mask:        0110011000      (Randomly generated)

Parents:     1010001110      0011010010

Offspring:   0011001010      1010010110
```

Reproduction Operators

- Uniform crossover



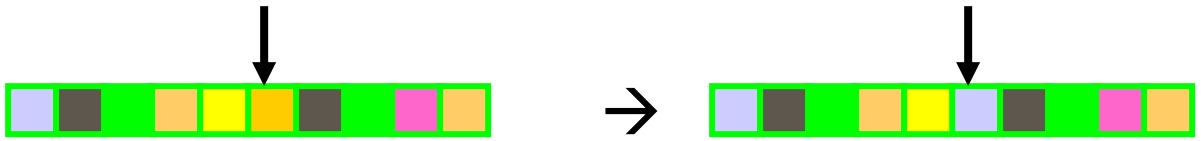
  - Is uniform crossover better than single crossover point?
    - Trade off between
      - Exploration: introduction of new combination of features
      - Exploitation: keep the good features in the existing solution

Problems with crossover

- Depending on coding, simple crossovers can have high chance to produce illegal offspring
  - E.g. in TSP with simple binary or path coding, most offspring will be illegal because not all cities will be in the offspring and some cities will be there more than once
- Uniform crossover can often be modified to avoid this problem
  - E.g. in TSP with simple path coding:
    - Where mask is 1, copy cities from one parent
    - Where mask is 0, choose the remaining cities in the order of the other parent

Reproduction Operators

- Mutation
  - Generating new offspring from single parent



  - Maintaining the diversity of the individuals
    - Crossover can only explore the combinations of the current gene pool
    - Mutation can "generate" new genes

Reproduction Operators

- Control parameters: **population size, crossover/mutation probability**
  - Problem specific
  - Increase population size
    - Increase diversity and computation time for each generation
  - Increase crossover probability
    - Increase the opportunity for recombination but also disruption of good combination
  - Increase mutation probability
    - Closer to randomly search
    - Help to introduce new gene or reintroduce the lost gene
- Varies the population
  - Usually using crossover operators to recombine the genes to generate the new population, then using mutation operators on the new population

Parent/Survivor Selection

- Strategies
  - Survivor selection
    - Always keep the best one
    - Elitist: deletion of the K worst
    - Probability selection : inverse to their fitness
    - Etc.

Parent/Survivor Selection

- Too strong fitness selection bias can lead to sub-optimal solution
- Too little fitness bias selection results in unfocused and meandering search

Parent/Survivor Selection

- Strategies
  - Parent selection
    - Uniform randomly selection
    - Probability selection : proportional to their fitness
    - Tournament selection (Multiple Objectives)

      Build a small comparison set

      Randomly select a pair with the higher rank one beats the lower one

      Non-dominated one beat the dominated one

      Niche count: the number of points in the population within certain distance, higher the niche count, lower the rank.
    - Etc.

Parent selection

Chance to be selected as parent proportional to fitness

- Roulette wheel

To avoid problems with fitness function

- Tournament

Not a very important parameter

Tournament Selection

- choose k (the tournament size) individuals from the population at random
- choose the best individual from the tournament with probability p
- choose the second best individual with probability p*(1-p)
- choose the third best individual with probability p*((1-p)^2)
- and so on

Tournament

- Binary tournament
  - Two individuals are randomly chosen; the fitter of the two is selected as a parent
- Probabilistic binary tournament
  - Two individuals are randomly chosen; with a chance p, $0.5<p<1$, the fitter of the two is selected as a parent
- Larger tournaments
  - n individuals are randomly chosen; the fittest one is selected as a parent

- By changing n and/or p, the GA can be adjusted dynamically

Roulette wheel

- Sum the fitness of all chromosomes, call it T
- Generate a random number N between 1 and T
- Return chromosome whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness

Chromosome:	1	2	3	4	5	6
Fitness:	8	2	17	7	4	11
Running total:	8	10	27	34	38	49
N (1 ≤ N ≤ 49):			23			
Selected:			3			

Problems with fitness range

- Premature convergence
  - ∆Fitness too large
  - Relatively superfit individuals dominate population
  - Population converges to a local maximum
  - Too much exploitation; too few exploration
- Slow finishing
  - ∆Fitness too small
  - No selection pressure
  - After many generations, average fitness has converged, but no global maximum is found; not sufficient difference between best and average fitness
  - Too few exploitation; too much exploration

Solutions for these problems

- Use tournament selection
  - Implicit fitness remapping
- Adjust fitness function for roulette wheel
  - Explicit fitness remapping
    - Fitness scaling
    - Fitness ranking

Fitness Function

Purpose

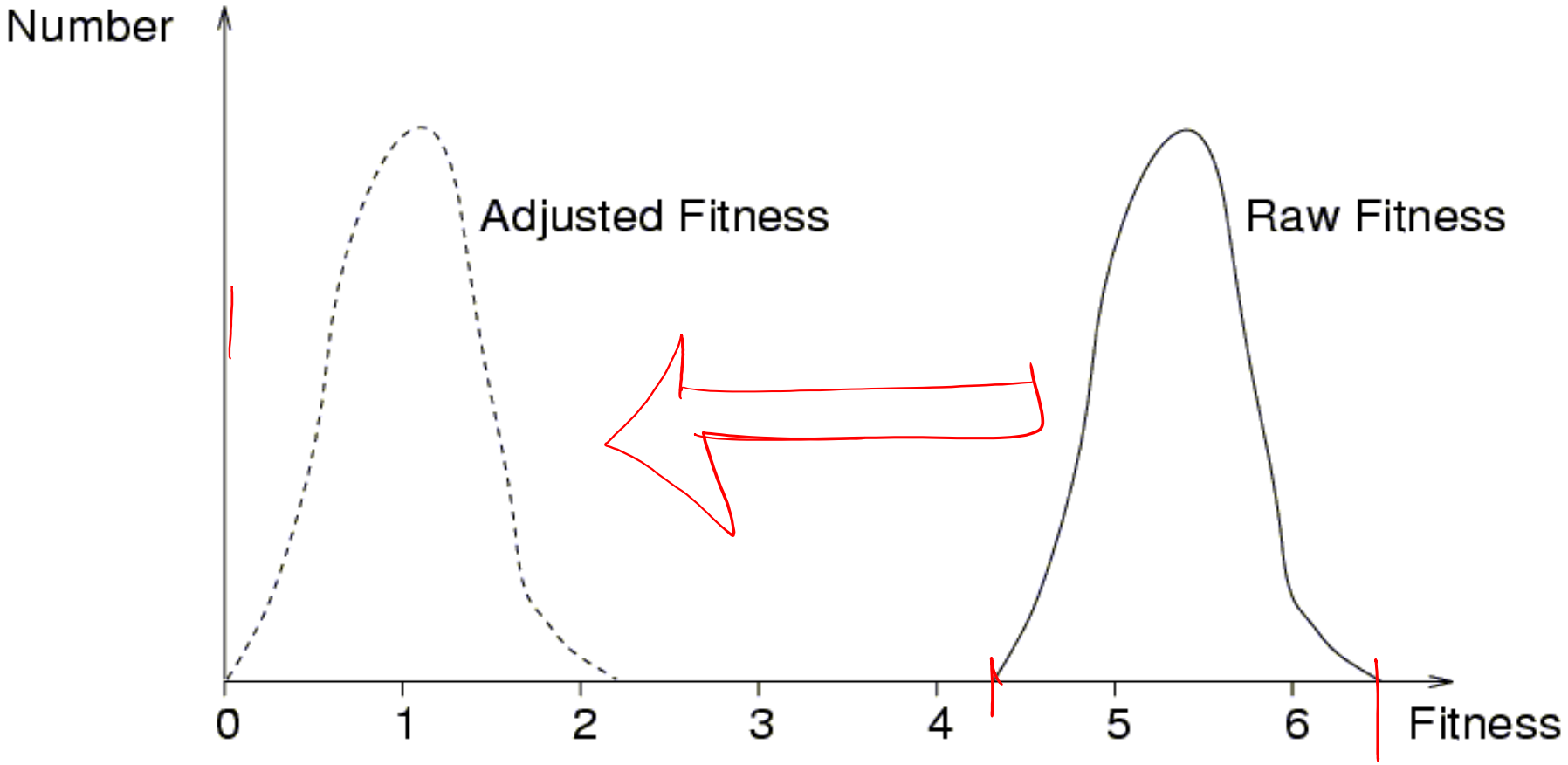
- Parent selection
- Measure for convergence
- For Steady state: Selection of individuals to die

- Should reflect the value of the chromosome in some "real" way
- Next to coding the most critical part of a GA

Fitness scaling

- Fitness values are scaled by subtraction and division so that worst value is close to 0 and the best value is close to a certain value, typically 2
  - Chance for the most fit individual is 2 times the average
  - Chance for the least fit individual is close to 0
- $\overline{x_i} = 2 * \frac{x_i - \min(x)}{\max(x) - \min(x)}$
- Problems when the original maximum is very extreme (super-fit) or when the original minimum is very extreme (super-unfit)
  - Can be solved by defining a minimum and/or a maximum value for the fitness

Example of Fitness Scaling



Fitness ranking

- Individuals are numbered in order of increasing fitness
- The rank in this order is the adjusted fitness
- Starting number and increment can be chosen in several ways and influence the results
- No problems with super-fit or super-unfit
- Often superior to scaling and windowing

Fitness Evaluation

- A key component in GA
- Time/quality trade off
- Multi-criterion fitness

Multi-objective/ multi criterion fitness

- 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:

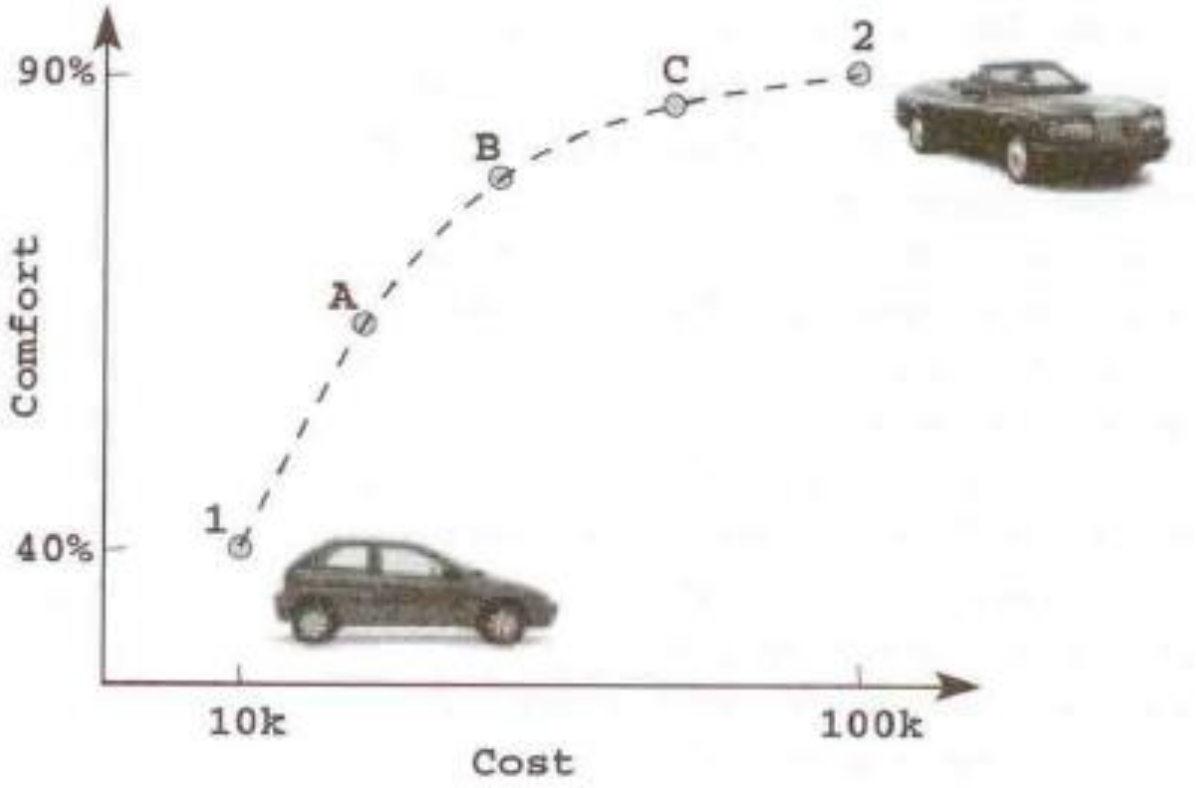
Mathematically

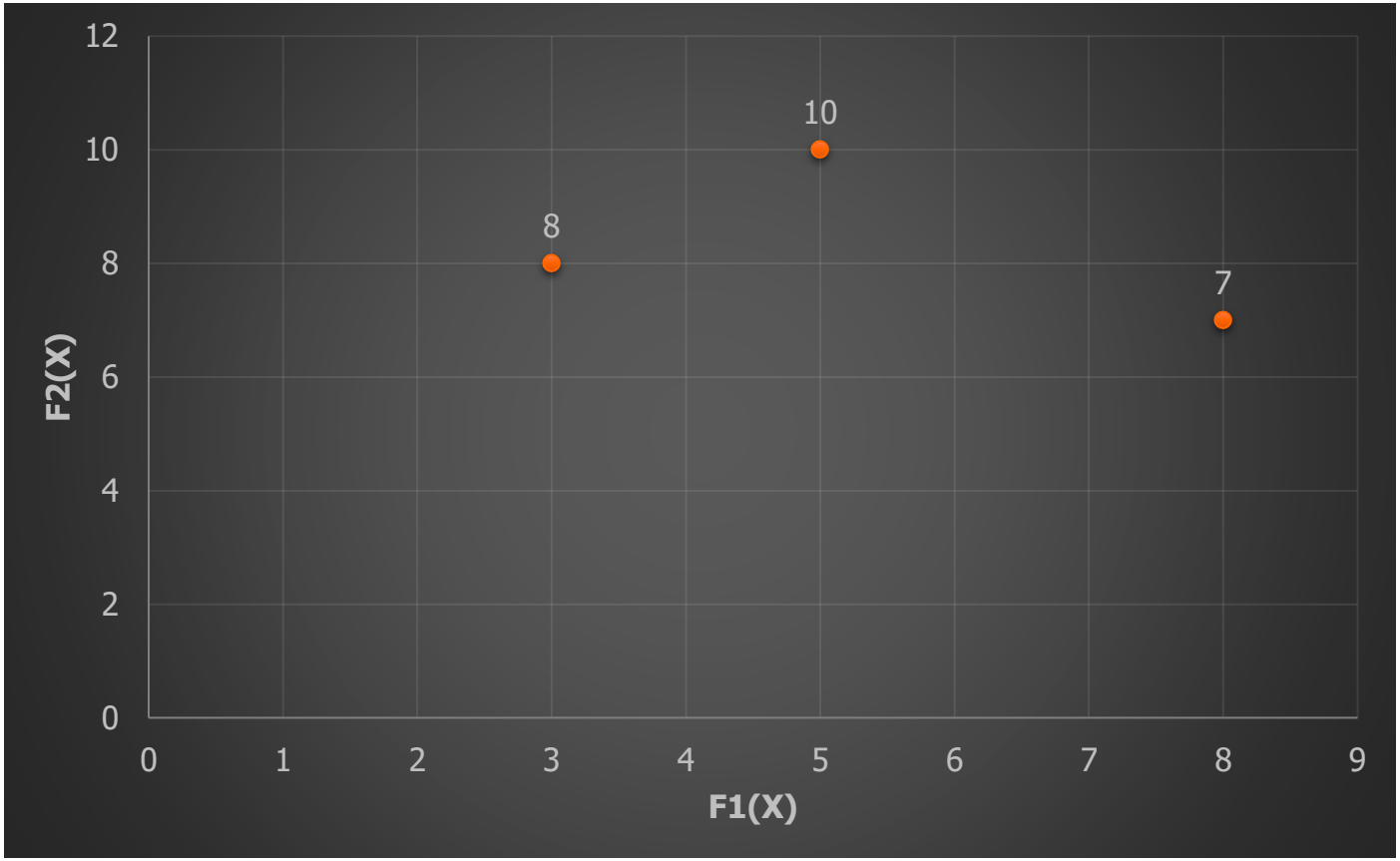
$$\begin{aligned} &\min/\max \quad f_m(\boldsymbol{x}), && m = 1, 2, \cdots, M \\ &\text{subject to } g_j(\boldsymbol{x}) \geq 0, && j = 1, 2, \cdots, J \\ &\qquad\qquad h_k(\boldsymbol{x}) = 0, && k = 1, 2, \cdots, K \\ &\qquad\qquad x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i = 1, 2, \cdots, n \end{aligned}$$

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

Example

- Buying a car: minimum cost, maximum comfort (Multi-objective)





Goal: Minimize f1(x)
Maximize f2(x)
Now, determine the ranking.

Dominance

- In the single-objective optimization problem, the superiority of a solution over other solutions is easily determined by comparing their objective function values
- Since multi-objective optimization problem deals with multiple objective functions, it is not easy to determine which solution/ set of solutions is best.
  - In multi-objective optimization problem, the goodness of a solution is determined by the *dominance*.

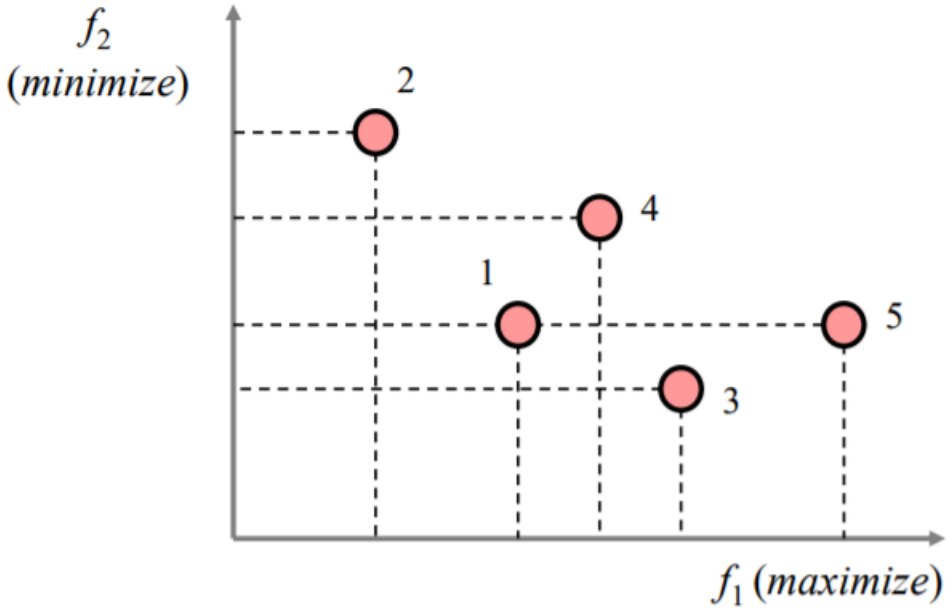
Test of Dominance

- Dominance Test:

  Solution 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 $x_2 \Leftrightarrow x_2$ is dominated by x_1
- Non-dominated solutions :
  - *If two solutions are compared, then the solutions are said to be non-dominated with respect to each other IF neither solution dominates the other*

Example Dominance Test

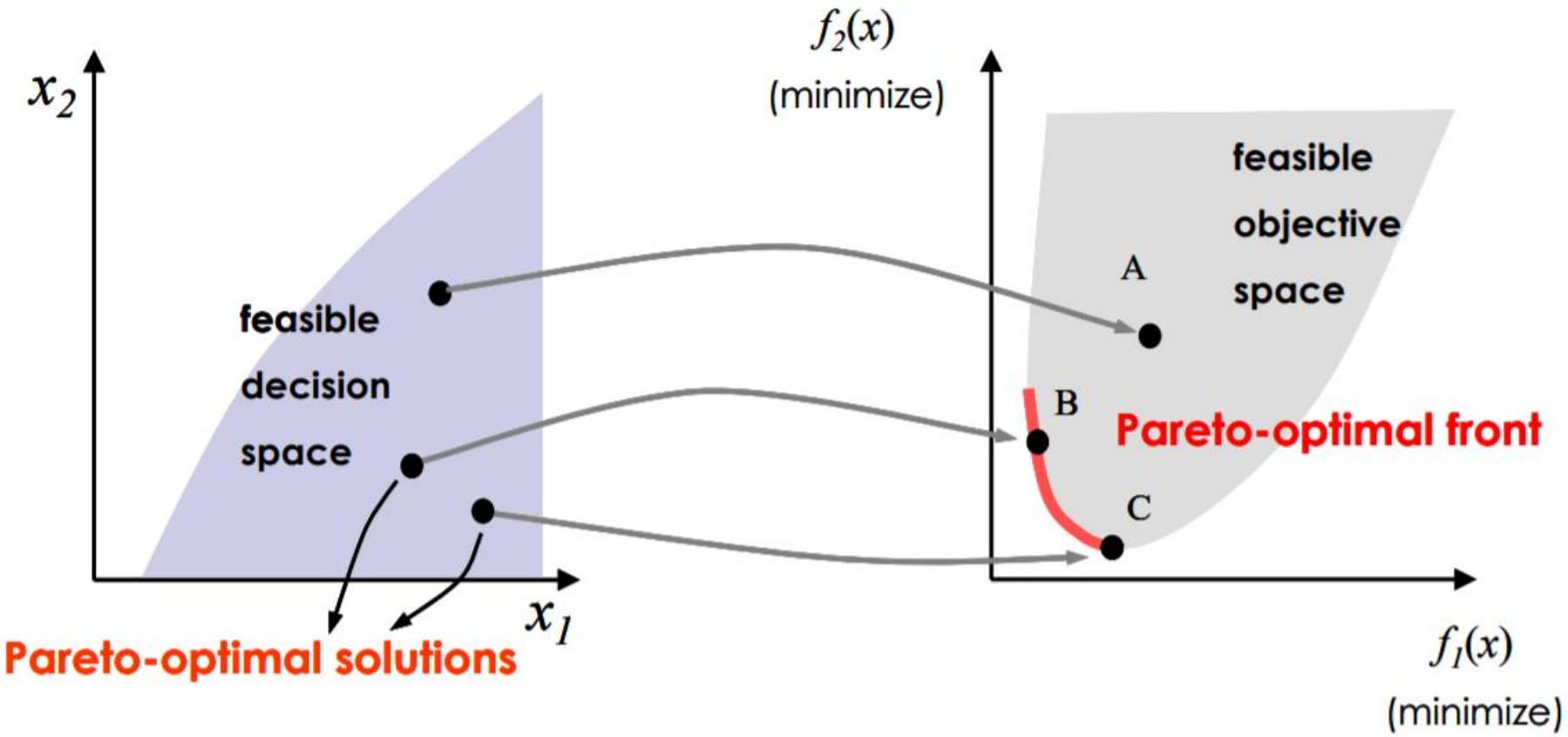


- 1 Vs 2: 1 dominates 2
- 1 Vs 5: 5 dominates 1
- 1 Vs 4: Neither solution dominates

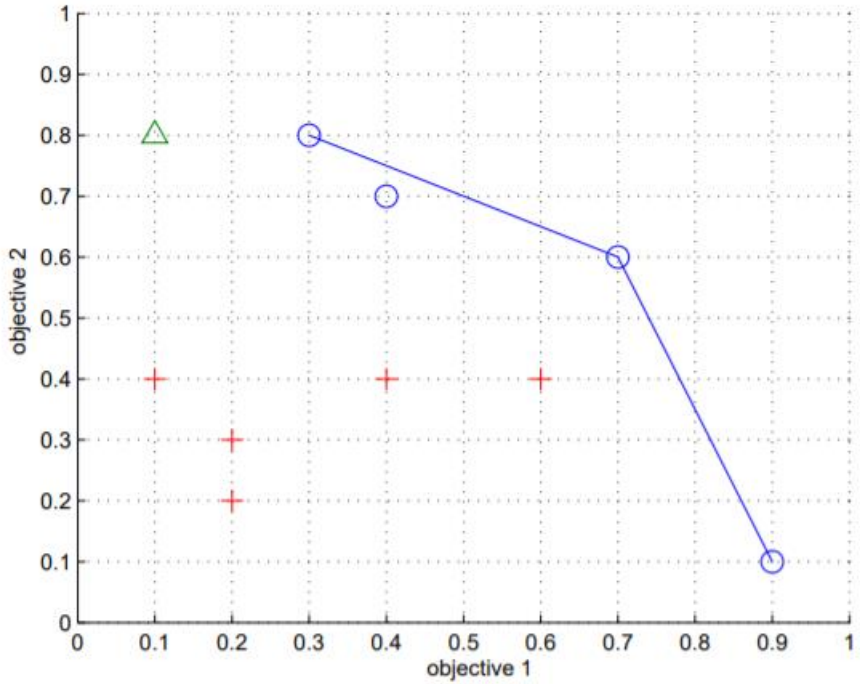
Pareto Optimal Solution

- Non-dominated solution set
  - 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



Graphical Depiction of Pareto Optimal Solution 2



Note: here the plot shows a maximization problem (aiming for higher objectives). Circles are pareto optimal, triangles are weakly pareto optimal.

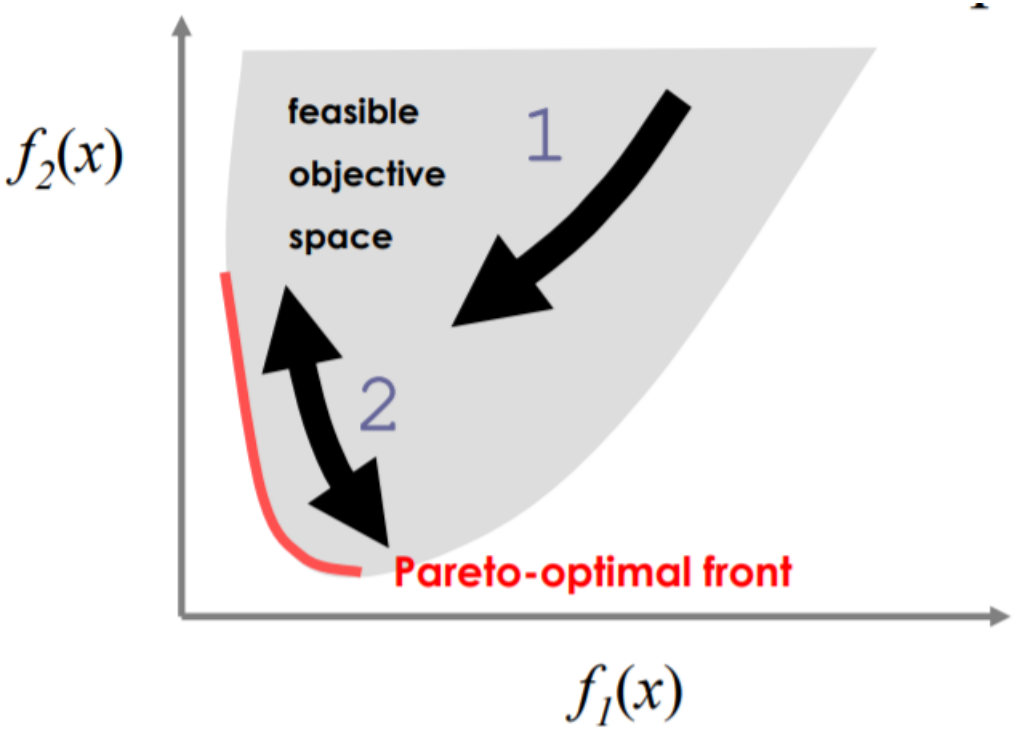
Graphical Depiction of Pareto Optimal Solution 3

Hotel	Stars	Distance from Beach	Price
A	**	0.7	1,175
B	*	1.2	1,237
C	*	0.2	750
D	***	0.2	2,250
E	***	0.5	2,550
F	**	0.5	980

We prefer hotels that are fancier, closer to the beach, and cheaper. Hotels A, B, E can be eliminated (not pareto optimal). Note that Hotel F is not the best in any objective, yet it is pareto optimal.[1]

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



How to obtain Pareto-optimal front?

Time complexity:
$O(mN^2)$

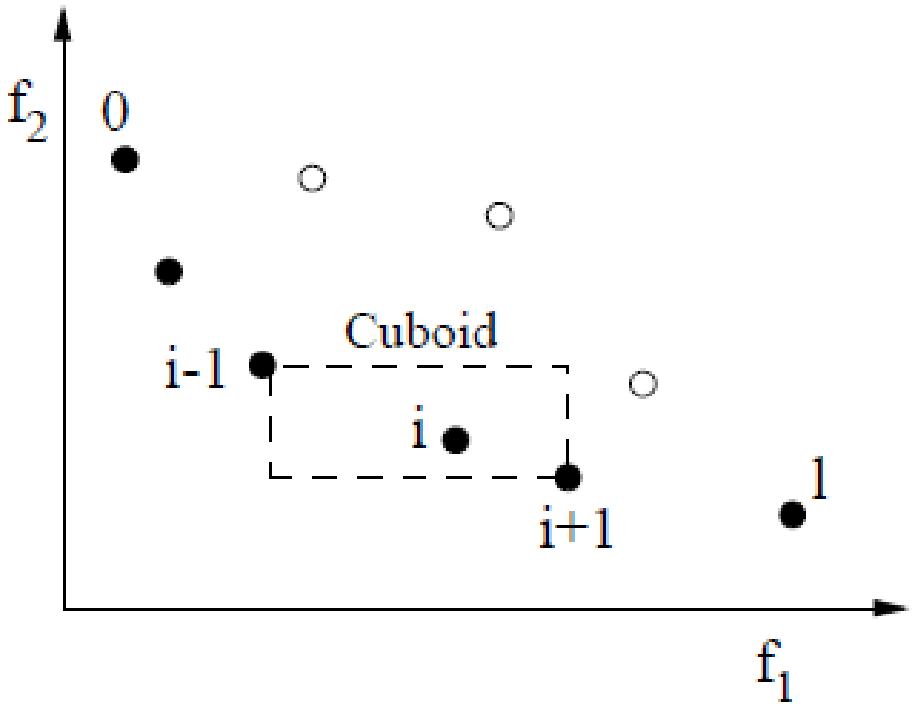
```
fast-nondominated-sort(P)
for each p ∈ P
   for each q ∈ P
      if (p ≺ q) then                    if p dominates q then
         S_p = S_p ∪ {q}                 include q in S_p
      else if (q ≺ p) then               if p is dominated by q then
         n_p = n_p + 1                   increment 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
```

Crowding distance

- To get an estimate of the density of solutions surrounding a particular point in the population, we take the average distance of the *two points on either side of this point along each of the objectives*.
- The quantity $i_{distance}$ serves as an estimate of the size of the largest cuboid enclosing the point i without including any other point in the population (we call this the *crowding distance*).

Crowding distance

- In following figure, the crowding distance of the i th solution in its front (marked with solid circles) is the average side-length of the cuboid (shown with a dashed box).



Crowding distance Algorithm

`crowding-distance-assignment` $(\mathcal{I})$

$l = |\mathcal{I}|$ — number of solutions in $\mathcal{I}$

for each i, set $\mathcal{I}[i]_{distance} = 0$ — initialize distance

for each objective m

  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$ — sort using each objective value

  $\mathcal{I}[1]_{distance} = \mathcal{I}[l]_{distance} = \infty$ — so that boundary points are always selected

  for $i = 2$ to $(l - 1)$ — for all other points

    $\mathcal{I}[i]_{distance} = \mathcal{I}[i]_{distance} + (\mathcal{I}[i + 1].m - \mathcal{I}[i - 1].m)$

Here $\mathcal{I}[i].m$ refers to the m-th objective function value of the i-th individual in the set $\mathcal{I}$. The complexity of this procedure is governed by the sorting algorithm. In the worst case (when all solutions are in one front), the sorting requires $O(mN \log N)$ computations.

Description of the Crowding distance Algorithm

- The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude.
- Thereafter, for each objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value.
- All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions.
- This calculation is continued with other objective functions.
- The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective.
- Each objective function is normalized before calculating the crowding distance.

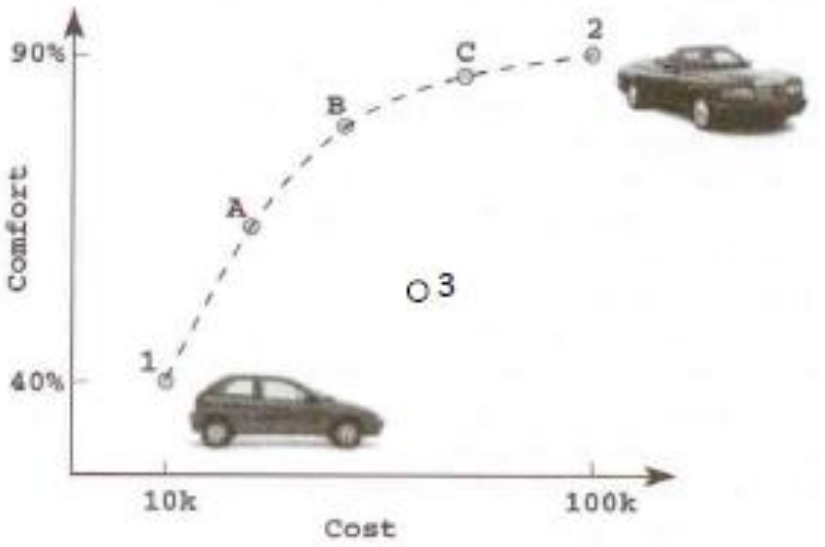
Crowded Comparison Operator

- The crowded comparison operator $\geq_n$ guides the selection process at the various stages of the algorithm towards a uniformly spread out Pareto-optimal front.
- Let us assume that every individual i in the population has two attributes.
  - Non-domination rank (i_{rank})
  - Local crowding distance ($i_{distance}$)
- We now define a partial order $\geq_n$ as
- $i \geq_n j$ if ($i_{rank} < j_{rank}$) or (($i_{rank} = j_{rank}$) and ($i_{distance} > j_{distance}$))
- Between two solutions with differing non-domination ranks we prefer the point with the lower rank. Otherwise, if both the points belong to the same front then we prefer the point which is located in a region with lesser number of points (the size of the cuboid inclosing it is larger)

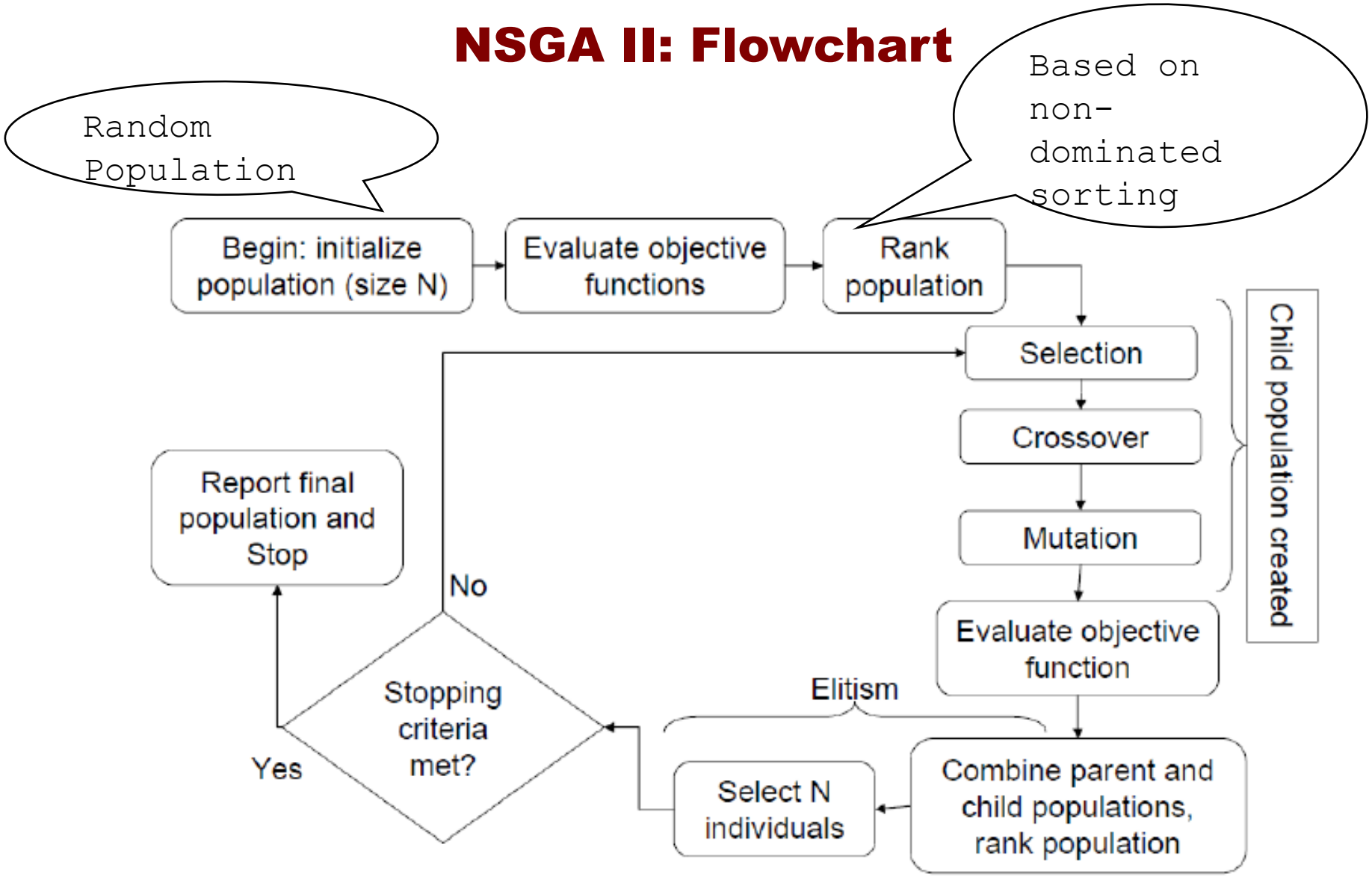
NSGA II

- Non-dominated Sorting Genetic Algorithm (NSGA)-II performs better than other constrained multi-objective optimizers* (PAEA, SPEA)
  - Better and faster convergence to true optimal front
  - Better spread on Pareto optimal front
- NSGA-II ranks designs based on non-domination
- For example : min-max problem
- Design 3 is dominated by both design A and B (and thus undesirable), but design A and B are non-dominated with respect to one another (and thus Pareto optimal).

Design	*Cost*	*Comfort*
A	25K	65%
B	45K	80%
3	55K	50%

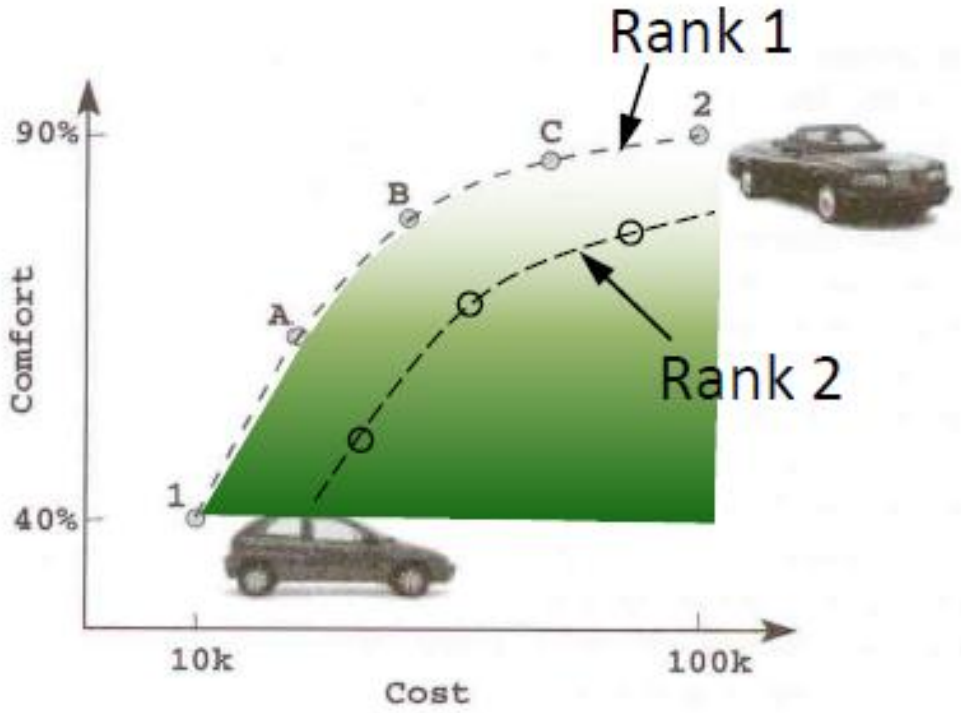


NSGA II: Flowchart



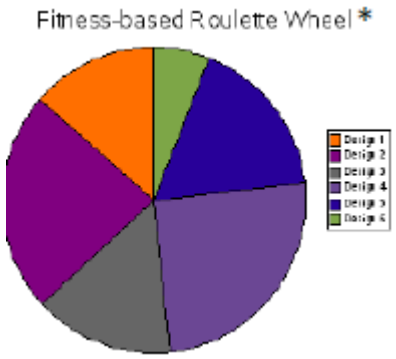
NSGA II: Details

- Ranks designs based on non-domination
  - The Pareto front is all rank 1 designs
  - If the rank 1 designs are removed, the next Pareto front will be all rank 2 designs, etc.



NSGA II: Selection and Fitness

- More fit designs have higher chance of passing their genes to the next generation
- Fitness is based on rank, low rank designs have higher fitness
- Selection :
  - Using rank based roulette wheel
  - Create roulette wheel with ns segments
  - Create random number between 0 and 1
  - Find segment on roulette wheel that contains the random number
  - Segment number corresponds to design number
  - Build parent database

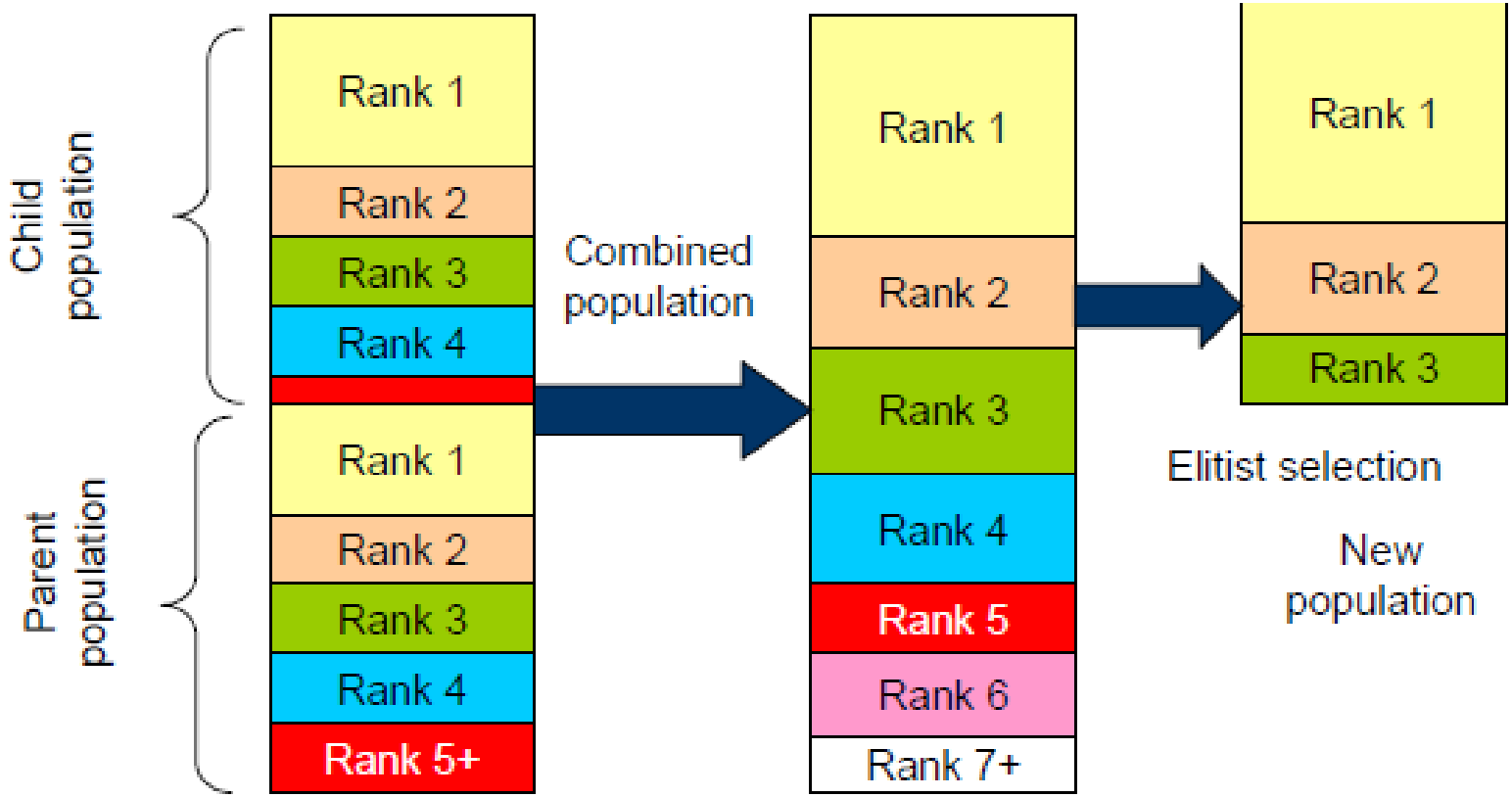


NSGA II: Child Population Creation

- Select two parents for each reproduction –randomly from parent database
- Crossover :
  - Probability close to 1
  - One point crossover –randomly select crossover point
  - Child = [parent1(1:cross_pt),parent2(cross_pt+1:N_var)]
- Mutation :
  - Exploration parameter
  - Probability of mutation is typically small (e.g. 0.2)
  - Randomly select gene to mutate
  - Randomly modify gene

NSGA II: Elitism

- Keeps best individuals
  - Combine the child and parent population
  - Select best individuals from the combined population



Reference

- Deb K, Pratap A, Agarwal S, Meyarivan TA. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation. 2002 Aug 7;6(2):182-97.