AI LAB

Artificial Intelligence LAB

Shay Bushinsky, Fall 2021

LAB4: Fighting local optima

Genetic
Algorithm Objectives

In this lecture

- Mutation Control
- Similarity Metrics
- Niching
- Crowding (deterministic & probabilistic)
- (Threshold & Clustering) Speciation
- Exaptation
- Multi Objective Optimization (Weighted & NSGA 2)

Our Aspiration List…

1. Optimality (find global optima)
2. Completeness (find all solutions)
3. Fast convergence
4. Reasonable running time

Fighting Local optima

Local optima signals

- Signals:
  1. Gene similarity
  2. Generation fitness average and standard deviation

Local optima signals

- Diversity is important since crossing over a homogeneous population does not yield new solutions
- The remedy is problem dependent

Combatting early convergence

1. Mutation control
2. Selection techniques
3. Punish Similarity
4. Diversify
5. Multi objective

Technique 1: mutation control

Mutation Control Methods

1. **The basic mutation operator**:
   - Easy implementation but hard to control result

2. **Non-uniform mutation**:
   - Mutation probability is greater early in the evolution, with the evolution advancing, mutation probability is appropriately reduced

Mutation Control Methods

3. **Adaptive Mutation:**
    - an individual with high fitness corresponds to a smaller mutation probability, and individuals with low fitness corresponds to a higher mutation probability

- This method can effectively protect the excellent individuals, but it can easily get trapped in local optima

Mutation Control Methods

- **4. Triggered Hyper Mutation:**

  - Increase the probability of mutation when the solution quality drops in order to encourage diversity

Technique 2: Selection Strategies – was covered earlier in course

Technique 3: NICHING

Niching

- **Niching** methods extend genetic algorithms to domains that require the location and maintenance of multiple solutions:

1. Classification and machine learning
2. Multimodal function optimization
3. Multi-objective function optimization
4. Simulation of complex and adaptive systems

Niching & Crowding

- Prevent the convergence of the whole population to just one of the peaks by **penalizing** individuals that are “close” to other individuals in the population

- Two techniques are used:
  - 1. Fitness Sharing
  - 2. Crowding

1. Sharing Method

- Population is first divided into niches
- Shared fitness of any individual is computed only with respect to the individuals that are in its niche
- **Sharing Radius** defines the niche size
- Individuals within this radius will be regarded as similar and thus need to share fitness

Fitness Sharing

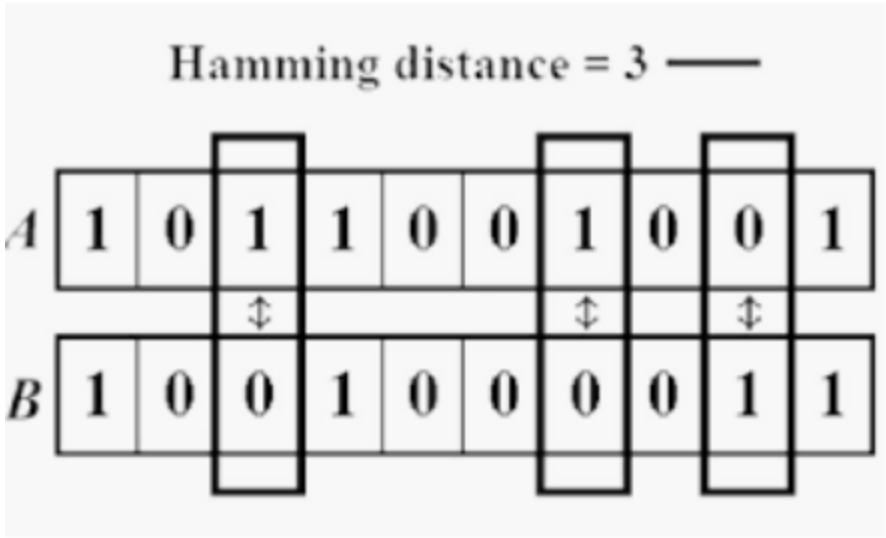
- Transforms the **raw fitness** of an individual into the **shared one** (usually lower)
- Motivation: there is only a limited and fixed number of “resources” (i.e., fitness value) available at each niche
- Individuals occupying the same niche will have to share the resources

Measuring similarity

Distance between instances

- The similarity between two individuals is defined by the distance between them – denoted d_{ij}.

- For example, the similarity between two binary strings can be defined by their Hamming distance.



Edit Distance- distance between strings

```
>>> s = "Mannhaton"
>>> s = s[:2] + s[3:]          # deletion
>>> s
'Manhaton'
>>> s = s[:5] + "t" + s[5:]    # insertion
>>> s
'Manhatton'
>>> s = s[:7] + "a" + s[8:]    # substitution
>>> s
'Manhattan'
```

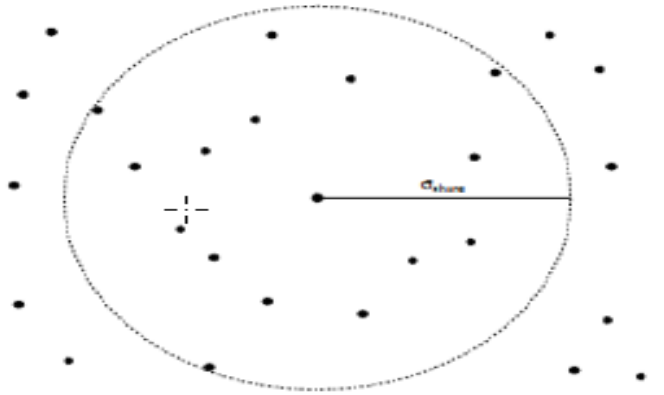
Distance between Permutations

- Example: **Kendall tau** distance between:
- 0 3 1 6 2 5 4 and 1 0 3 6 4 2 5 is: **4** because the pairs: 0-1, 3-1, 2-4, 5-4

  are in different order in the two - all other pairs are in the same order
- In order to calculate the Kendall tau distance, pair each value with every other value and count the number of times the values in list 1 are in the opposite order of the values in list 2

σ Sharing Radius parameter

- Sharing radius, σ-share, defines the niche size



Fitness is shared between all the individuals in the circle

Fitness sharing principles

1. The more similar two genes are, the lower their individual fitness will become…
2. The more it shares the more its fitness is scaled down

Sharing function and shared fitness

- Define a sharing function:

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha}, & \text{if } d_{ij} < \sigma_{share}, \\ 0, & \text{otherwise,} \end{cases}$$

- Based on it, the shared fitness is defined as:

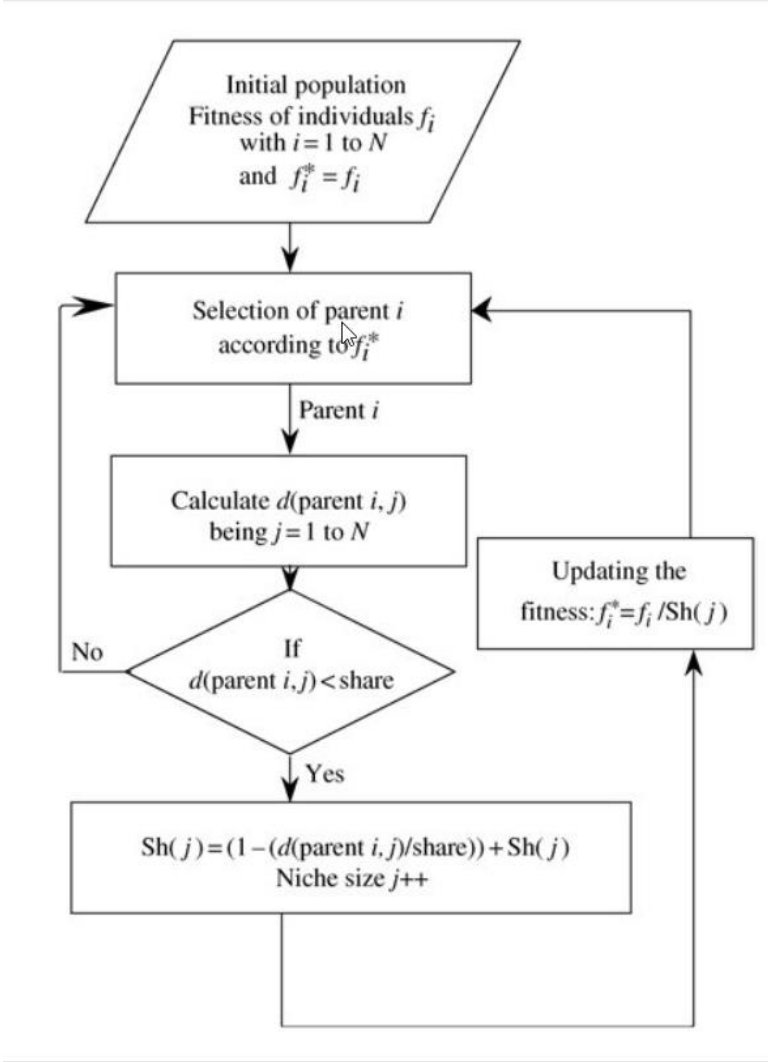
$$f_{share}(i) = \frac{f_{raw}(i)}{\sum_{j=1}^{\mu} sh(d_{ij})}$$

- μ denotes the population size
- α is a constant – usually equals 1
- σ niche radius is fixed by the user at some minimum distance between peaks
- The fitness of each member is scaled down based on its proximity to others in same niche

Notes

1. dij is a matrix of similarity on the entire population
2. If the sh function finds that dij is less than sigma-share, it returns a value in the range [0,1] that increases as dij decreases

NICHE algorithm flow



The Sharing effect

- Discourage convergence to a single region of the fitness function: The more individuals try to move in, the worse off they all are

- If the GA converges to a single local optimum somewhere, the fitness of that optimum decreases because of the increased competition within the niche

At local optima

- Eventually, another region of the fitness landscape becomes more attractive, and individuals migrate over there

- The idea is to reach a steady state -- a fixed point in the dynamics -- where an appropriate representation of each niche is maintained

Niching Effect

- Reduce the effect of genetic drift resulting from the selection operator in the simple genetic algorithms

- They maintain population diversity and permit genetic algorithms to explore more search space to identify multiple peaks, whether optimal or otherwise

Critique

- Sharing is hard because of the need to manually set the niche radius, and the algorithm is quite sensitive to this choice

- It is also computation consuming

Crowding

Crowding in Nature

- Similar individuals in natural population, often of the same species, compete against each other for limited resources

- Dissimilar individuals tend to occupy different niches, they typically don’t compete

Crowding

- New members of a species replace older members of that species, not replacing members of other species
- Crowding doesn’t increase the diversity of population, rather it strives to **maintain the pre-existing diversity**
- It’s not directly influenced by fitness value

Crowding

- Crowding was first introduced by De Jong as a technique for preserving population diversity and preventing premature convergence

- Offspring were replaced according to their improved fitness

- In Deterministic Crowding not only the quality of potential solutions is important, but also their proximity to their parents

Standard vs. Deterministic

- Standard crowding:
    - In this method, only a fraction of the global population specified by a percentage G (generation gap) reproduces and dies in each generation

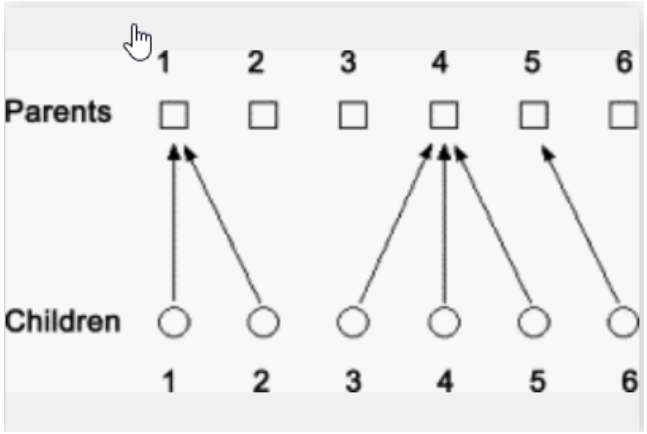
Crowding Phases

- In the pairing phase, offspring individuals are **paired** with individuals in the current population according to a **similarity metric**

- In the replacement phase, a decision is made for each pair of individuals as to which of them will remain in the population

Deterministic Crowding

- Instead of dealing with an explicit radius, we work by limiting the set of individuals that a new offspring can replace to some set of similar solutions
- For example, an offspring might be allowed to replace only one of its parents.

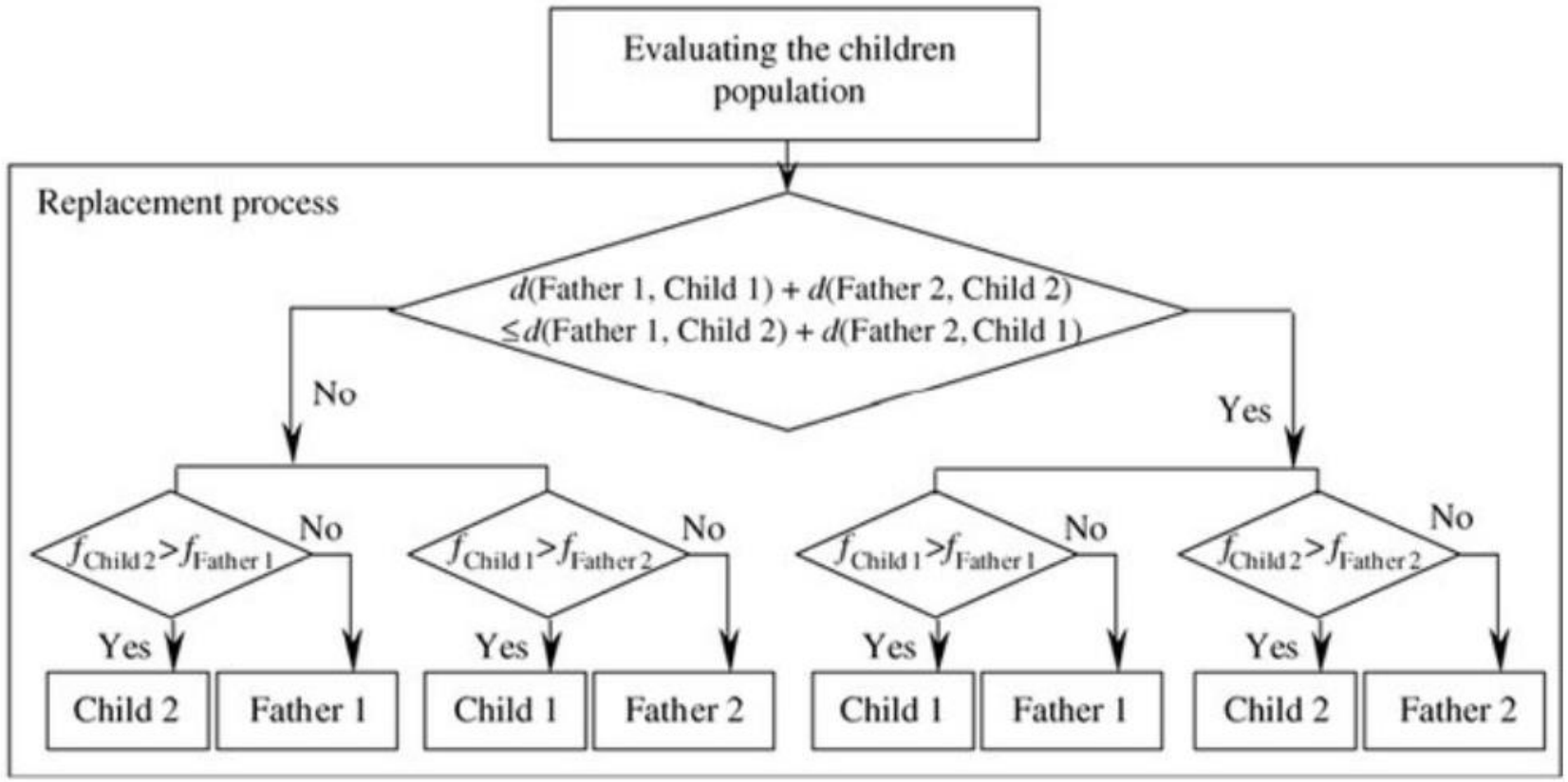


Deterministic Crowding Algorithm

- Deterministic crowding matches up parents and children by minimizing a distance measure over all parent-child combinations

- it uses the deterministic acceptance rule of always picking the best fit individual in each parent and child

Deterministic Crowding Flow



Crowding Effect

- The effect is to try to prevent replacing a unique individual with one that is very similar to a dozen others in the population and thus to preserve diversity

Strengths of Deterministic Crowding Algorithm

1. Requires no additional parameters
2. Simple and Fast

Critique

- No restoration process: Species of higher fitness tend to win over species of lower fitness thus niches may get lost in course of evolution

Probabilistic crowding

General Description

- In probabilistic crowding, offspring compete (against their most similar parent and the survivor is chosen with a probability proportional to their fitness

- Originally De Jong used CF=2 : two parents

Probabilistic Crowding

- Primarily a distance-based niching method
- Main difference is the use of a probabilistic rather than deterministic acceptance function:
- No longer do stronger individuals always win over weaker individuals, they win proportionally according to their fitness, thus we get the restorative pressure

Algorithm in a nutshell

- 2 parents 2 children evolution assumed:

1. Select/Sample 2 parents uniformly
2. Create 2 children (mutation etc.)
3. Both children are paired with parents according to overall best parent-child similarity
4. Each child competes with its paired parent in a binary tournament
5. Replacement is decided according to probability proportional to their fitness

Offspring generation options

1. M Variant: mutation only
2. M&C: Mutation and Crossover
3. C: Crossover only

Probabilistic Crowding

- An offspring replaces the most similar individual taken from a randomly drawn subpopulation of size CF (crowding factor) from the global population

Probabilistic Crowding

- Two core ideas of probabilistic crowding are:
- 1. To hold tournament between similar individuals
- 2. To let tournaments be probabilistic

Crowding Factor Sampling

- Random sample of *CF* (Crowding Factor) individuals, is sampled from the population

- Larger crowding factor indicates less tolerance for the similar solutions, smaller values indicate similar solutions are more welcome

Binary tournament winner

- Two similar individuals are sampled to compete and replace one of these two individuals in the next generation

- The probability of x winning is:

$$p_{\mathbf{x}} = p(\mathbf{x}) = \frac{f(\mathbf{x})}{f(\mathbf{x}) + f(\mathbf{y})},$$

The Role of Similarity

- Similarity comes about implicitly, when mutation only is employed or
- Explicitly, by using distance measure in connection with crossover or
- Explicit search for family members

Speciation

Speciation

- Speciation is focused on converging to the actual peaks

- The *speciation* heuristic **penalizes** crossover between candidate solutions that are too similar;
-
- This encourages population diversity and helps prevent premature convergence to a less optimal solution

Threshold Speciation

- Relies on similarity measurements between two genes
- The **species count** is the desired number of species (common default=30)
- The **species threshold** specifies the minimum similarity that genes must have to be the same species

Threshold control

- Example:
- Genome 1: [2.0, 3.0, 5.0]
- Genome 2: [1.0, 2.0, 1.0]
- Euclidian Distance = 4.242641

- If the **Euclidean distance** is bellow the speciation threshold => these two genomes would be in the same species
- The threshold is **adjusted** to maintain the parameter species count
- Increase / decrease at the end of generation

Variant: Clustering Speciation

- Threshold speciation increases the effectiveness of crossover which can be a destructive operator

- Use k-means or k-medoids to cluster the population – no need for the threshold parameter

Random immigrants

Random Immigrants

- A fraction of the current population is replaced by **randomly** generated individuals in each generation of the run

- A replacement strategy, like replacing **random** or worst individuals of the population, defines which individuals are replaced by the **immigrants**

The island Model

- The **Island Model** assumes that the total population used by the **genetic algorithm** is broken down into subpopulations referred to as **islands**

- Each **island** is in effect executing a version of the **genetic algorithm** on each subpopulation

Immigration Policy

- No. of individuals undergoing migration
- Frequency of migrations
- Policy of selecting immigrants
- Immigrant replacement policy
- Topology of communication amongst islands
- Nature of island communication (synchronous / asynchronous)

Exaptation - preadaptation

Exaptation

- A process by which a structure that has been adapted by evolution for its performance at a certain function is to be rudimentarily useful at some new function

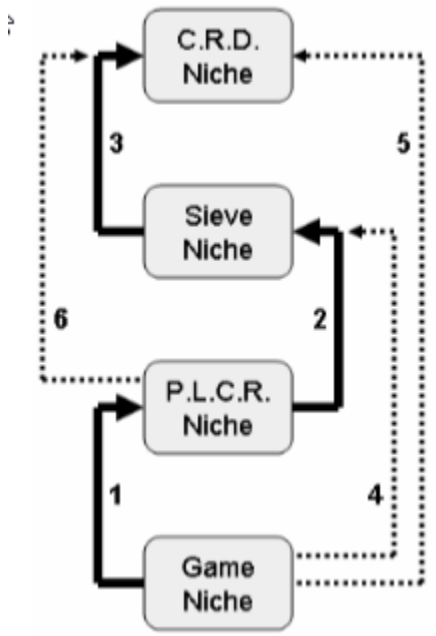
- Bird feathers are a classic example: initially they may have evolved for temperature regulation, but later were adapted for flight

Niche Migration

- Different niches are seeded –each with **different fitness function**

- Before next generation, check for potential migrant individuals to copy into another niche

Six migration paths



Exaptation via Migration

- An individual is chosen from source niche by size-two tournament selection
- Its fitness is evaluated by its niche fitness function
- No migration if it is not **viable** in the destination niche
- If viable, then migrated if better than least fit in the destination

MULTI Objective FUNCTIONS

Multi objective functions

- Multi-objective optimization problems involving two and three objectives
- Optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives.
- Ex: Minimizing cost while maximizing comfort while buying a car

Pareto Optimal

- For a nontrivial multi-objective optimization problem, there does not exist a single solution that simultaneously optimizes each objective

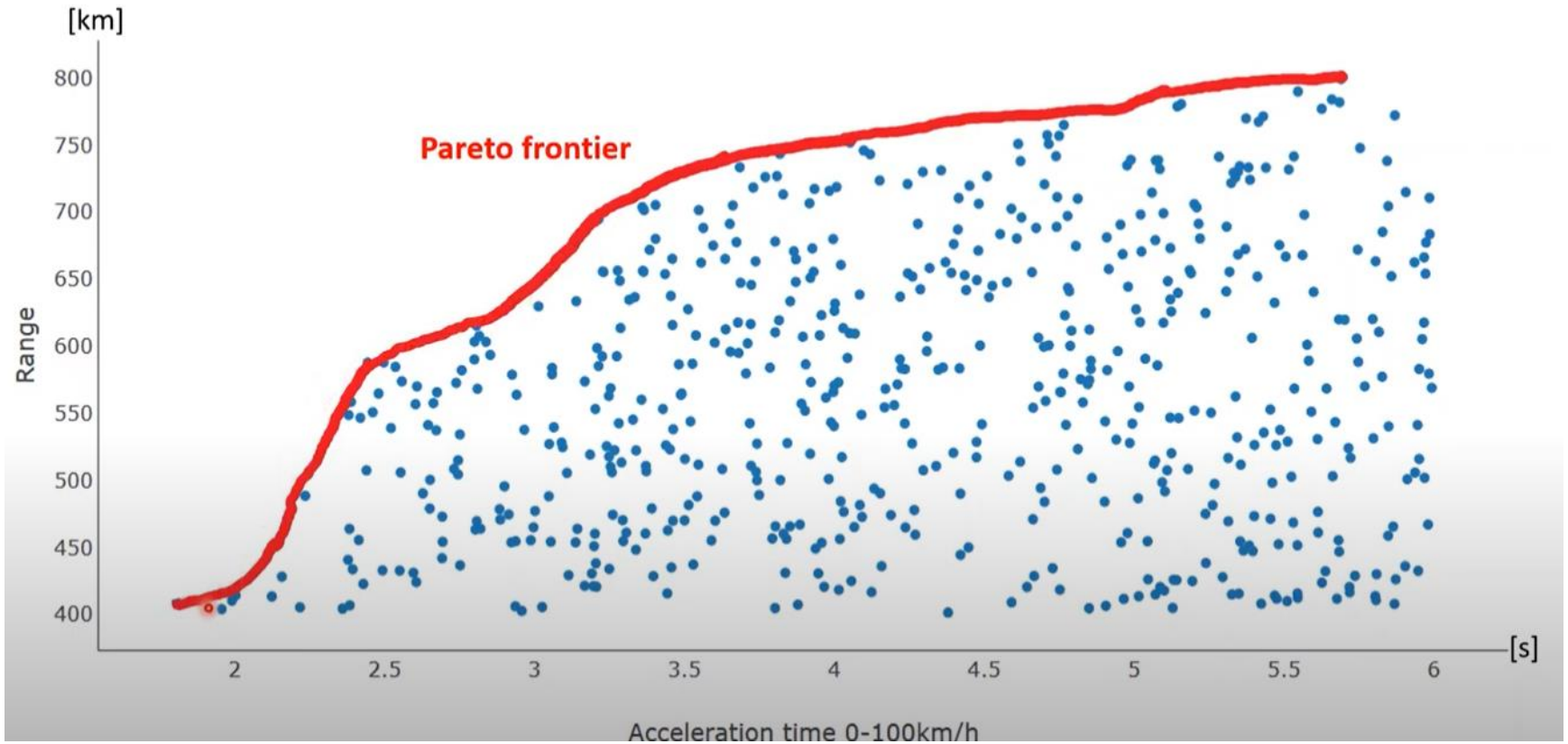
- In that case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of “Pareto optimal” solutions

Example: Electric car

- Conflicting objectives:
  1. Increase acceleration time
  2. reduces the range and vice versa

- A gene: [wheel size, engine power, battery capacity]
- There is no one perfect solution

Pareto Frontier: max range, min acceleration



Pareto Optimal Definition

- A solution is called Pareto Optimal, if none of the objective functions can be improved in value without degrading some of the other objective values

- Without additional subjective preference information, all Pareto optimal solutions are considered equally good (as vectors cannot be ordered completely)

Optimization Techniques

1. Weighted Sum

2. Probabilistic Approach

Problem Example

- Suppose one wants to minimize both of the following functions simultaneously:

- $F1 = 750x1+60(25-x1)\ x2+45(25-\ x1)(25-\ x2)$
- $F2 = (25-\ x1)\ x2$

Weighted Sum Approach

Convert a multiple objective problem into a single objective problem:

1. Assign a weight to each function such that:

   $w1 + w2 = 1$ and $w1 , w2 \geq 0$

1. Solve: $FT = \sum wi\ Fi = w1\ F1 + w2\ F2$

   But this will only provide us with a single Pareto point…

   Finding all points: Alter the weights and solve again

Weighted Sum approach typical Problems

1. Inability to generate points in non-convex portions of the frontier
2. Inability to generate a uniform sampling of the frontier
3. A non-intuitive relationship between combinatorial parameters (weights, etc.) and performances
4. Poor efficiency (can require an excessive number of function evaluations)

NSGA-II

- A Fast and Elitist Multi-objective Genetic Algorithm
- Deb, K., Pratap, A., and Agarwal, S.. A

- **N**on-dominated **S**orting **G**enetic **A**lgorithm
- For multi-objective function optimization

Method

- Successive sampling of the search space

- The best individuals are calculated by non-dominated sorting breaking ties using the crowding distance

The Population

- Population of parameters:

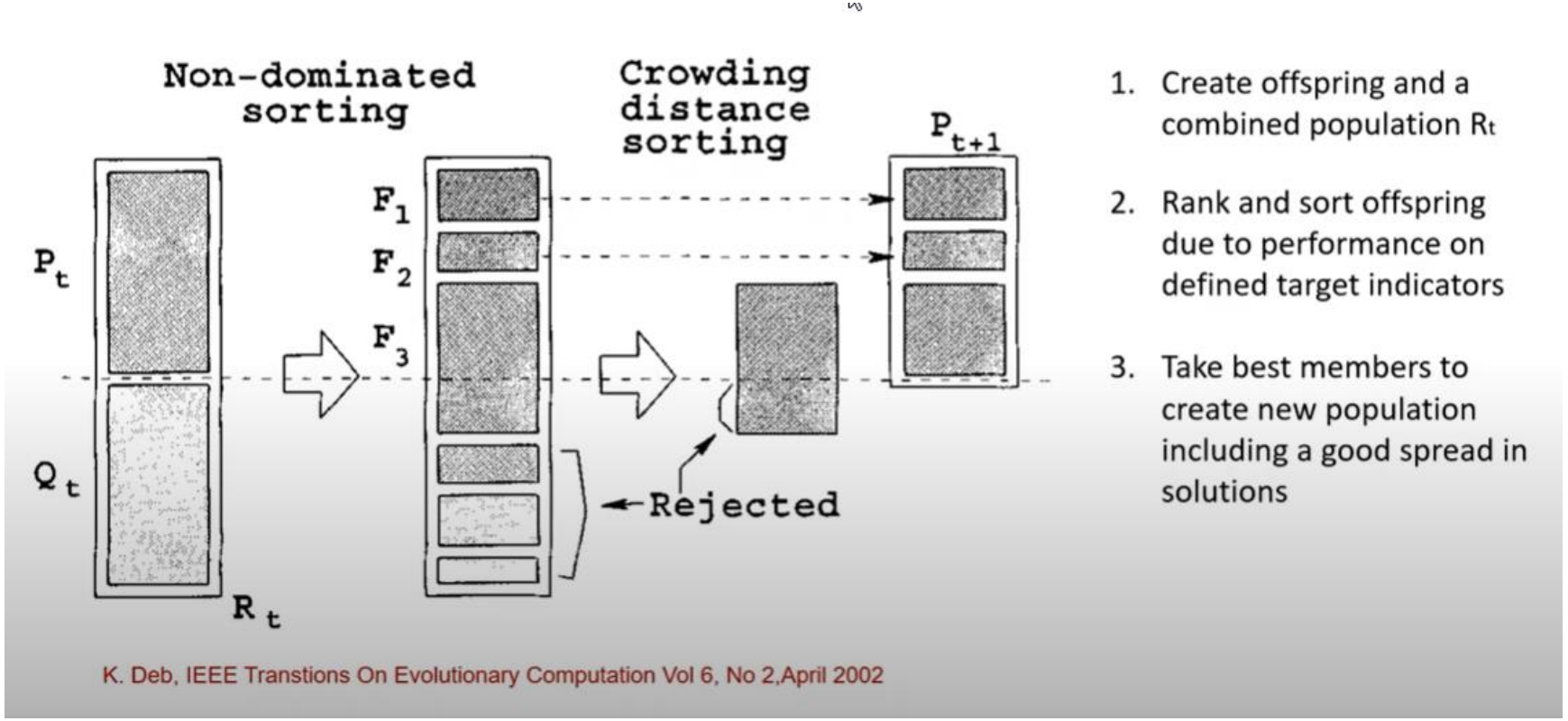
- **rt**: the new population
  is composed out of two populations:

1. **pt** the previous population
2. **qt** the offspring's of pt

Population Ranking

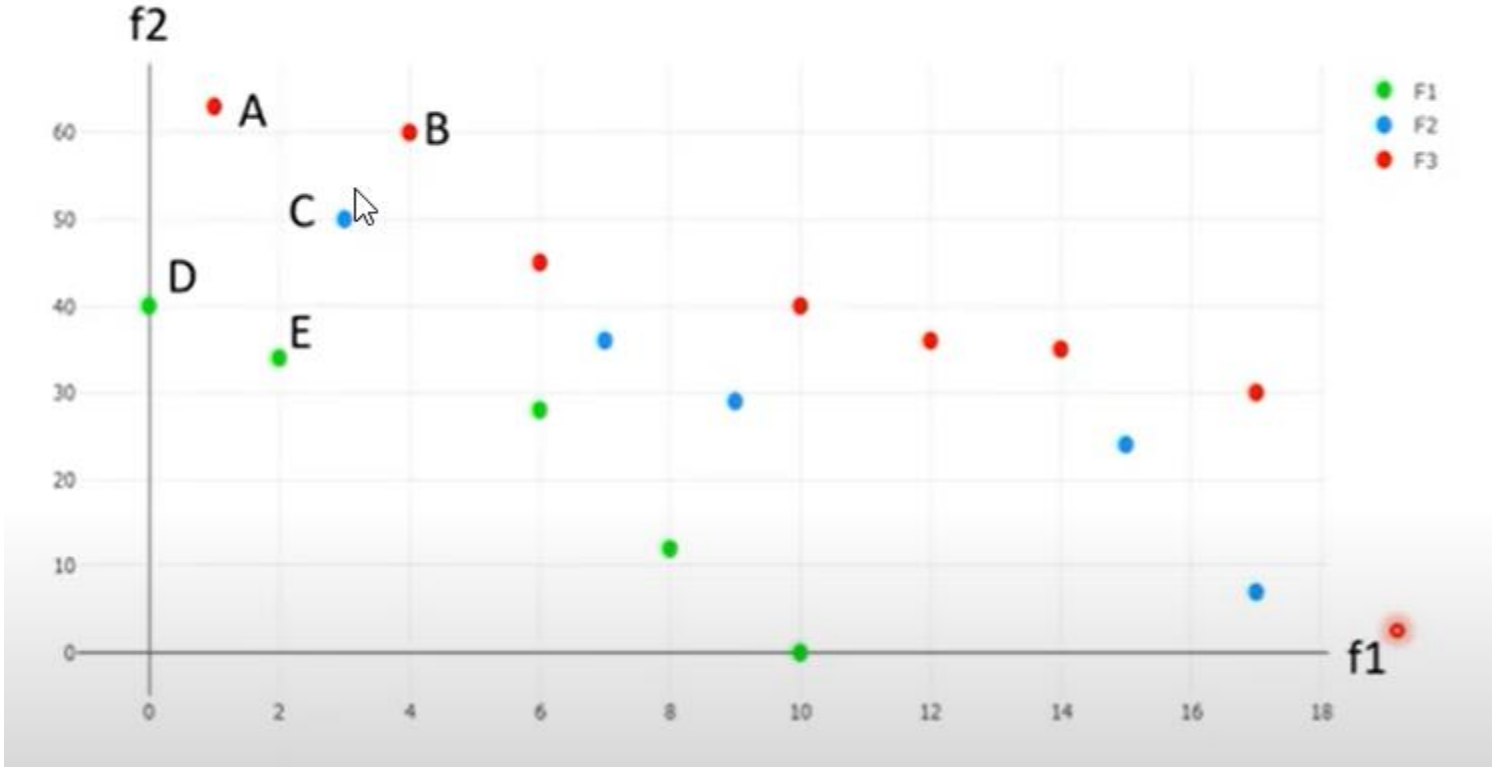
- Ranking on different fronts:
- every individual is ranked according to the objectives (target indicators)
- Several fronts are maintained: f1, f2 ,f3...
- The smaller the front index - the better the front is in terms of its gene performance
- Fronts are added until the entire population is filled
- (the last front may not be inserted at its entirety)

NSGA-II Main-Loop



1. Create offspring and a combined population R_t
2. Rank and sort offspring due to performance on defined target indicators
3. Take best members to create new population including a good spread in solutions

Non-Dominated Sorting



Non-dominated Sorting

- Assign each gene to its correct front

- Def: An individual **dominates** another individual iff it is better than the other in at least one indicator and not worse on all the other indicator(s)

Domination Example

- Assume the objective is to minimize indicator1 and indicator2 (two kpis):

- g1 = (x1,y1) , g2=(x2,y2)
- g1 dominates g2 iff:
- (x1<=x2 & y1<=y2) & (x1<x2 | y1 < y2)

non-dominated sorting algorithm

- Each gene ("current") is compared with all the others for domination
- Two elements are generated:

1. a **list** of all individuals **dominated by** the current individual
2. a **counter** (domination_count) set to how many individuals **dominate** the current individual

Generating Fronts

a) The first front (f1) will include all individuals with domination_count =0 meaning none of f1 members have an individual that dominates them
b) Scan all members of f1 and subtract 1 from all its list members domination_count

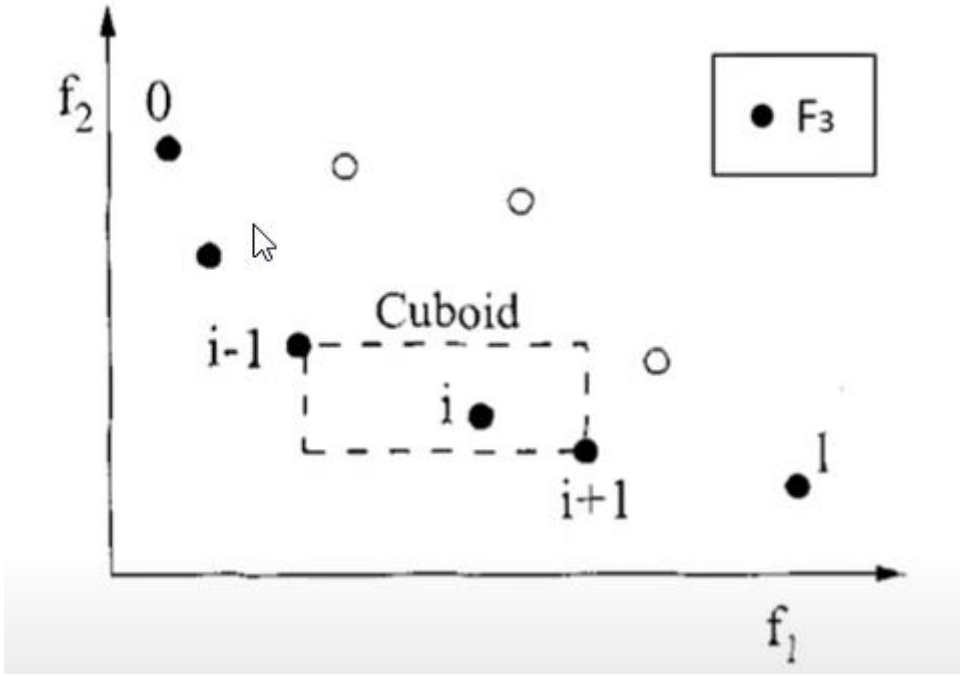
Reiterate steps a and b to create fronts f2, f3 etc.

process B: prepare to create offsprings:

- Use **crowding-distance** for the selection-strategy to avoid local optima:

- Individuals with higher crowding distance are picked first
- The total crowding distance for each gene is the sum of its crowding-distances per target-indicator (objective)

Crowding Distance Sorting



For o in objectives:

$$sort(front, o)$$
$$distance(front(\min)) = \inf$$
$$distance(front(\max)) = \inf$$
$$distance(i) = distance(i) + \frac{o(i+1) - o(i-1)}{o(max) - o(min)}$$

Algorithm for computing the distance of individual i:

- for o in objectives:
- sort(front, o) # sort front per indicator o
- distance(front(min)) = inf
- distance(front(max)) = inf
- distance(i) = distance(i) + ((o(i+1) - o(i-1)) / (o(max) - o(min))
- this means that the delta of the point from its neighboring points is scaled according to the delta of the extreme points in the front

process C: create offspring

- Repeat:
- 1. Tournament Selection:
- select parent1: (niche criteria)
- Sample g1 and g2
- g1=(a1,b1) where a1 is his rank and b2 is his crowding distance
- g2=(a2,b2)
- e.g. if a1=a2 & b1 > b2 select g1 as parent for reproduction
- select parent2: (using the same process)
- 2. Crossover
- 3. Mutation

Initialization

- Initially a random parent population **P0** is created
- This population is sorted based on non-domination
- Each solution is assigned a fitness equal to its non-domination level (1 is the best level)
- Thus, minimization of fitness is assumed
- Binary tournament selection, recombination and mutation operators are used to create child population **Q0** of size N
- From now on:

Generation Iteration

- Rt = Pt U Qt # combine
  parent and children population (Rt will be of size 2N)

- F = fast-nondominated-sort(Rt)

  # F = (F1; F2; : : :), all non-dominated fronts of Rt

-

Generation Iteration cont.

- until |Pt+1| < N : # till the parent population is filled

- crowding-distance-assignment(Fi)

  # calculate crowding distance in Fi

- Pt+1 = Pt+1 U Fi
- # include i-th non-dominated front in the parent pop

Generation Iteration cont.

- Sort(Pt+1; >= n) # sort in descending order using >= n
- Pt+1 = Pt+1[0 : N] # choose the first N elements of Pt+1
- Qt+1 = make-new-pop(Pt+1) # use selection, crossover and mutation to create a new population Qt+1
- t = t + 1