CO3091 - Computational Intelligence and Software Engineering

Lecture 12

Image from: http://economictimes.indiatimes.com/photo/40769692.cms

# Multi-Objective Evolutionary Algorithms

Leandro L. Minku

# Overview

- Multi-objective optimisation problems.
- Single-objective algorithmic design for multi-objective problems.
- Non-dominated Sorting Genetic Algorithm II (NSGA-II).

# Multi-Objective Optimisation Problems

- Real world problems frequently have more than one objective.
- Software project scheduling problem:
  - Cost and duration (to be minimised).
- Some requirements selection formulations:
  - Cost (to be minimised) and score (to be maximised).

# Using Single-Objective Algorithms for Multi-Objective Problems

- In order to deal with multi-objective problems by using single-objective optimisation algorithms, the multiple objectives have to be combined into a single fitness function.

- E.g.: software project scheduling problem

$$\text{fitness}(\mathbf{x}) = w_{cost} * \text{cost}(\mathbf{x}) + w_{dur} * \text{duration}(\mathbf{x})$$

(to be minimised)

$w_{cost}$ and $w_{dur}$ $\in [0,1]$

$w_{cost} + w_{dur} = 1$

# Using Single-Objective Algorithms for Multi-Objective Problems

- Generic fitness function for *k* objectives:

$$\text{fitness}(x) = \sum_{i=1}^{k} w_i f_i(x)$$

Weighted average of objective functions $f_i(x)$.

$$w_i \in [0,1],\ 1 \leq i \leq k$$

$$\sum_{i=1}^{k} w_i = 1$$

# Advantage of Combining Objectives into a Single Fitness Function

- Advantage of combining multiple objectives into a single fitness function:
    - Once weights are set, any single-objective optimisation algorithm can be used.

# Disadvantages of Combining Objectives into a Single Fitness Function

- Disadvantage 1:
  - Finding suitable weights is not easy before knowing what different trade-offs among objectives may be available.
  - Example:

$$fitness(\mathbf{x}) = w_{cost} * cost(\mathbf{x}) + w_{dur} * duration(\mathbf{x})$$ (to be minimised)

$w_{cost} = 0.7$

$w_{dur} = 0.3$

Candidate solution 1:
cost = 15,000
duration = 12 months

Candidate solution 2:
cost = 15,500
duration = 24 months

Which of these solutions is better?

# Disadvantages of Combining Objectives into a Single Fitness Function

- Disadvantage 1:
  - Finding suitable weights is not easy before knowing what different trade-offs among objectives may be available.
  - Example:

$$\text{fitness}(\mathbf{x}) = w_{cost} * \text{cost}(\mathbf{x}) + w_{dur} * \text{duration}(\mathbf{x})$$ (to be minimised)

$w_{cost} = 0.7$

$w_{dur} = 0.3$

Candidate solution 1:
cost = 15,000
duration = 12 months

fitness = 10503.6

Candidate solution 2:
cost = 15,500
duration = 24 months

fitness = 10857.2

# Disadvantages of Combining Objectives into a Single Fitness Function

- Disadvantage 1:
  - Finding suitable weights is not easy before knowing what different trade-offs among objectives may be available.
  - Example:

fitness(**x**) = $w_{cost}$ * cost(**x**) + $w_{dur}$ * duration(**x**)    (to be minimised)

$w_{cost}$ = 0.7

$w_{dur}$ = 0.3

Candidate solution 1:
cost = 15,000
duration = 12 months

Candidate solution 2:
cost = 14,500
duration = 24 months

Which of these solutions is better?

# Disadvantages of Combining Objectives into a Single Fitness Function

- Disadvantage 1:
  - Finding suitable weights is not easy before knowing what different trade-offs among objectives may be available.
  - Example:

fitness(**x**) = $w_{cost}$ * cost(**x**) + $w_{dur}$ * duration(**x**)    (to be minimised)

$w_{cost}$ = 0.7

$w_{dur}$ = 0.3

Candidate solution 1:
cost = 15,000
duration = 12 months

fitness = 10503.6

Candidate solution 2:
cost = 14,500
duration = 24 months

fitness = 10157.2

# Disadvantages of Combining Objectives into a Single Fitness Function

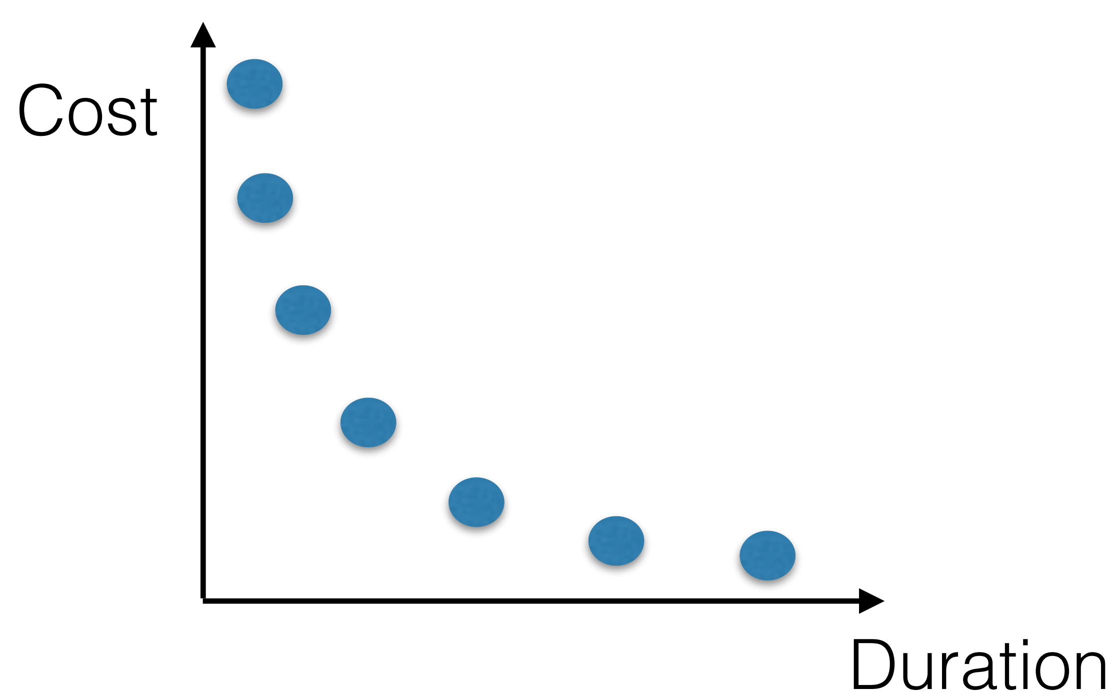
- Disadvantage 2:
  - If the different objectives have different scales, weights must reflect not only preferences towards certain objectives, but also treat the different scales.
  - E.g.: costs may typically be between 1000 and 80,000. Durations may typically vary between 3 and 48.

    If $w_{cost} = w_{dur} = 0.5$, costs will tend to have greater importance.
- Disadvantage 3:
  - The weights will strongly affect the results.
    - You may not get certain solutions that could be interesting, but do not look interesting based on the chosen weights.

# Multi-Objective Evolutionary Algorithms

- Consider different objectives separately, instead of combining them into a single fitness function.

- Advantages:
  - No need to set weights.
  - Having objectives in different scales is not a problem.
  - Retrieve a set of solutions with different trade-offs. E.g.:




[Demo]

# Multi-Objective Evolutionary Algorithms

- Consider different objectives separately, instead of combining them into a single fitness function.

- Advantages:
  - No need to set weights.
  - Having objectives in different scales is not a problem.
  - Retrieve a set of solutions with different trade-offs.

- Disadvantage:
  - It is difficult to visualise trade-offs among >3 objectives.
  - Some multi-objective evolutionary algorithms also struggle to produce good solutions when we have >3 objectives.

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

- One of the most famous multi-objective optimisation algorithms.
- Evolutionary algorithms put some selective pressure towards better individuals (parents and / or survival selection).
- The concept of what is a better individual is simple when there is a single objective.
  - Minimisation problem:

    $f(sol_A) = 5$,
    $f(sol_B) = 10$ —> $sol_A$ is better than $sol_B$

    $f(sol_A) = 5$,
    $f(sol_B) = 5$ —> $sol_A$ is equally good to $sol_B$

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

- When more than one objective is considered separately, this idea does not work well. E.g.:
    - Minimise objective f1 and objective f2.
      f1(solA) = 10, f2(solA) = 5
      f1(solB) = 4,   f2(solB) = 10
      Which solution is better?

- NSGA-II is based on the concept of dominance (and non-dominance) to determine which solutions are better.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months

Candidate solution B:
cost (min) = 16,000
duration (min) = 24 months

Solution A dominates solution B.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months

Candidate solution B:
cost (min) = 16,000
duration (min) = 24 months

Solution B does not dominate solution A.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months

Candidate solution B:
cost (min) = 14,500
duration (min) = 24 months

Solution A does not dominate solution B.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months

Candidate solution B:
cost (min) = 14,500
duration (min) = 24 months

Solution B does not dominate solution A.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

Candidate solution B:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 11

Solution A does not dominate solution B.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

Candidate solution B:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 11

Solution B dominates solution A.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

Candidate solution B:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

Solution A does not dominate solution B.

# Dominance

- A solution $sol_A$ dominates another solution $sol_B$ if the following conditions are satisfied:
  - $sol_A$ is equal or better than $sol_B$ in all objectives, and
  - $sol_A$ is strictly better than $sol_B$ in at least one objective.

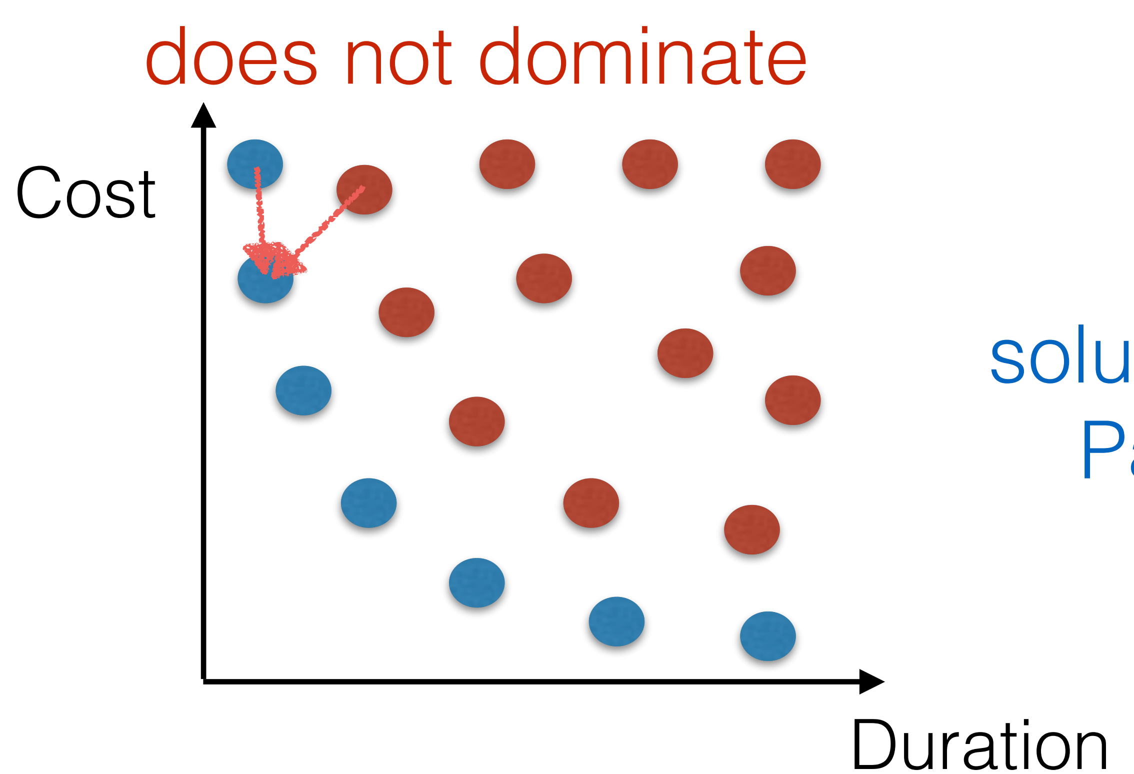
Candidate solution A:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

Candidate solution B:
cost (min) = 15,000
duration (min) = 12 months
robustness (max) = 10

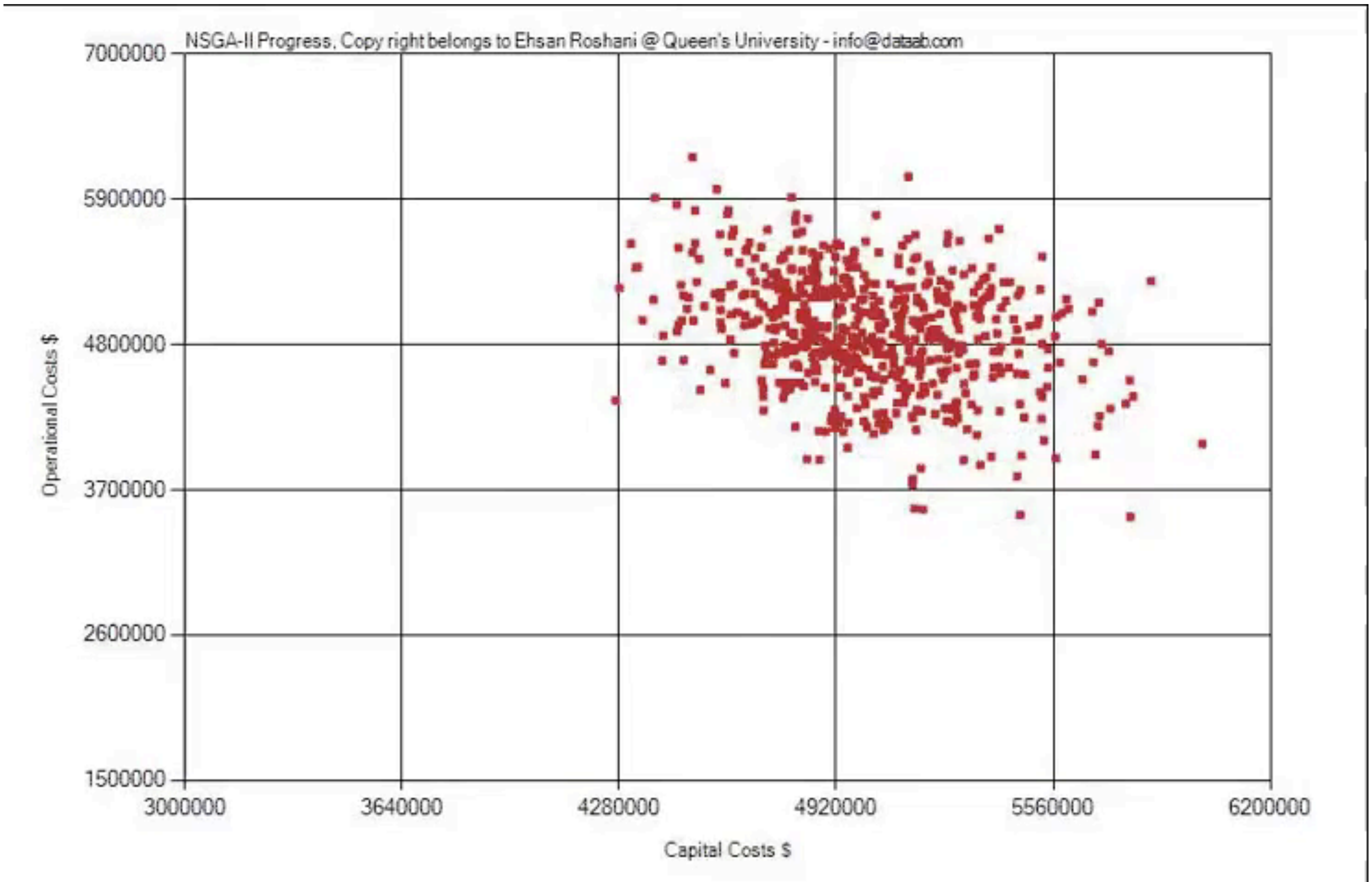
Solution B does not dominate solution A.

# Optimal Solutions

- Instead of having single best solutions, we have sets of best solutions with different trade-offs.
- Pareto front: set of solutions that are non-dominated by any other solution in the search space.
- NSGA-II aims at finding the Pareto front.



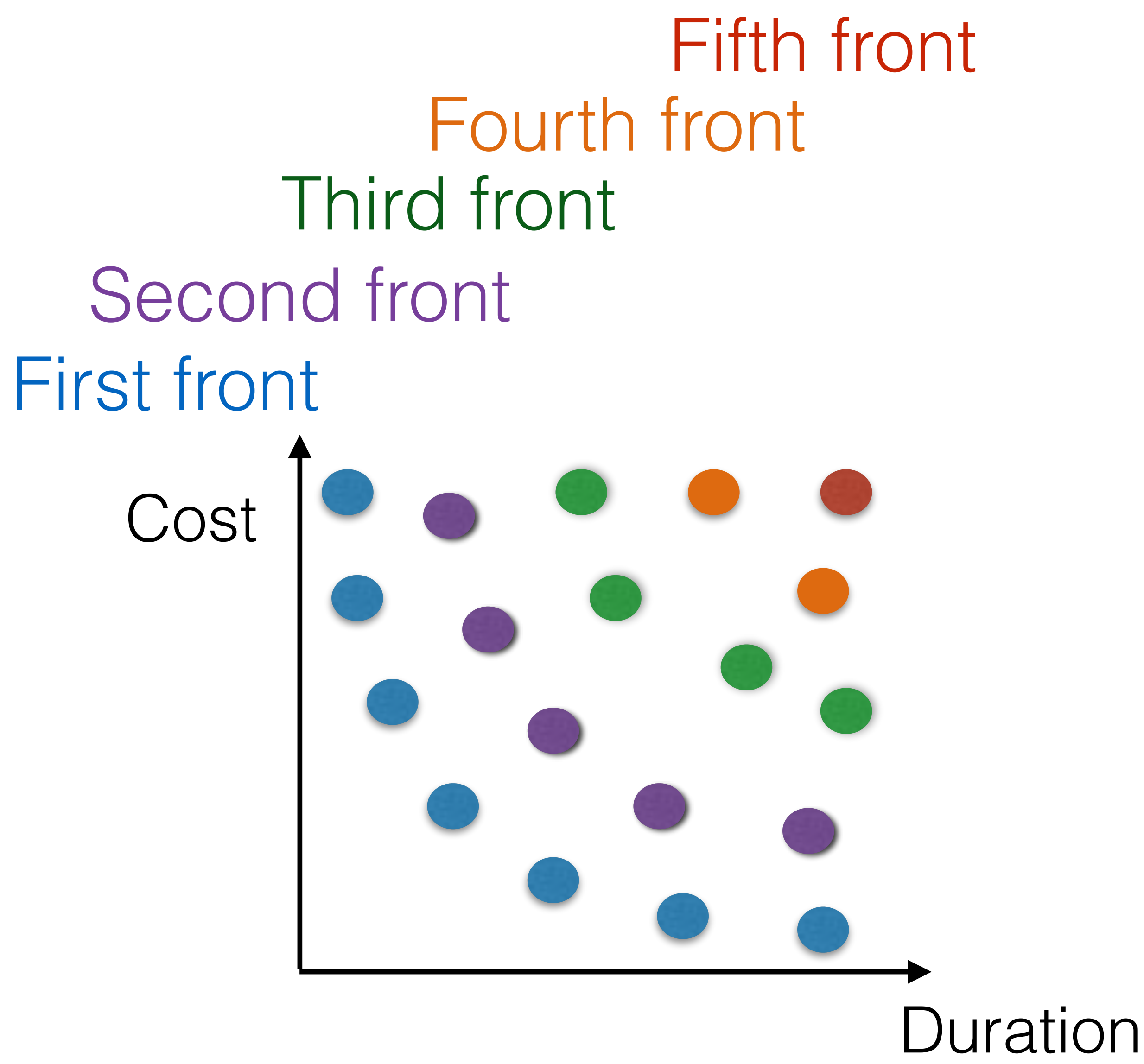

Consider that all possible solutions are plotted in this graph. Pareto front is shown in blue.




[Youtube video posted by DataAb: https://youtu.be/sEEiGM9em8s]

# Non-dominated Fronts

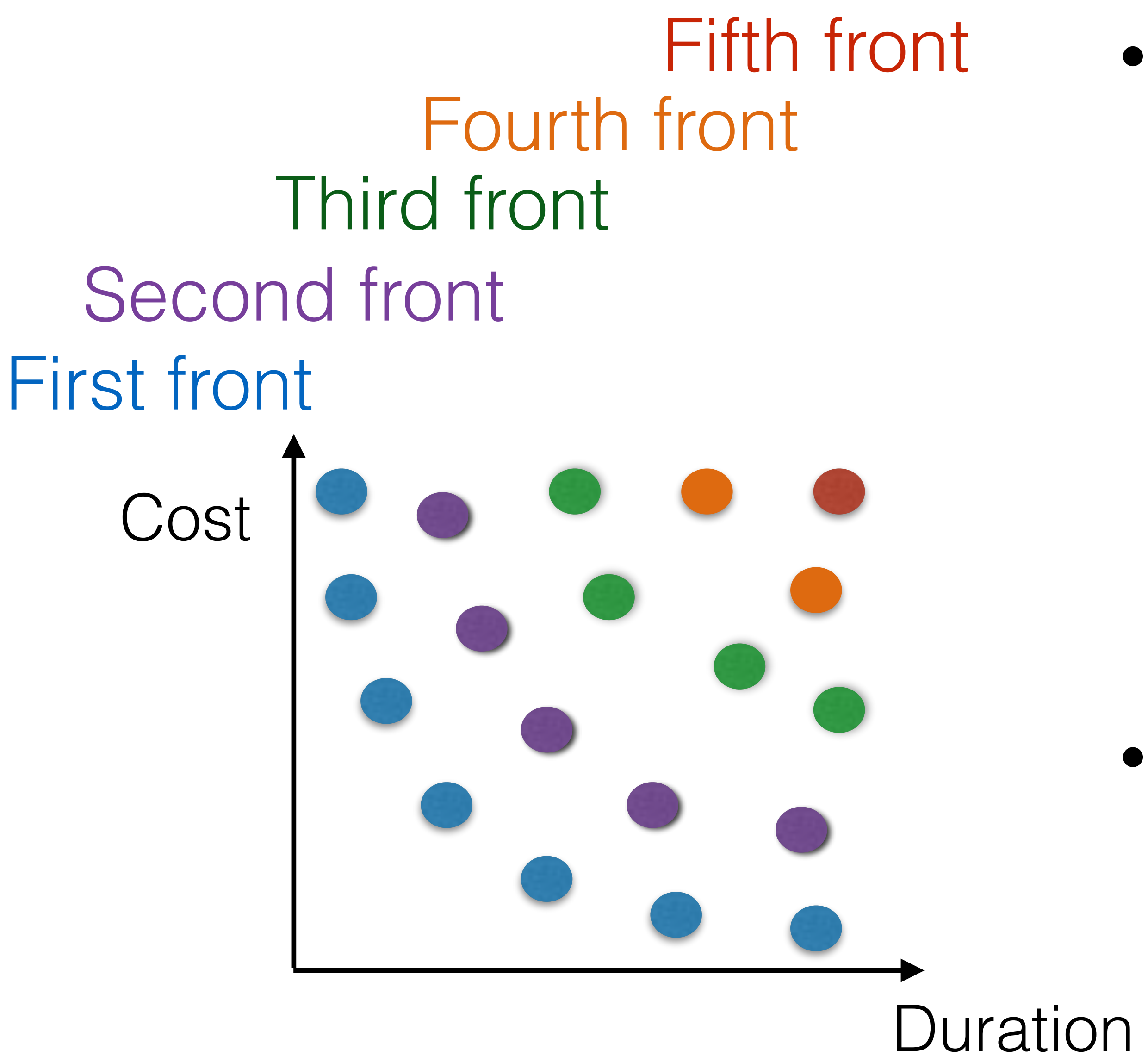
- NSGA-II sorts solutions according to their quality based on non-dominated fronts (non-dominated sorting).
- This makes it easier to compare solutions.




Consider that all solutions from a certain generation are plotted in this graph.

# Non-dominated Fronts

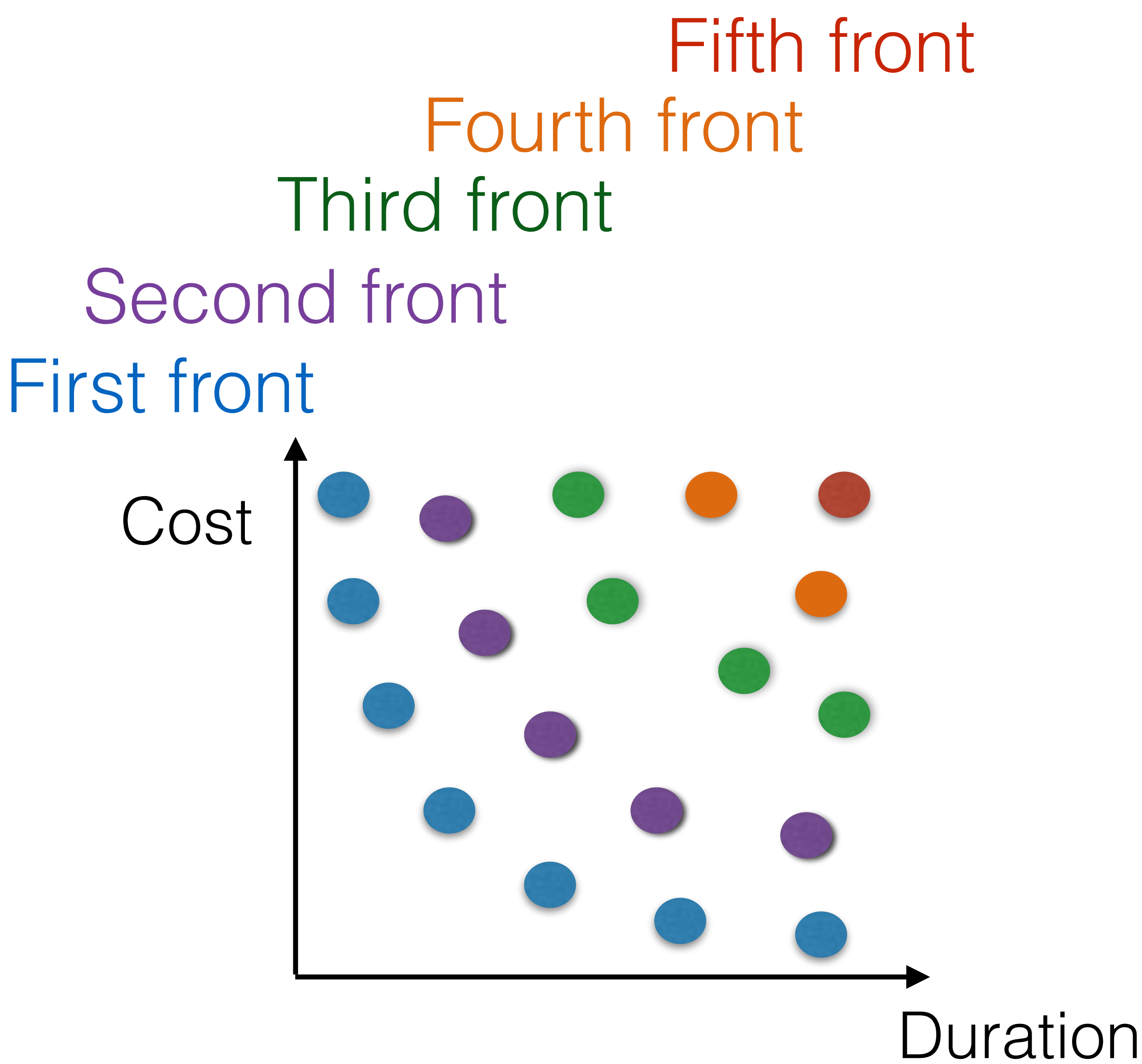
Solutions in former fronts are considered better than solutions in latter fronts and will be preferred for parents / survival selection.




- Solutions in the same front do not dominate each other, and are not dominated by any solution from latter fronts.
- Some solutions in former fronts dominate solutions in latter fronts.

# Non-dominated Fronts

NSGA-II considers solutions in former fronts as better than solutions in latter fronts and will prefer them during parents / survival selection.

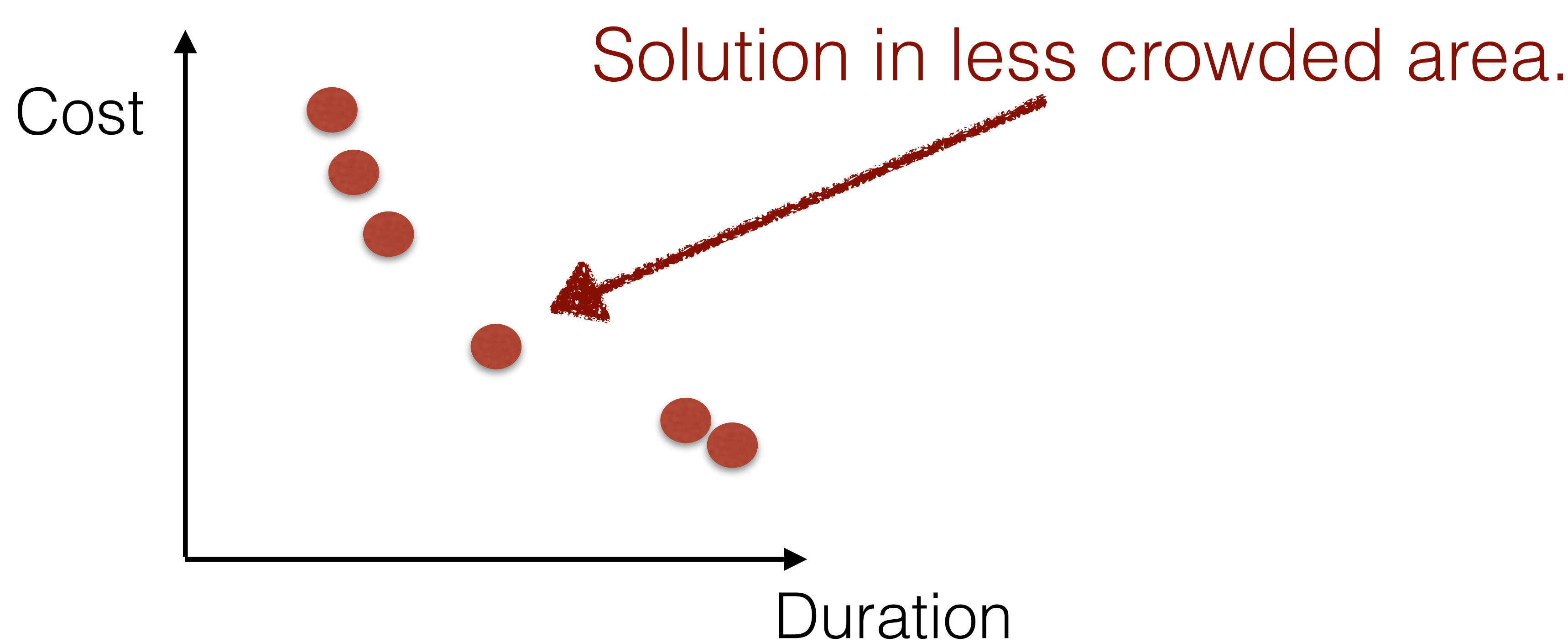



Many solutions are non-dominated by each other.

How to decide between solutions from the same front?

# Deciding Between Non-Dominated Solutions

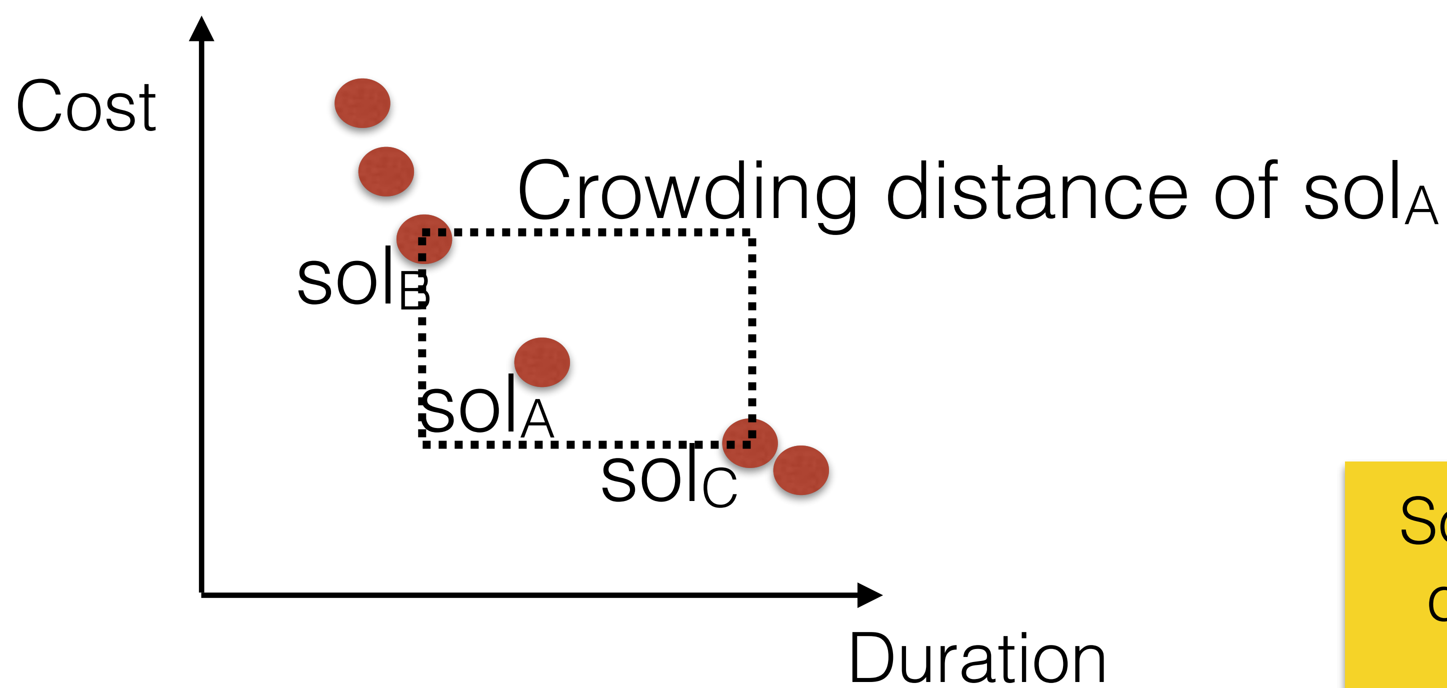
- We prefer solutions in less crowded areas of a given front.




How to determine how crowded a certain solution is?

# Crowding Distance

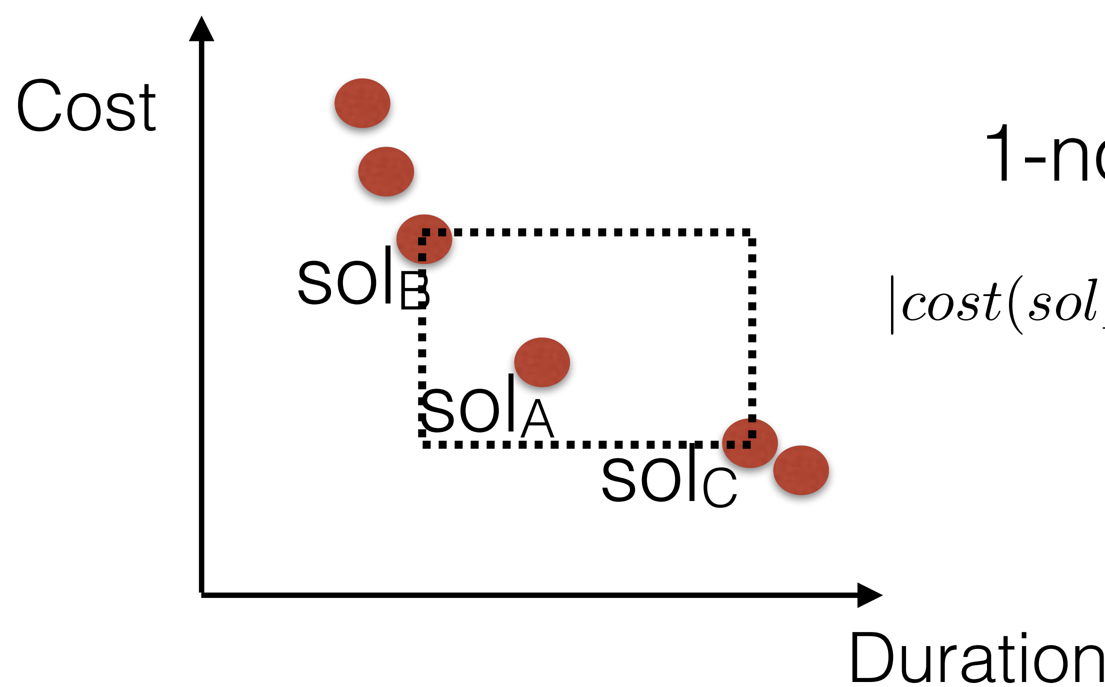
- The crowding distance of a solution defines how crowded the region where a given solution is.
    1. Consider the non-dominated front where a solution is.
    2. Find the two solutions in either side of the solution.
    3. Calculate the distance between these two solutions in the objective space.




Solutions with higher crowding distance are in less crowded regions.

# Crowding Distance

- The crowding distance of a solution defines how crowded the region where a given solution is.
  1. Consider the non-dominated front where a solution is.
  2. Find the two solutions in either side of the solution.
  3. Calculate the distance between these two solutions in the objective space.



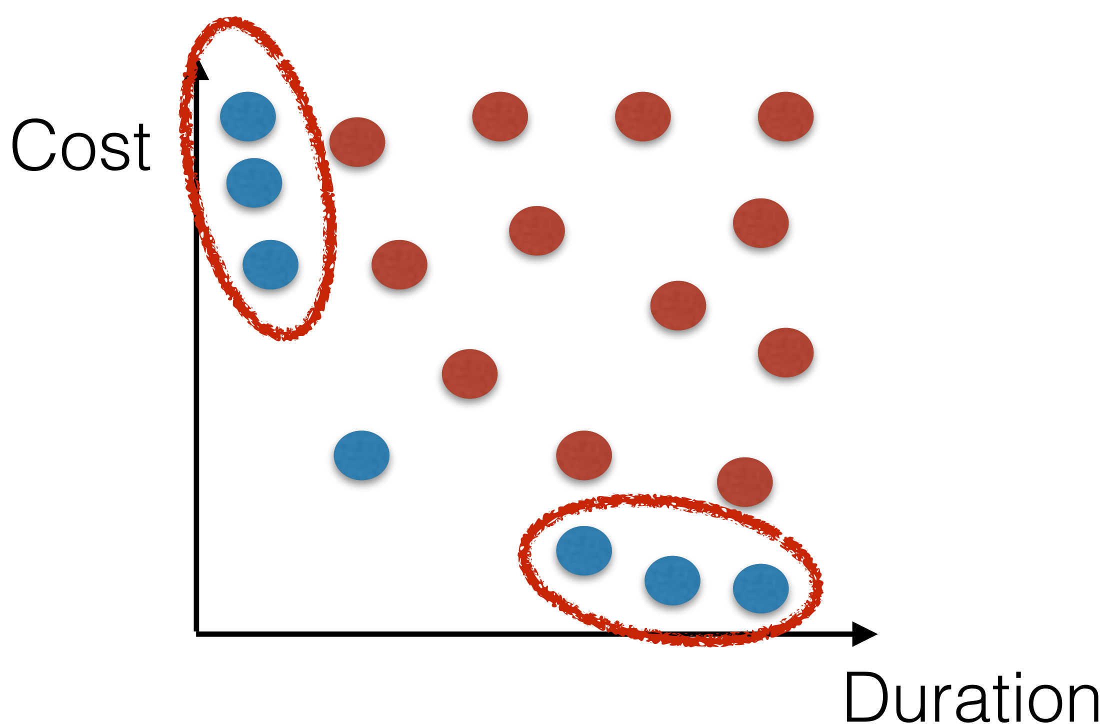

Crowding distance of $sol_A$:
1-norm distance between $sol_B$ and $sol_C$

$$|cost(sol_B) - cost(sol_C)| + |duration(sol_B) - duration(sol_C)|$$

Crowding distance for the left-most and right-most solutions is infinite.

# Why Preferring Solutions from Less Crowded Areas?

- We aim at finding the Pareto set of optimal solutions.
- Pareto set provides several trade-offs among objectives.
- If we lack diversity, we may may miss some optimal solutions, and be unable to provide a good spread of trade-offs between different objectives.




Favouring solutions from less crowded areas encourages more diversity, helping to find the Pareto set.

# Deciding Between Solutions

- If $sol_A$ is in a former non-dominated font than $sol_B$:
  - we prefer the $sol_A$.
- If $sol_A$ and $sol_B$ are in the same non-dominated front:
  - we prefer the solution from less crowded area, i.e., with higher crowding distance.

# NSGA-II Algorithm

1. Set $t = 0$ (current generation)
2. Initialise population $P_t$ with size $N$.
3. Sort $P_t$ into different non-dominated fronts.
4. Determine the crowding distance of each individual in $P_t$.
5. While $t < t_{max}$
   1. Select parents from $P_t$ using 2-tournament selection based on non-dominated fronts and crowding distance.
   2. Apply crossover to generate children individuals $C$ with probability $Pc$.
   3. Apply mutation to children individuals $C$ with probability $Pm$.
   4. $S$ <— $P_t$ $U$ $C$.
   5. Sort $S$ in different non-dominated fronts $F_0$ to $F_n$.
   6. Determine the crowding distance of each individual in $S$.
   7. Select survivors from $S$ based on non-dominated fronts and crowding distance.
   8. t <— t+1

# NSGA-II Algorithm — Survivor Selection

Elitist survivor selection that picks the best individuals according to the non-dominated fronts and the crowding distance.

1. Set $P_{t+1}$ <— {}, i <— 0
2. While size of $P_{t+1}$ + size of $F_i$ <= $N$
    1. $P_{t+1}$ <— $P_{t+1}$ U $F_i$
    2. i++
3. Top $P_{t+1}$ up with the individuals from $F_i$ that have the highest crowding distance.

While whole front $F_i$ fits within $P_{t+1}$.

# Further Reading

A fast and elitist multiobjective genetic algorithm: NSGA-II

K. Deb, A. Pratap, S. Agarwal, T. Meyarivan

IEEE Transactions on Evolutionary Computation

Vol 6, Issue 2, pages 182—197, 2002

Read until section III.

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=996017&tag=1

Optional:

Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems

Z. Wang, K. Tang, X. Yao

IEEE Transactions on Reliability

Vol 59, Issue 3, pages 563—575, 2010

http://ieeexplore.ieee.org/abstract/document/5549979/