# Multi-objective Differential Evolution in Finance

Håken Jevne

CTO & Co-founder Fronteer Solutions AS

February 19, 2019

## Outline

### Part 1

- Modern Portfolio Theory
- Differential Evolution
- Multi-objective Optimization

### Part 2

- Fronteer Solutions and Harvest
- My history and the road to where we are now
- Lessons learned

## About me

- M.Sc. in Computer Science (Bio-inspired AI) from NTNU
- Published the paper “Evolving Constrained mean-VaR Efficient Frontiers” in 2012
- CTO & Co-founder of Fronteer Solutions AS

## Fronteer Solutions AS

- Licensed fund manager
- Manages 250 MNOK in “Faktorfondet Harvest”
- 10 employees
- 3000+ investors
- Operates in Norway and Sweden
- Owned by founders, Investinor, Schibsted and FinStart Nordic

## Modern Portfolio Theory

- Harry Markowitz published the seminal work in this field in 1952 for which he later was awarded the Nobel Price in economics.
- Markowitz developed the simple but profound notion that investors should consider not only returns, but the risk associated with the investments as well.
- He studied the relationship between reward versus risk in portfolio optimization and how correlation and diversification affect risk.
- His *mean-variance* model laid the foundation for modern portfolio theory.

## Portfolio Selection

- A portfolio is a collection of financial assets. An investor may choose to allocate his or her funds amongst these different types of assets.
- Portfolio selection is concerned with the challenge of optimally allocating such funds amongst a set of risky assets so as to maximize returns and minimize risk.
- A portfolio that provides minimum risk for an expected return or provides maximal expected return given an upper bound on risk, is termed an *efficient portfolio*.

## Portfolio Definition

Consider a *finite* set of assets $i = 1, 2, \ldots, n$. A portfolio $x \in \mathbb{R}^n$ is a vector of allocation of these assets

$$x = (x_1, x_2, \ldots, x_n). \tag{1}$$

A portfolio must be fully invested, thus $\sum_{i=1}^{n} x_i = 1$ and $x_i \in [0, 1]$ for all $i$ to disallow short sales.

## Portfolio return

The expected portfolio return, $\mu_P$, is given by

$$\mu_P = \sum_{i=1}^{n} \mu_i x_i \tag{2}$$

where $\mu \in \mathbb{R}^n$ is the vector of expected asset returns and $x$ is a portfolio vector.

The daily average asset return for asset $i$:

$$\mu_i = \frac{1}{T} \sum_{t=1}^{T} r_{it}$$

where $r_{it}$ is the daily asset return for asset $i$ at day $t$, and $T$ is the number of historical returns to consider.

## Portfolio risk

The risk associated with an asset is its variance.

$$\sigma_i^2 = var(r_i) = \frac{1}{T}\sum_{t=1}^{T}(r_{it} - \mu_i)^2 \tag{3}$$

The portfolio variance is given as the sum of the weighted assets combined variance.

$$\sigma_P^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_i x_j \tag{4}$$

where the covariance of two assets is

$$\sigma_{ij} = covar(r_i, r_j) = \frac{1}{T}\sum_{t=1}^{T}(r_{it} - \mu_i)(r_{jt} - \mu_j)$$

## An interesting property of covariance

The covariance of two random variables can be less than the variance of each random variable, i.e. the risk of two assets combined can be less than the risk of each individual asset.

## Portfolio Optimization - Variance minimization

To construct a portfolio of minimal variance for a given level of return the following optimization model can be solved

$$\begin{aligned} \text{minimize} \quad & \sigma_P^2 \\ \text{subject to} \quad & \mu_P = \bar{\mu} \\ & \sum_{i=1}^{n} x_i = 1 \\ & 0 \leq x_i \leq 1 \end{aligned}$$

where $\bar{\mu}$ is the target expected return.

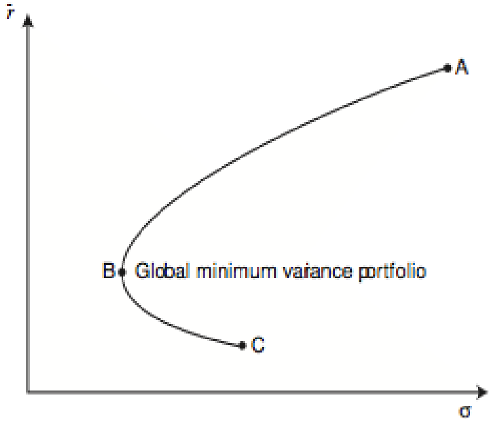
The problem can also be written in standard normal (matrix) form and solved by a quadratic solver.

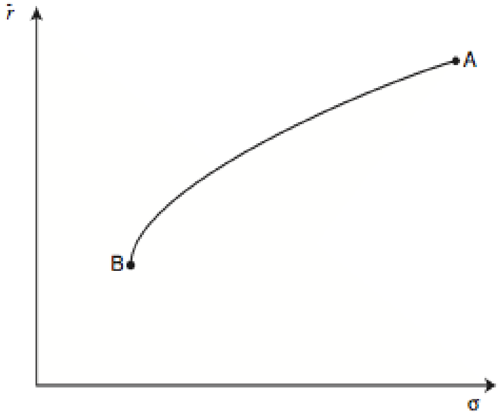
$$\begin{aligned} \text{minimize} \quad & x^\intercal S x + \bar{\mu}^\intercal x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned} \tag{5}$$

where $S$ is the covariance matrix, $Gx \leq h$ are the inequality constraints and $Ax = b$ are the equality constraints.

## The Efficient Frontier

**Efficient Portfolio.** *A portfolio is efficient if it has maximal expected return given an upper bound on risk, or, equivalently, it has minimal risk for a given expected return.*





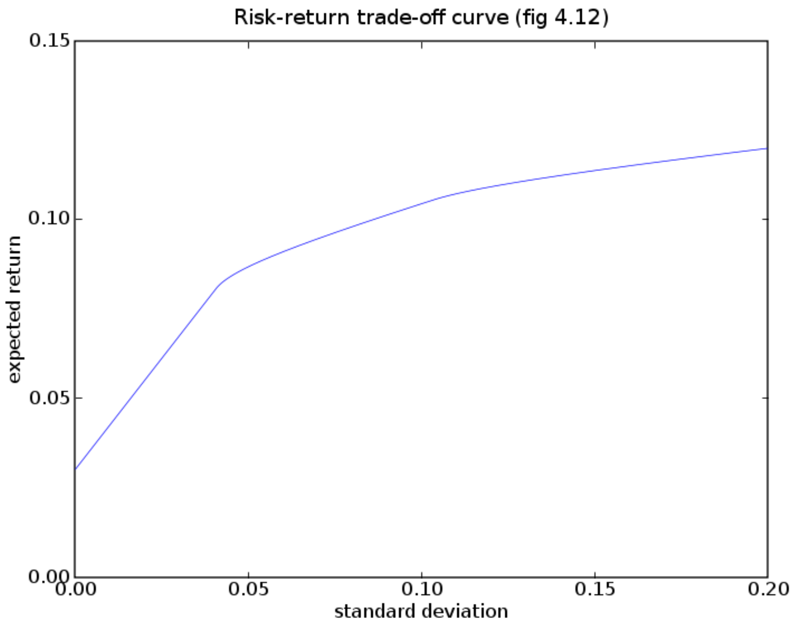
The minimum variance curve to the left and the efficient frontier to the right.

## Python example - Quadratic optimzation[1]

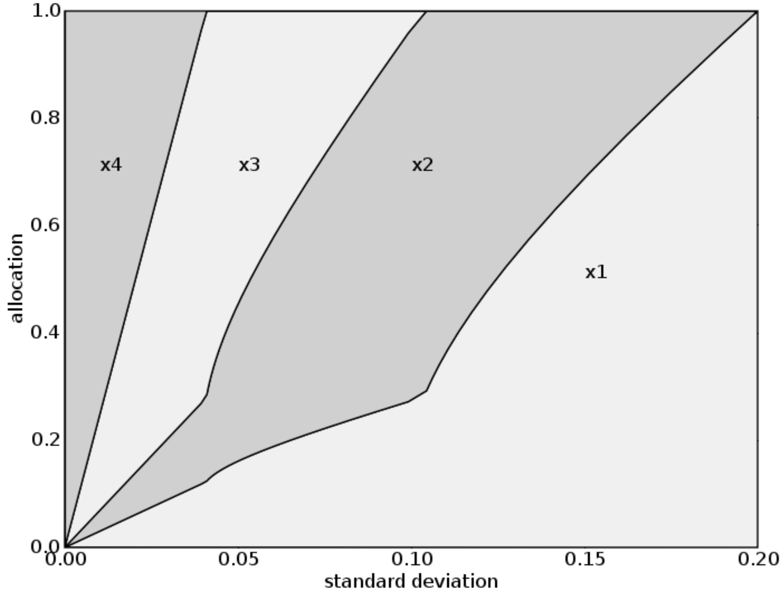
```
import pandas as pd
from cvxopt import matrix

adj_close = pd.read_cvs('data.csv')
r = adj_close.pct_change()
E_r, S = r.mean(), r.cov()
m, n = r.shape

G = matrix(0.0, (n,n))
G[::n+1] = -1.0
h = matrix(0.0, (n,1))
A = matrix(1.0, (1,n))
b = matrix(1.0)
```

[1]Example from the book *Convex Optimization* by Boyd and Vandenberghe

```
import numpy as np
from cvxopt.solvers import qp
from cvxopt.blas import dot

mus = np.linspace(E_r.min(), E_r.max(), 100)
xs = [qp(mu * S, -E_r, G, h, A, b) for mu in mus]
returns = [dot(E_r, x) for x in xs]
risks = [sqrt(dot(x, S * x)) for x in xs]
```


Risk-return trade-off curve (fig 4.12)
expected return
standard deviation
0.00
0.05
0.10
0.15
0.00
0.05
0.10
0.15
0.20

Optimal allocations (fig 4.12)



## Limitations of mean-variance optimization

The model is widely adopted mainly for its ability to easily explain the concept of risk and return but it is important to acknowledge its limitations.

**Computational Complexity.** For a large set of assets the computational difficulty associated with computing the covariance matrix and solving the quadratic programming model increases exponentially.

**Variance as Risk Measure.** Since variance is the squared standard deviation the model does not discriminate negative from positive returns, even though investors seek positive returns.

**Higher Moments.** The mean-variance model only considers the first two moments of the return distributions, namely the mean and variance. Returns are not normally distributed or even symmetrically distributed

**Portfolio Lots.** On an exchange, assets cannot be traded in fractions, only as whole lots. That means you have a rounding error from the efficient portfolio to the real acquired portfolio.

**Fragmentation.** The cost of buying portfolios in the efficient frontier increases significantly since large-scale portfolios usually contains many nonzero elements.

- Minimum transaction lots
- Transaction Costs
- Minimum Allocations

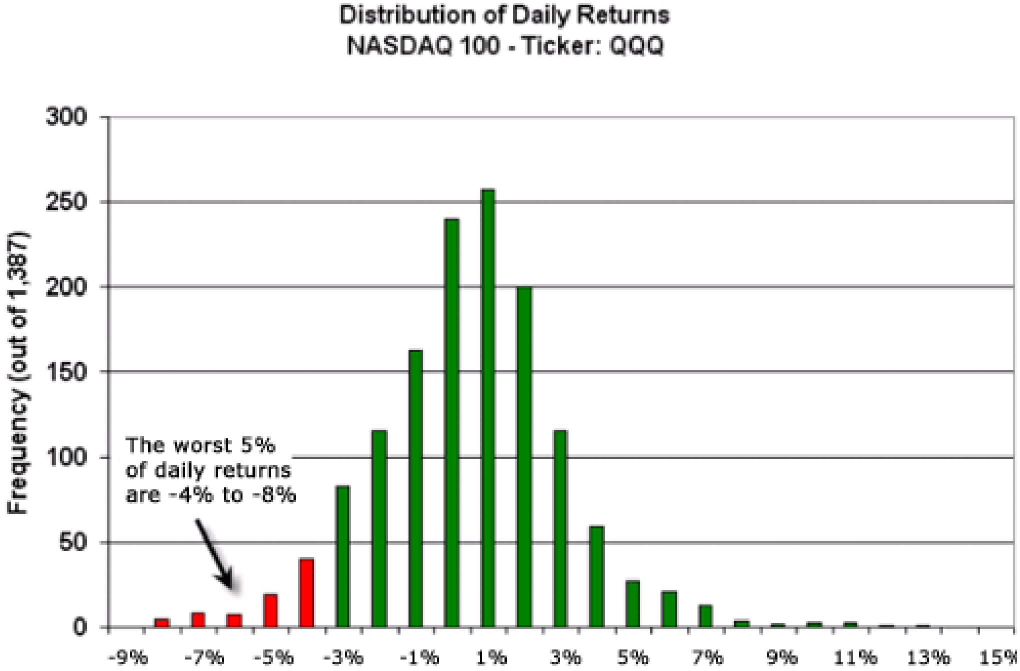
*Some of these limitations are addressed in later models, however it is difficult to implement in optimization frameworks due to the complexities they introduce in numerical optimization software.*

## Value-at-Risk (VaR) optimization

In the last decade, Value at Risk (VaR) has become one of the most popular risk measures in finance. VaR gives a quantitative measure of the possible downside of an investment.

It is best explained by an example. If an investment has a VaR of $1M over a one-week time period, at a 95% confidence interval, there is only a 5% chance the investor will lose more than $1M that week.

Risk averse investors typically set a high confidence interval while risk-seeking investors might use a lower confidence interval.


Distribution of Daily Returns
NASDAQ 100 - Ticker: QQQ
Frequency (out of 1,387)
300
250
200
150
100
50
0
The worst 5% of daily returns are -4% to -8%
-9%
-7%
-5%
-3%
-1%
1%
3%
5%
7%
9%
11%
13%
15%

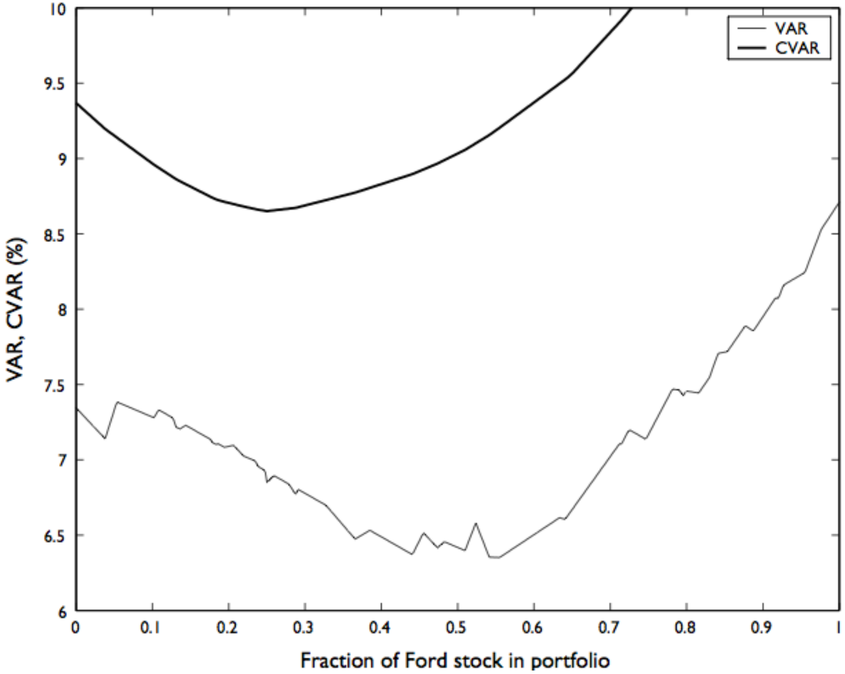
## VaR Complexity

VaR is inherently more difficult to optimize than e.g. variance. The VaR function is non-convex, non-differentiable and may exhibit many local minima, and is of a combinatorial character (it grows exponentially in computational complexity).

The complexity of optimizing VaR can be illustrated with an example from Gaivoronski and Pflug (2005). Consider two assets $x_1$ and $x_2$ and a portfolio $x$ defined by a linear combination of these assets

$$x(\lambda) = \lambda x_1 + (1 - \lambda) x_2 \qquad 0 \leq \lambda \leq 1$$

The difficulty of minimizing VaR comes from the discreteness of the observations in the historical simulations.


VAR
CVAR
VAR, CVAR (%)
Fraction of Ford stock in portfolio
6
6.5
7
7.5
8
8.5
9
9.5
10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

## Real world objective functions in finance

### Expected Returns Estimation

- Historical returns
- Factor models
- Expert analysis
- Neural Networks
- Genetic Programming

### Risk Estimation

- Variance (VAR)
- Value at Risk (VaR)
- Conditional Value at Risk (CVaR)
- Mean-absolute deviation (MAD)
- Drawdown
- Markov Chain Monte Carlo simulations
- GARCH-Copula models

## Differential Evolution (DE)

- *Storn and Price (1995,1997)* "Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces."
- Individuals are encoded as vectors and new offspring are created through the processes of *mutation* and *crossover* between a target vector and several donor vectors from the parent population.
- This differs from other evolutionary algorithms where features in the offspring are usually taken in turn from either parent.
- DE moves from exploration $\rightarrow$ exploitation of the solution space as the algorithm converges.
- DE is well suited for continuous spaces.

## Basic concepts of DE

- The initial population is chosen randomly and should be distributed uniformly throughout the solution space.
- New candidate solutions are generated by selecting a *target vector* with multiple donor vectors and applying mutation, crossover and replacement.
- Mutation is performed by combining multiple parents by vector addition, subtraction and scaling.
- Each individual has to serve as target vector once to keep diversity and the population size constant.

## Stage 1: Mutation

For each target vector $x_{i,G} = 1, 2, \ldots, NP$, randomly select $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ where $i \neq r_1 \neq r_2 \neq r_3$. A *mutant vector* $v$ is then created as follows:

$$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} - x_{r_3,G}) \tag{6}$$

where $NP$ is the population size; $G$ is the current generation and $F \in [0, 2]$ is a scaling factor for the size of the difference vector.

## Stage 2: Crossover

The mutant vector and the target vector are combined through crossover, resulting in a new vector $u$, called a *trial vector*:

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1}) \tag{7}$$

The application of crossover takes the form:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases} \tag{8}$$
$$j = 1, 2, \ldots, D$$

where $randb(j) \in [0, 1]$ is a uniform random number, $CR \in [0, 1]$ is the crossover constant and $rnbr(i) \in \{1, 2, \ldots, D\}$ is a random index.

## Stage 3: Replacement

The final stage involves fitness comparison of the resulting trial vector $u_{i,G+1}$ and the original target vector $x_{i,G}$ where the one with the highest fitness is forwarded to the next generation.

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if } f(u_{i,G+1}) > f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{9}$$

where $f$ is a fitness function.

## Differential Evolution Pseudo code

```
1:  For generation $G$
2:  for all $x$ in $P_G$ do
3:      Select randomly $x_1, x_2, x_3 \in P_G$ so that $x_1 \neq x_2 \neq x_3 \neq x$
4:      Create mutant vector $v = x_1 + F(x_2 - x_3)$
5:      for $i = 1$ to $|x|$ do
6:          if $U(0,1) < CR$ or $rnd_x(1, |x|) = i$ then
7:              $u[i] = v[i]$
8:          else
9:              $u[i] = x[i]$
10:         end if
11:     end for
12:     if $f(u) > f(x)$ then
13:         $P_{G+1} \leftarrow u$
14:     else
15:         $P_{G+1} \leftarrow x$
16:     end if
17: end for
```

## Variations in Differential Evolution

There are several variations within differential evolution. The following notation is used to separate different DE variants

$$DE/x/y/z$$

where

- x is the selected vector to mutate. This can be e.g., *rand* for a random population vector or *best* for the vector of highest fitness.
- y is the number of difference vectors used, usually 1.
- z is the type of crossover used, e.g. "*bin*" for independent binomial selection or "*exp*" for exponential selection.

The DE strategy described earlier can be represented as *DE/rand/1/bin* in this notation.

The differences amongst DE variants are mainly the way donor solutions are selected, the number of difference vectors and the type of recombination operator used.

Iorio and Li (2004) uses the strategy *DE/current-to-rand/1* which is calculated as follows:

$$v_{i,G+1} = x_{i,G} + K(x_{r_3,G} - x_{i,G}) + F(x_{r_1,G} - x_{r_2,G})$$

The shorthand description of this model states that *DE/current-to-rand/1* generates vectors that are linear combinations of the current vector $x_{i,G}$, and a randomly chosen donor $x_{r3,G}$.

The extra difference vector adds more diversity to the mutant vector $v$ and might help the algorithm “jump” out of local minima.

For a list of other DE variants see Mezura-Montes (2008).

## Multi-objective Differential Evolution (MODE)

- In the real-world, multi-objective optimization problems are very common, particularly in engineering applications.
- Multi-objective optimization problems involve multiple objectives, often conflicting, which should be optimized simultaneously.
- EA have proven very efficient at solving multi-objective optimization problems, see Deb et al. (2002).
- The portfolio selection problem is an example of a multi-objective optimization problem. The problem consists of optimizing both risk and return at the same time, i.e. maximize return while minimizing risk.

The problem, with VaR as the risk measure, is named *mean-VaR* and defined as

$$\begin{aligned} \text{maximize} \quad & R(x) \\ \text{minimize} \quad & VaR_\alpha(x) \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = 1 \\ & 0 \le x_i \le 1 \end{aligned} \tag{10}$$

where $x$ is a candidate solution i.e., portfolio vector, $R(x)$ is the portfolio return and $VaR_\alpha(x)$ is the portfolio Value-at-Risk with a confidence level equal to $1 - \alpha$.

Optimizing (10) will result in a Pareto-front which will converge to the efficient frontier as the EA populations are evolved.

## Efficient Frontier $\Rightarrow$ Pareto Optimality

Multiple-objective optimization seeks to find a set of optimal solutions. The set of solutions are bounded by a curve, termed the Pareto front.

A solution is termed Pareto-optimal (and will lie on the curve) if there exists no other feasible solution which would decrease some criteria without causing a simultaneous increase in at least one other criteria (assuming minimization),

By plotting the Pareto-front of an evolved *mean-VaR* optimization problem, with return on the $y$-axis and risk on the $x$-axis, the Pareto-front becomes the *efficient frontier* described earlier.

## NSGA-II and NSDE

The Non-dominated Sorting Genetic Algorithm (NSGA-II) is perhaps the most widely adopted multi-objective evolutionary algorithm.

The NSGA-II algorithm incorporates non-dominated sorting with crowding distance to select the best individuals from each generation.

Given two candidate vectors with differing non-dominated ranks, the solution with the lower (better) rank is preferred. Otherwise, if both solutions has the same rank, then the solution that is located in a lesser crowded region is preferred.

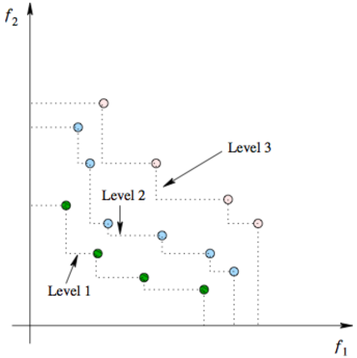
$$i \prec_n j \text{ if } (i_{rank} < j_{rank}) \text{ or } ((i_{rank} = j_{rank}) \text{ and } (i_{distance} > j_{distance}))$$

NSGA-II can be modified to use Differential Evolution for mutation and crossover as in Iorio and Li (2004) which presents the elitist Non-dominated Sorting Differential Evolution (NSDE) algorithm.

## Performing Elitism

A naive (slow) implementation works as follows

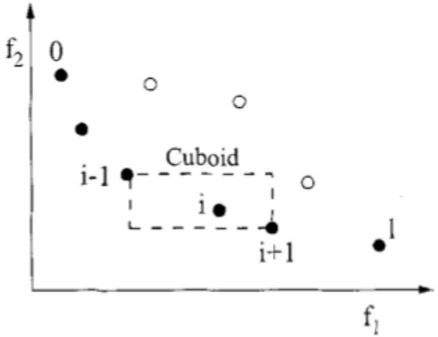
- rank all non-dominated solutions with rank 1, then remove them from the set
- rank all non-dominated solutions in the new set with rank 2, and remove them form the set
- repeat this process until all solutions have been ranked but this is computationally slow



```
fast-non-dominated-sort($P$)
for each $p \in P$
    $S_p = \emptyset$
    $n_p = 0$
    for each $q \in P$
        if $(p \prec q)$ then                    If $p$ dominates $q$
            $S_p = S_p \cup \{q\}$               Add $q$ to the set of solutions dominated by $p$
        else if $(q \prec p)$ then
            $n_p = n_p + 1$                      Increment the domination counter of $p$
    if $n_p = 0$ then                            $p$ belongs to the first front
        $p_{\text{rank}} = 1$
        $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
$i = 1$                                          Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$
    $Q = \emptyset$                              Used to store the members of the next front
    for each $p \in \mathcal{F}_i$
        for each $q \in S_p$
            $n_q = n_q - 1$
            if $n_q = 0$ then                    $q$ belongs to the next front
                $q_{\text{rank}} = i + 1$
                $Q = Q \cup \{q\}$
    $i = i + 1$
    $\mathcal{F}_i = Q$
```

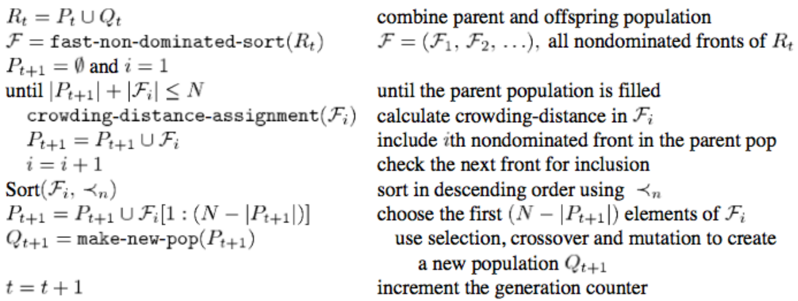
## Promoting Diversity

```
crowding-distance-assignment($\mathcal{I}$)
$l = |\mathcal{I}|$                                        number of solutions in $\mathcal{I}$
for each $i$, set $\mathcal{I}[i]_{\text{distance}} = 0$             initialize distance
for each objective $m$
  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$                            sort using each objective value
  $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$       so that boundary points are always selected
  for $i = 2$ to $(l - 1)$                                for all other points
    $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$
```

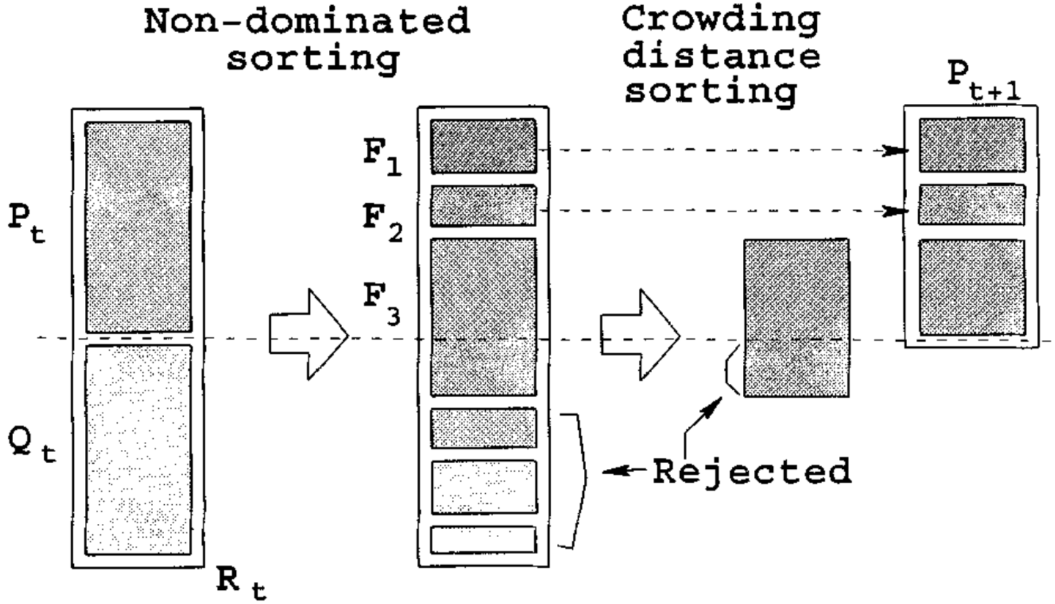


## Non-dominated Sorting Procedure

- NSGA-II starts with an initial population $P_t$ of size $N$
- The parent and offspring populations are joined and sorted to get $\mathcal{F}$ of all non-dominated fronts
- A new population $P_{t+1}$ is filled with $N$ individuals from $\mathcal{F}_\rangle$, taking the individuals of lowest rank and with highest crowding distance first
- $P_{t+1}$ is the basis for the offspring population $Q_{t+1}$ of size $N$
- The algorithm iterates until some termination condition is met



```
$R_t = P_t \cup Q_t$                                  combine parent and offspring population
$\mathcal{F} =$ fast-non-dominated-sort($R_t$)          $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all nondominated fronts of $R_t$
$P_{t+1} = \emptyset$ and $i = 1$
until $|P_{t+1}| + |\mathcal{F}_i| \leq N$               until the parent population is filled
    crowding-distance-assignment($\mathcal{F}_i$)      calculate crowding-distance in $\mathcal{F}_i$
    $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$             include $i$th nondominated front in the parent pop
    $i = i + 1$                                      check the next front for inclusion
Sort($\mathcal{F}_i$, $\prec_n$)                        sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$   choose the first $(N - |P_{t+1}|)$ elements of $\mathcal{F}_i$
$Q_{t+1} =$ make-new-pop($P_{t+1}$)                    use selection, crossover and mutation to create
                                                       a new population $Q_{t+1}$
$t = t + 1$                                          increment the generation counter
```


Non-dominated sorting
Crowding distance sorting
$P_{t+1}$
$P_t$
$F_1$
$F_2$
$F_3$
$Q_t$
Rejected
$R_t$

## Differences between DE and NSDE (singe-objective $\rightarrow$ multi-objective)

The NSDE algorithm is almost identical to the NSGA-II algorithm except for the mutation and crossover operations and a minor adjustment due to the features of DE.

In single-objective DE the trial vector $u$ from the mutation and crossover operations are compared with the target vector $x$, and the most fit vector is brought to the next population.

*In a multi-objective setting this comparison is impossible as one does not know which vector is better until all vectors are sorted and assigned a rank.*

Therefore the offspring population $Q_{t+1}$ is populated with trial vectors $u$ until it reaches a size of $N$. The individuals of $Q_{t+1}$ are evaluated by the objective functions and then sorted in junction with the trial vectors by the non-dominated sorting algorithm described above.

## Constraints handling

### Minimum Transaction Lots

$$\begin{aligned} \text{maximize} \quad & R(x) \\ \text{minimize} \quad & VaR(x) \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i c_i \leq b \\ & w_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \\ & x_i = \left\lfloor \frac{b w_i}{c_i} \right\rfloor \end{aligned} \tag{11}$$

where $b$ is the total budget to invest and $w$ is a candidate solution. All candidate solutions $w_i \in [0, 1]$ but this may violate $\sum_{i=1}^{n} w_i = 1$ i.e, the budget constraint, so all candidate solutions are normalized.

$x$ is a decoded version of the candidate solution $w$ - representing the number of lots invested in each asset. $c_i$ is the cost of buying asset $i$ and $x_i$ is an integer - the fractional budget in asset $i$ divided by its cost (rounded down).

The first inequality constraint in (11) ensures the portfolio does not cost more than the available budget. This inequality constraint indicates that not all funds are necessarily invested

$$\epsilon = b - \sum_{i=1}^{n} x_i c_i$$

The residual $\epsilon$ might be reinvested in the portfolio if it is larger than any of the asset costs.

## Transaction Costs

Traditionally, transaction costs are a piecewise linear function with a flat fee up to a threshold $\theta_1$ and a linear fee $\theta_2$ of the transaction value after that. The transaction cost is the maximum value of $\theta_1$ or $\theta_2$ multiplied by the transaction value.

The transaction cost $C(x)$, from buying a portfolio $x$, is thus:

$$C(x) = max(\theta_1, \theta_2 \sum_{i=1}^{n} x_i c_i)$$

(11) can be extended to include transaction costs by subtracting the transaction costs from the expected portfolio return (in monetary terms).

## References

H. Markowitz, “Portfolio Selection,” The Journal of Finance, vol. 7, no. 1, pp. 77–91, 1952.

R. Storn and K. Price, “Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces,” ICSI, Tech. Rep., Mar. 1995.

——, “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces,” Journal of global optimization, vol. 11, no. 4, pp. 341–359, 1997.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE transactions on evolutionary computation, vol. 6, no. 2, pp. 182–197, 2002.

A. W. Iorio and X. Li, “Solving Rotated Multi-objective Optimization Problems Using Differential Evolution,” Advances in Artificial Intelligence, pp. 861–872, 2004.