

Multi-objective

Aayush Bajaj

July 10, 2026

Contents

1	the problem	1
2	pareto optimality	1
2.1	dominance	1
2.2	the pareto set and front	2
2.3	ideal and nadir points	2
3	scalarisation	3
3.1	weighted sums	3
3.2	epsilon-constraint	4
3.3	lexicographic ordering	4
4	evolutionary methods: NSGA-II	5
5	code	5
6	see also	7

single-objective optimisation is a polite fiction. real decisions trade cost against quality, return against risk, speed against accuracy — and the objectives *disagree*, otherwise you would not have listed them separately. multi-objective optimisation refuses to mash them into one number prematurely. the price of that honesty: “the optimum” stops being a point and becomes a *set* — the pareto front — and half the subject is about how to trace it, the other half about how to pick from it.

*the ordering is named for
pareto (1896) though edgeworth
had the idea in 1881 — hence
the pedantic name
edgeworth-pareto optimality*

1 the problem

$$\min_{x \in X} F(x) = (f_1(x), f_2(x), \dots, f_M(x)), \quad (1)$$

with decision space $X \subseteq \mathbb{R}^n$ and $M \geq 2$ objectives, all minimised by convention (flip signs to taste). the image $Y = F(X) \subseteq \mathbb{R}^M$ is the **attainable set** in objective space. because $\mathbb{R}^M$ has no total order for $M \geq 2$, “minimise F ” is not yet a well-posed instruction — the first job is to say precisely which points beat which.

2 pareto optimality

2.1 dominance

fix two objective vectors $y, y' \in \mathbb{R}^M$. componentwise comparison gives three grades (terminology varies across books; this is the common modern usage):

- **weak dominance:** $y \preceq y'$ iff $y_i \leq y'_i$ for all i . (allows $y = y'$.)
- **dominance:** $y \prec y'$ iff $y_i \leq y'_i$ for all i **and** $y_k < y'_k$ for at least one k — no worse anywhere, strictly better somewhere.
- **strict (strong) dominance:** $y_i < y'_i$ for *all* i — strictly better everywhere.

dominance is a strict partial order: irreflexive, transitive, and — the crucial novelty versus $M = 1$ — *partial*. two points can be mutually non-dominated ($y_1 < y'_1$ but $y_2 > y'_2$), and then the mathematics simply has no opinion about which is better. that agnosticism is the whole content of the subject.

2.2 the pareto set and front

- $x^* \in X$ is **pareto optimal** iff no $x \in X$ has $F(x) \prec F(x^*)$: you cannot improve any objective without paying in another.
- x^* is **weakly pareto optimal** iff no $x \in X$ strictly dominates it — nothing is better in *every* objective simultaneously. every pareto optimum is weakly pareto optimal; the converse fails (a weak optimum may admit a free improvement in one objective at zero cost in the rest — a flat segment of the boundary).
- the **pareto set** is the set of pareto-optimal x in decision space; its image, the **pareto front**, lives in objective space. for $M = 2$ the front is (part of) the lower-left boundary of Y , monotonically trading f_1 against f_2 .

2.3 ideal and nadir points

two reference corners bracket the front:

- **ideal point:** $z_i^{\text{ideal}} = \min_{x \in X} f_i(x)$ — each coordinate optimised *separately*. attaining all coordinates at once would require the objectives not to conflict; generically $z^{\text{ideal}} \notin Y$.
- **nadir point:** $z_i^{\text{nad}} = \max\{f_i(x) : x \text{ pareto optimal}\}$ — the worst value of f_i over the pareto set, not over all of X (the latter is just the maximum and is useless). for $M = 2$ the nadir falls out of the pay-off table: optimise each objective alone, evaluate the other there. for $M \geq 3$ the pay-off table can over- or under-estimate the true nadir, and computing it exactly is genuinely hard — an honest annoyance, since most scalarisation schemes want the range $[z_i^{\text{ideal}}, z_i^{\text{nad}}]$ to normalise objectives or place constraint bounds.

the utopian point is the ideal minus a small positive vector — strictly unattainable, useful as a reference that no feasible point can tie

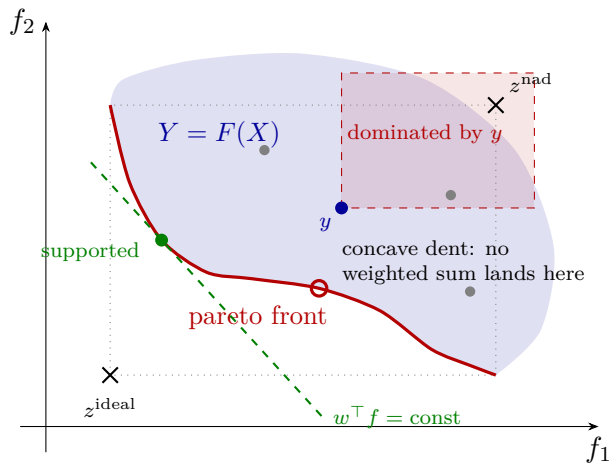


Figure 1: objective space of a biobjective minimisation. the shaded blob is the attainable set; its lower-left boundary (red) is the pareto front, bracketed by the ideal and nadir corners. everything up-right of a point is dominated by it. the dashed supporting line certifies the green shoulder point as a weighted-sum optimum — but no slope of that line can ever rest in the concave dent.

3 scalarisation

a **scalarisation** converts the vector problem into a parametric family of scalar problems; sweeping the parameter traces the front, and each member of the family is solved by ordinary machinery (simplex, newton, whatever fits). where the human enters matters too:

- **a priori:** encode preferences first (weights, priorities, goals), solve once.
- **a posteriori:** generate the whole front, let the decision-maker point at it afterwards.
- **interactive:** alternate — solve, show, adjust, repeat.

the two workhorse scalarisations behave very differently, and the difference is geometric.

3.1 weighted sums

pick weights $w \in \mathbb{R}^M$, $w \geq 0$, $w \neq 0$ (normalise $\sum_i w_i = 1$ if you like) and solve

$$\min_{x \in X} \sum_{i=1}^M w_i f_i(x). \quad (2)$$

what you can trust:

- **sufficiency.** if $w > 0$ componentwise, every minimiser is pareto optimal. proof in one line: if $F(x') \prec F(x^*)$ then $w^\top F(x') < w^\top F(x^*)$ — no worse in any coordinate, strictly better in one, all weights positive — contradicting minimality. if some $w_i = 0$ you only get *weak* pareto optimality (the zero-weighted objectives are invisible and may be improvable for free).
- **necessity, under convexity.** if X is convex and every f_i is convex, then every pareto-optimal point minimises the weighted sum for *some* $w \geq 0$, $w \neq 0$. this is the supporting-hyperplane theorem in disguise: the set $Y + \mathbb{R}_{\geq 0}^M$ (attainable points plus everything they dominate) is convex, front points sit on its boundary, and a convex set admits a supporting hyperplane at each boundary point; the hyperplane's normal is the weight vector.¹

the geometry is worth internalising for $M = 2$: the level sets of $w_1 f_1 + w_2 f_2$ are parallel lines of slope $-w_1/w_2$. minimising slides such a line down-left until it last touches Y — the touching point is the optimum, and the line *supports* Y there, as in the figure above.

the honest failure. on a **non-convex front** this breaks, structurally and unfixably. a point in a concave dent of the front lies strictly *above* the chord joining the

¹the standard theory references: k. miettinen, *nonlinear multiobjective optimization*, kluwer, 1999; m. ehrgott, *multicriteria optimization*, 2nd ed., springer, 2005.

convex shoulders either side of it. any candidate supporting line through the dent point has some slope; whichever slope you pick, sliding it down-left it will strike a shoulder *before* it reaches the dent — equivalently, for every w at least one shoulder has a strictly smaller weighted sum than the dent point. so dent points are minimisers for **no** choice of weights: the weighted-sum sweep is blind to exactly the part of the front where the trade-off is most interesting. two further quiet failures even on convex fronts: a uniform grid of weights does *not* give a uniform spread of front points (flat and steep stretches of the front hog or starve the samples), and objectives on different scales need normalising by the ideal-nadir range before the weights mean anything.

the reachable points are precisely the front points on the lower-left boundary of the convex hull of the attainable set

3.2 epsilon-constraint

keep one objective as *the* objective and demote the rest to constraints:²

$$\begin{aligned} \min_{x \in X} \quad & f_j(x) \\ \text{subject to} \quad & f_i(x) \leq \varepsilon_i \quad \text{for all } i \neq j. \end{aligned}$$

sweep the bounds ε_i over $[z_i^{\text{ideal}}, z_i^{\text{nad}}]$ and collect solutions. because the method carves the feasible set with axis-aligned constraints rather than probing it with hyperplanes, **convexity is irrelevant**: every pareto-optimal point is a solution of the epsilon-constraint problem for suitable bounds (take $\varepsilon_i = f_i(x^*)$) — including the dent points that weighted sums can never see.

the honest caveats:

- **weak optimality.** a solution is guaranteed only *weakly* pareto optimal in general. if the minimiser is unique it is genuinely pareto optimal; if not, the solver may hand you a point with slack in some $f_i \leq \varepsilon_i$ that could be tightened for free. standard clean-ups: a **lexicographic second phase** (holding f_j at its optimum, minimise the remaining objectives), or the **augmented** objective $f_j + \delta \sum_{i \neq j} f_i$ for a tiny $\delta > 0$, which breaks such ties in favour of properly pareto-optimal points.
- **bound placement needs the ranges.** useful ε grids need (estimates of) ideal and nadir values — which, for $M \geq 3$, are themselves nontrivial to compute. bounds set too tight make the subproblem infeasible; part of the sweep budget is always spent discovering where the front actually lives.
- **cost.** one full single-objective solve per grid point, and the grid is $(M - 1)$ -dimensional — fine for $M = 2$, painful by $M = 4$.

3.3 lexicographic ordering

when the objectives have a genuine hierarchy — safety before cost before comfort — impose a strict priority order and solve a cascade:

$$z_1 = \min_{x \in X} f_1(x), \quad z_2 = \min\{f_2(x) : x \in X, f_1(x) = z_1\}, \quad \dots \quad (3)$$

²introduced by y. y. haimes, l. s. lasdon and d. a. wismer, *on a bicriterion formulation of the problems of integrated system identification and system optimization*, ieee transactions on systems, man, and cybernetics 1(3), 1971.

each stage optimises the next objective over the optimal set of the previous ones. lexicographic optima are always pareto optimal: a dominating point would have to match every objective and beat one — but then it would beat the cascade at the first objective where they differ, contradicting optimality at that stage.

- **the character of the solution:** absolutist. an infinitesimal gain in priority 1 outweighs *any* gain in priority 2 — no finite trade-off rate exists. this is exactly right for hard hierarchies (regulatory limits before profit) and exactly wrong for genuine trade-offs, where you want the front, not one corner of it.
- **in practice:** tolerance relaxations ($f_1(x) \leq z_1 + \delta_1$ instead of equality) soften the cascade; and **preemptive goal programming** is precisely lexicographic minimisation of deviations from stated targets — the natural next page is [goal programming](#), which trades the ordering for explicit aspiration levels.

4 evolutionary methods: NSGA-II

when the problem is black-box, non-convex, or you want the whole front from one run, population methods dominate the practice, and NSGA-II³ is the reference point. it maintains a population of N candidate solutions and, each generation: creates N offspring by tournament selection, crossover and mutation; pools parents and offspring (elitist $(\mu + \lambda)$ survival — a non-dominated point can only be lost to a better one); runs **fast non-dominated sorting** on the pooled $2N$ points, peeling them into ranked fronts (rank 1 = non-dominated in the pool, rank 2 = non-dominated once rank 1 is removed, and so on) at $O(MN^2)$ cost per generation; refills the next population front by front; and truncates the one front that overflows using **crowding distance** — for each point, the sum over objectives of the normalised gap between its two neighbours when that front is sorted along that objective, with boundary points assigned infinite distance so the extremes always survive. selection prefers lower rank, breaking ties by larger crowding distance, which pushes the population both *toward* the front and *along* it. honesty requires two notes: there is no convergence guarantee to the true pareto front, and beyond three-or-so objectives dominance itself goes limp — almost every pair of random points is mutually non-dominated, ranks collapse, and the field moves to reference-direction (NSGA-III) or indicator-based methods.

5 code

epsilon-constraint on a biobjective LP — the pentagon from the [linear programming](#) page, now pulled in two directions: maximise profit $3x + 5y$ (so $f_1 = -(3x + 5y)$) while minimising emissions $f_2 = 2x + y$. then the same two scalarisations head-to-head on an analytic problem built to have a concave front: $f_1 = x$, $f_2 = 1 - x^2$ on $x \in [0, 1]$, whose entire curve is pareto optimal.

```

1 import numpy as np
2 from scipy.optimize import linprog, minimize_scalar
3
4 # biobjective LP over the pentagon x <= 4, 2y <= 12, 3x + 2y <= 18, x,y >= 0
5 # f1 = -(3x + 5y) (maximise profit)
6 # f2 = 2x + y (minimise emissions)
7 A_ub = [[1, 0], [0, 2], [3, 2]]
8 b_ub = [4, 12, 18]
9

```

³k. deb, a. pratap, s. agarwal and t. meyarivan, *a fast and elitist multiobjective genetic algorithm: NSGA-II*, iee transactions on evolutionary computation 6(2), 2002, pp. 182-197.

```

10 print("epsilon-constraint sweep: min f1 s.t. f2 <= eps")
11 print(f"{'eps':>5} {'x':>6} {'y':>6} {'f1':>8} {'f2':>6}")
12 for eps in np.arange(0, 10.5, 1.0):
13     res = linprog([-3, -5], A_ub=A_ub + [[2, 1]], b_ub=b_ub + [eps],
14                 bounds=[(0, None)] * 2, method="highs")
15     x, y = res.x
16     print(f"{'eps':>5.1f} {'x':>6.3f} {'y':>6.3f} {-(3*x+5*y):>8.3f} {2*x+y:>6.3f}")
17
18 # non-convex front: min f1 = x, min f2 = 1 - x^2, x in [0, 1]
19 # whole curve {(x, 1 - x^2)} is pareto optimal, but the front is concave.
20 print("\nweighted-sum sweep on the concave front: min w f1 + (1-w) f2")
21 print(f"{'w':>5} {'x*':>6} {'f1':>6} {'f2':>6}")
22 for w in np.arange(0.0, 1.01, 0.125):
23     res = minimize_scalar(lambda x: w * x + (1 - w) * (1 - x * x),
24                         bounds=(0, 1), method="bounded")
25     # bounded brent can stall on an interior maximum; compare endpoints too
26     cands = [res.x, 0.0, 1.0]
27     x = min(cands, key=lambda x: w * x + (1 - w) * (1 - x * x))
28     print(f"{'w':>5.3f} {'x':>6.3f} {'x*':>6.3f} {1 - x*x:>6.3f}")
29
30 print("\nepsilon-constraint on the same front: min f2 s.t. f1 <= eps")
31 print(f"{'eps':>5} {'x*':>6} {'f1':>6} {'f2':>6}")
32 for eps in np.arange(0.0, 1.01, 0.125):
33     res = minimize_scalar(lambda x: 1 - x * x, bounds=(0, eps if eps > 0 else
34     ↪ 1e-12),
35                         method="bounded")
36     x = res.x
37     print(f"{'eps':>5.3f} {'x':>6.3f} {'x*':>6.3f} {1 - x*x:>6.3f}")

```

epsilon-constraint sweep: min f1 s.t. f2 <= eps

eps	x	y	f1	f2
0.0	0.000	-0.000	-0.000	0.000
1.0	0.000	1.000	-5.000	1.000
2.0	0.000	2.000	-10.000	2.000
3.0	0.000	3.000	-15.000	3.000
4.0	0.000	4.000	-20.000	4.000
5.0	0.000	5.000	-25.000	5.000
6.0	-0.000	6.000	-30.000	6.000
7.0	0.500	6.000	-31.500	7.000
8.0	1.000	6.000	-33.000	8.000
9.0	1.500	6.000	-34.500	9.000
10.0	2.000	6.000	-36.000	10.000

weighted-sum sweep on the concave front: min w f1 + (1-w) f2

w	x*	f1	f2
0.000	1.000	1.000	0.000
0.125	1.000	1.000	0.000
0.250	1.000	1.000	0.000
0.375	1.000	1.000	0.000
0.500	0.000	0.000	1.000
0.625	0.000	0.000	1.000
0.750	0.000	0.000	1.000
0.875	0.000	0.000	1.000
1.000	0.000	0.000	1.000

epsilon-constraint on the same front: min f2 s.t. f1 <= eps

eps	x*	f1	f2
0.000	0.000	0.000	1.000

0.125	0.125	0.125	0.984
0.250	0.250	0.250	0.938
0.375	0.375	0.375	0.859
0.500	0.500	0.500	0.750
0.625	0.625	0.625	0.609
0.750	0.750	0.750	0.438
0.875	0.875	0.875	0.234
1.000	1.000	1.000	0.000

reading the tables:

- the LP front is **piecewise linear with a kink**. up to $\varepsilon = 6$ the sweep buys profit at 5 per unit of emissions (raising y , the efficient product); past the kink — the vertex $(0, 6)$, where the $2y \leq 12$ constraint bites — it must switch to x at only 1.5 per unit. kinks in an LP front are images of vertices of the feasible polytope; the shadow-price drop from 5 to 1.5 is the dual story of the same event. (the -0.000 s are the solver's signed zeros — cosmetic, not error.)
- the weighted-sum sweep on the concave front finds **two points, ever**: nine weight choices, and every single one lands on an endpoint, snapping from one to the other as w crosses $1/2$. the entire open interior of the front — all of it pareto optimal — is invisible, exactly as the supporting-line argument predicts.
- the epsilon-constraint sweep on the same problem walks the whole front, dent and all, in nine evenly spaced steps. (here each minimiser is unique, so every returned point is genuinely — not just weakly — pareto optimal; no clean-up phase needed.)

6 see also

- **goal programming** — aspiration levels and deviation variables; the preemptive variant is lexicographic ordering in disguise
- **linear programming** — the scalarised subproblems above, and where the front's kinks come from
- **non-linear programming** — the machinery for non-convex scalarised subproblems
- **stochastic programming** — expected cost versus risk is a multi-objective problem wearing a probability costume
- **robust optimisation** — nominal performance versus worst-case performance, another two-objective trade