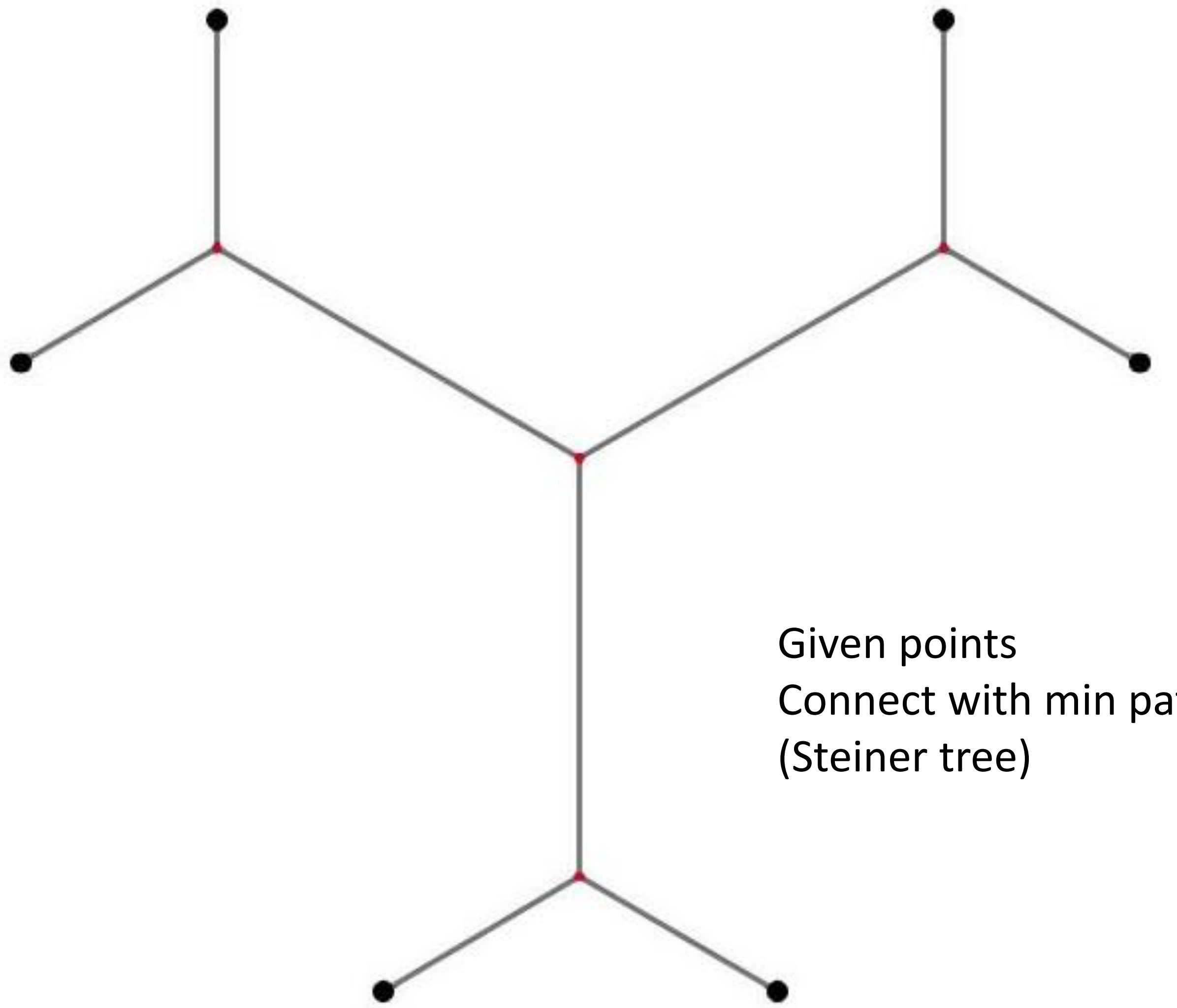
Design Optimization Problems

Michael Emmerich

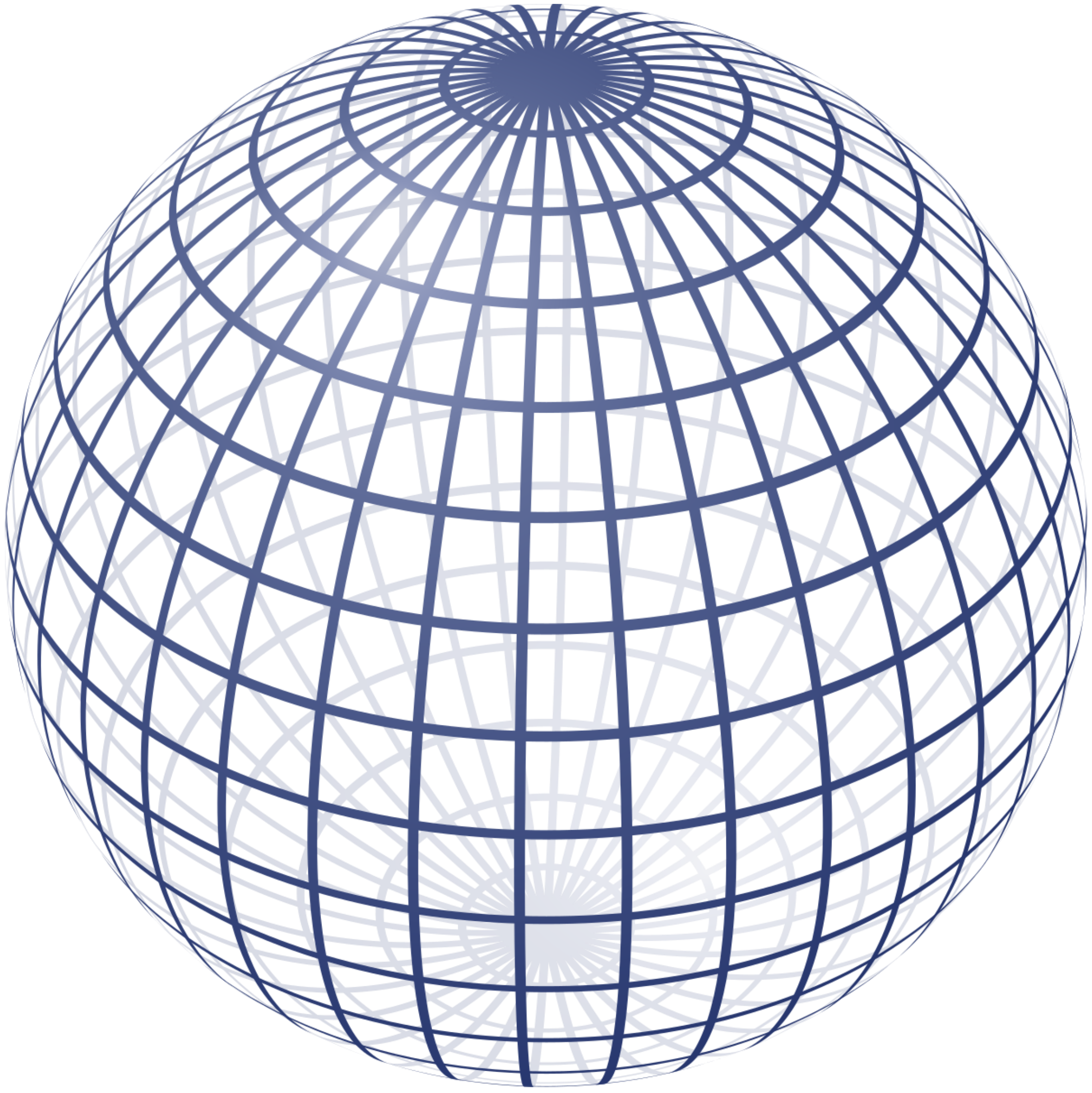
Learning Goals

- Platon’s dream: What are ideal solutions to design problems?
- Design as a discipline

Perfect Structures

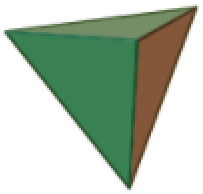
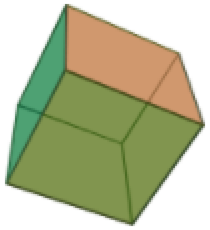
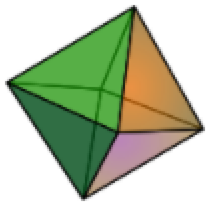
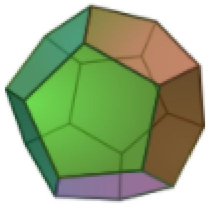
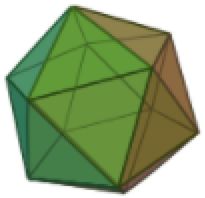


Given points
Connect with min path
(Steiner tree)



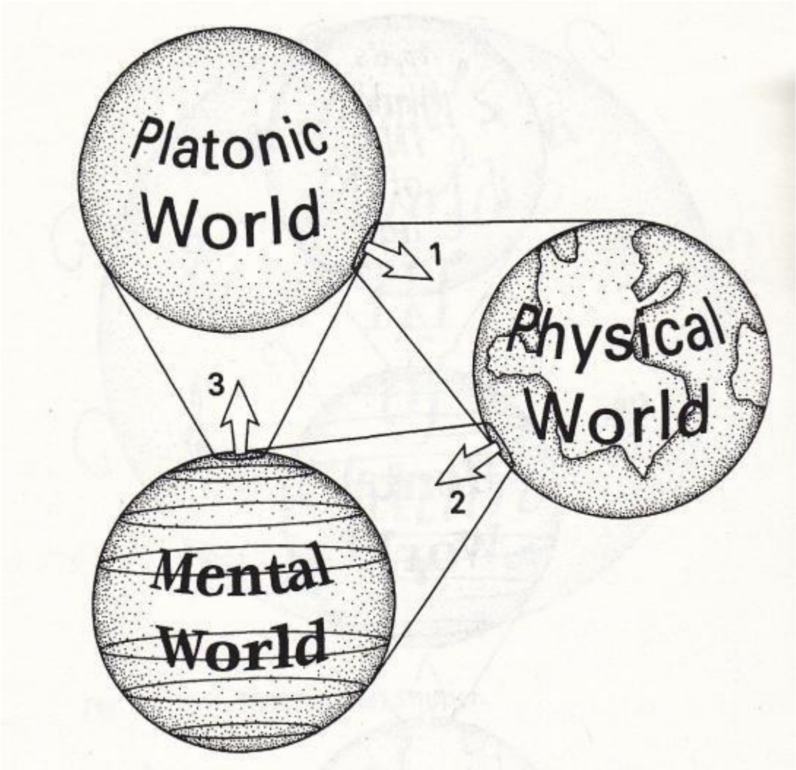
Min Surface, Volume = const

Platonic bodies – ‘Ideal’ shapes

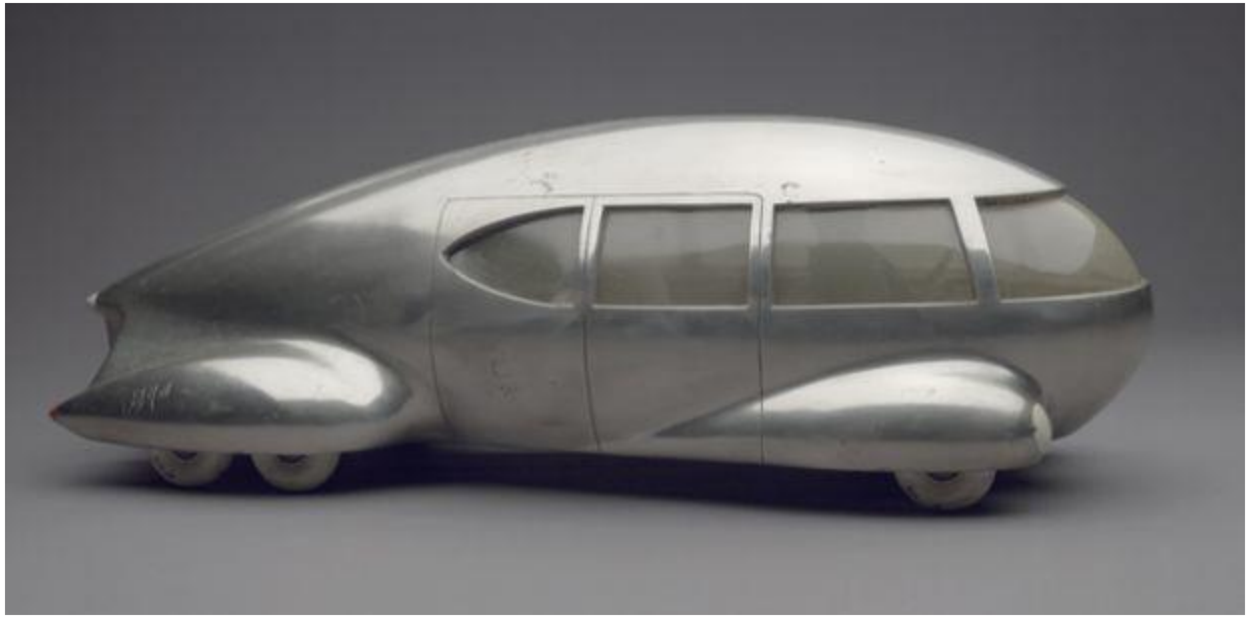
Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Four faces	Six faces	Eight faces	Twelve faces	Twenty faces
 (Animation) (3D model)	 (Animation) (3D model)	 (Animation) (3D model)	 (Animation) (3D model)	 (Animation) (3D model)

Shapes of maximally even (equal) surfaces at their boundary
https://en.wikipedia.org/wiki/Platonic_solid

Penrose: Three worlds



Perfect Designs, $c_w = \min$

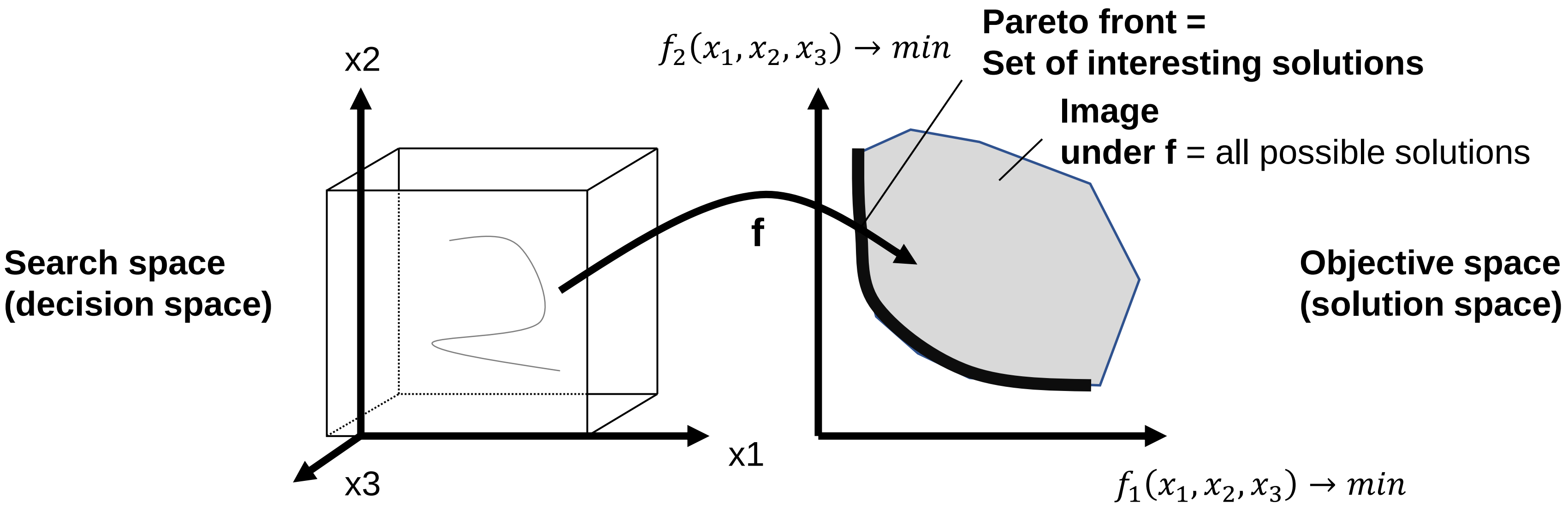


Norman Bel Geddes, "Motor Car No. 9 (without tail fin)"

Aalto shape (beauty is hard to measure …)

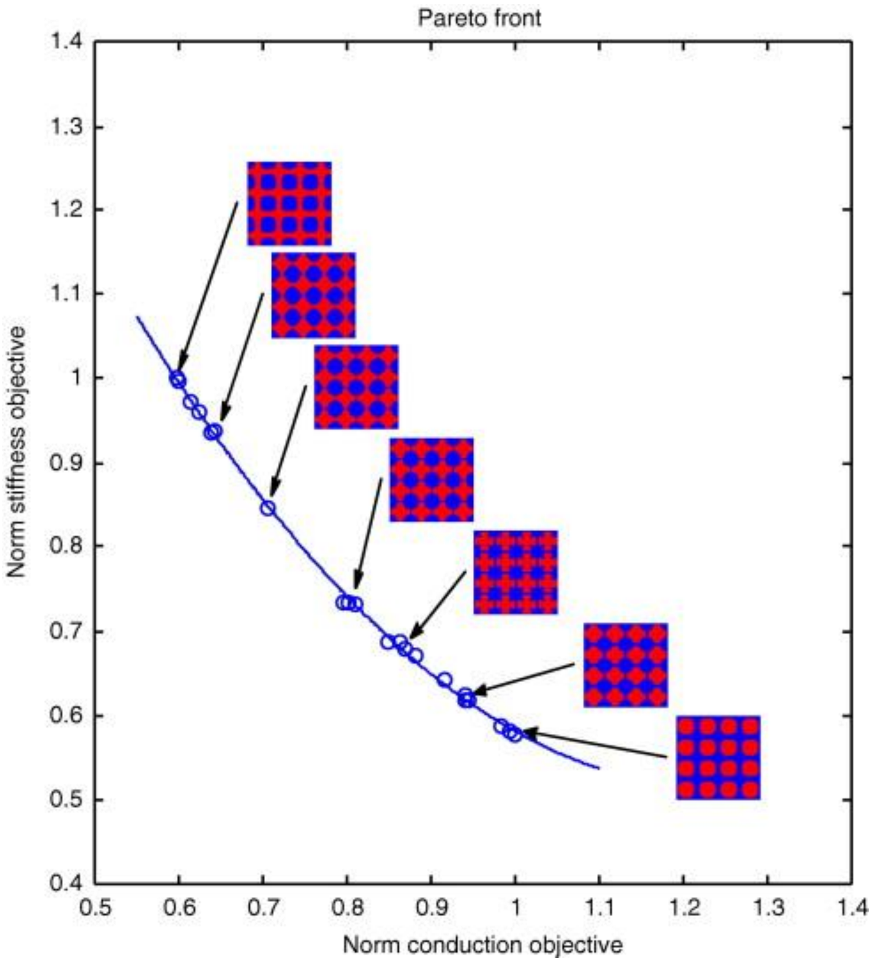
Think of examples of beautiful shapes in nature …

Pareto optimality



Research questions

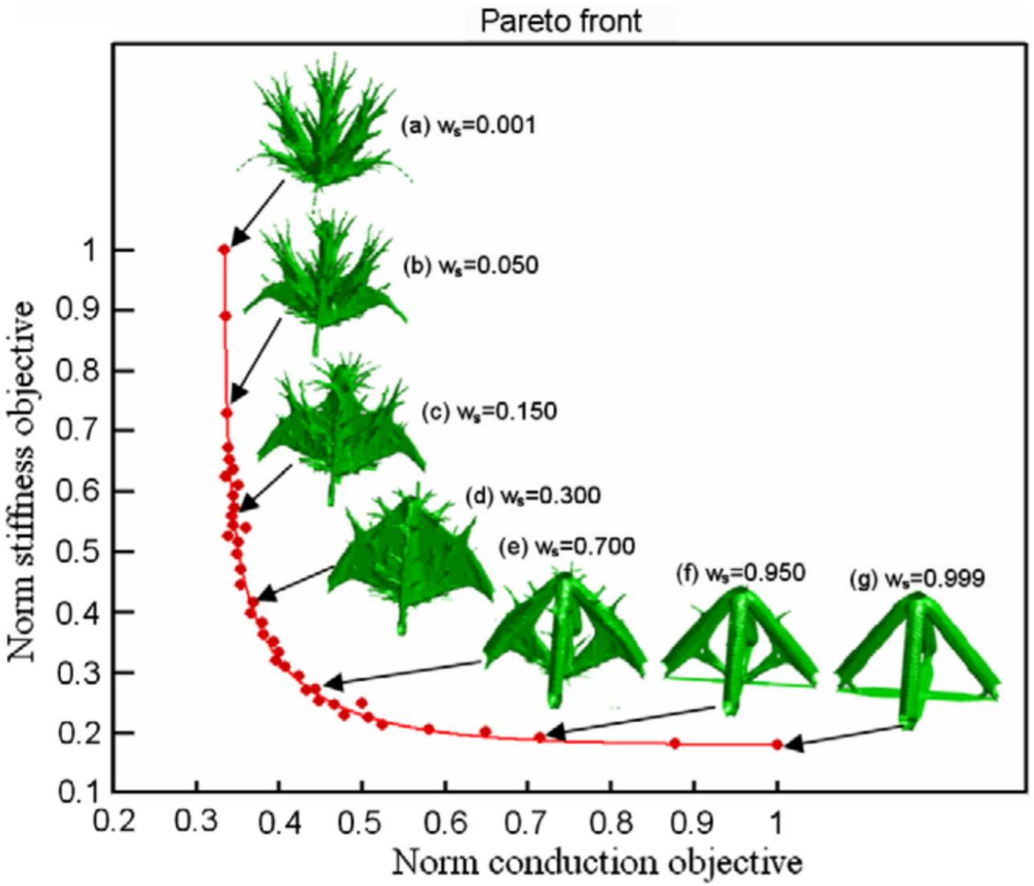
- Find 'Pareto perfect' structures:
  - Micro-
  - Macro-
- Efficiency
- Precision & Coverage

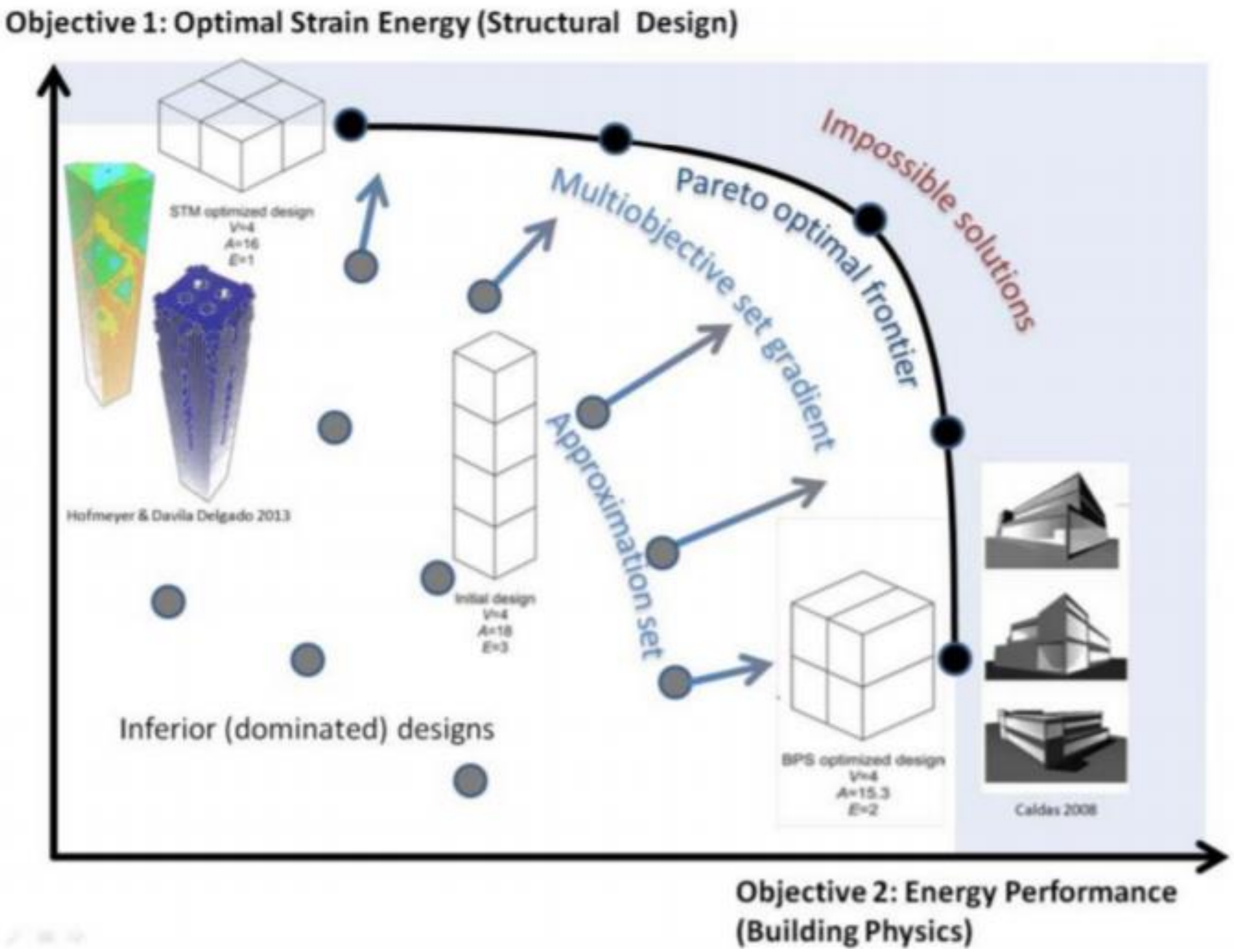


Niek de Kruijf, Shiwei Zhou, Qing Li, Yiu-Wing Mai, Topological design of structures and composite materials with multiobjectives, International Journal of Solids and Structures, Volume 44, Issues 22-23, 2007, Pages 7092-7109

‘Pareto morphing’ along the Pareto front

Chen, Yuhang, Shiwei Zhou, and Qing Li. "Multiobjective topology optimization for finite periodic structures." *Computers & Structures* 88.11-12 (2010): 806-811.




Objective 1: Optimal Strain Energy (Structural Design)
STM optimized design
V=4
A=16
E=1
Hofmeyer & Davila Delgado 2013
Multiobjective set gradient
Pareto optimal frontier
Impossible solutions
Approximation set
Initial design
V=4
A=18
E=3
Inferior (dominated) designs
BPS optimized design
V=4
A=15.3
E=2
Caldas 2008
Objective 2: Energy Performance
(Building Physics)

Question

- How can we express shapes by means of decision variables?
- How can we make sure that constraints are kept?

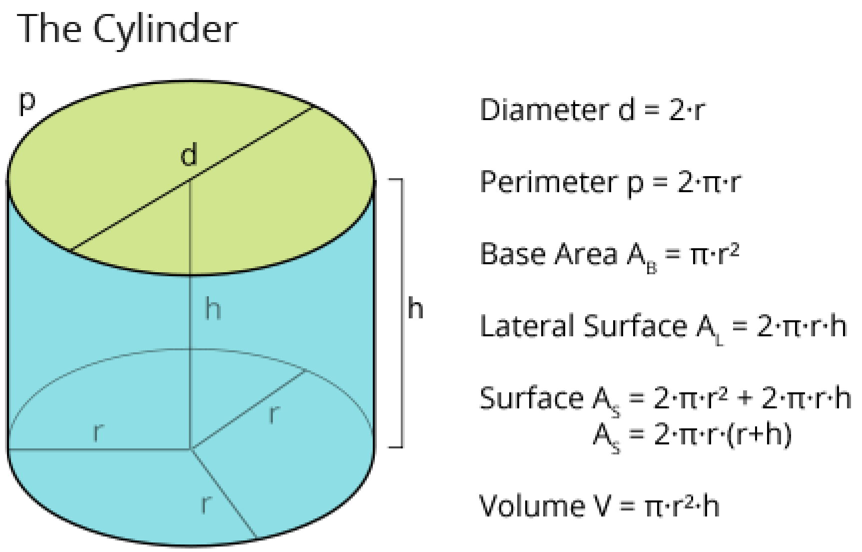
Example: Cylinder (tin)

Volume → max

Surface → min

Parameters: r, h

Constraints: r > 0, h > 0



Solution

$$V(r,h) = \pi r^2 h = \epsilon \Rightarrow h = \frac{\epsilon}{\pi r^2}$$

$$A(r,h) = 2\pi r^2 + 2\pi r h$$

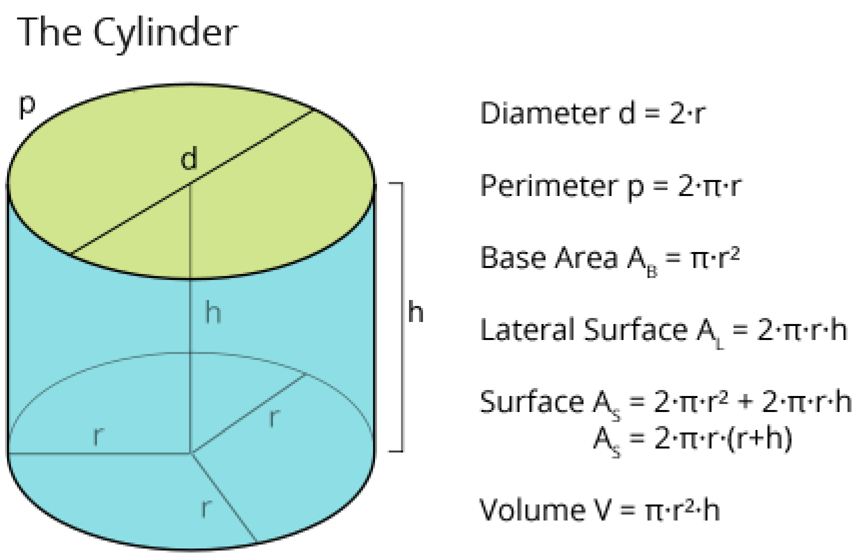
$$= 2\pi r^2 + \frac{2\pi r}{\pi r^2} = 2\pi r^2 + \frac{2}{r} \rightarrow min$$

$$\nabla A(r) = \frac{dA(r)}{dr} = 4\pi r - \frac{2\epsilon}{r^2} = 0$$

$$\Leftrightarrow r^3 = \frac{2\epsilon}{4\pi} \Leftrightarrow r = \sqrt[3]{\frac{\epsilon}{2\pi}}$$

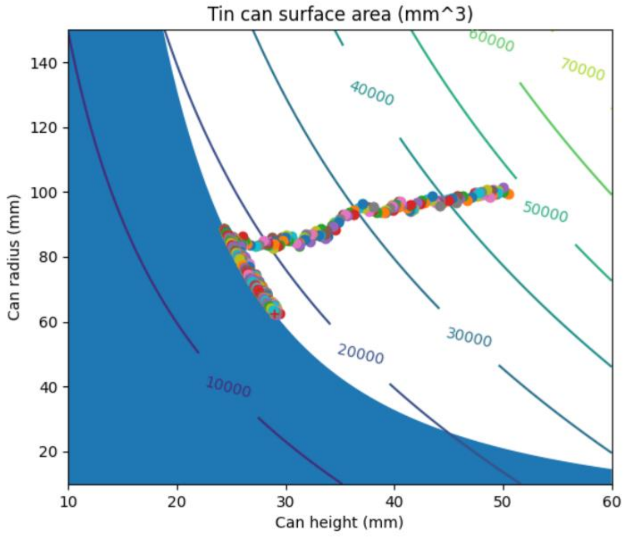
Efficient Set:

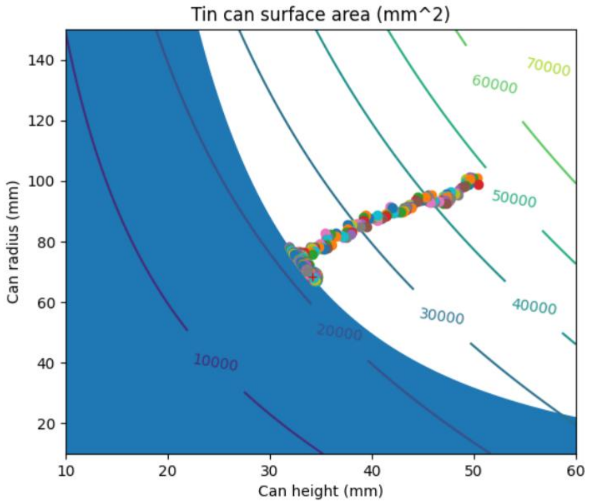
$$\mathrm{X_e} = \{(\sqrt[3]{\frac{\epsilon}{2\pi}}, \frac{\epsilon}{\pi r^2}) | \epsilon \in [0, \infty]\}$$



Computation of Pareto front with local search method

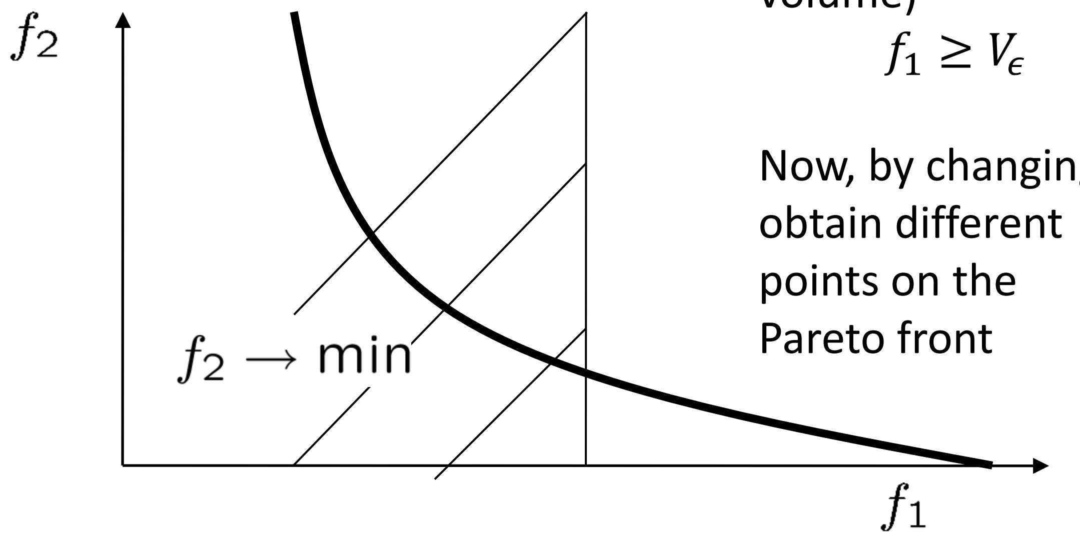
- Set the target volume to different levels
- Minimize the surface area for each level
- Here: Local Search method is used
- Idea: Random variation of a point, select if improvement
- Add penalty in case of constraint violation





$f_1 \rightarrow max$ (e.g. volume)

$f_1 \geq V_\epsilon$



Now, by changing V_ϵ obtain different points on the Pareto front

Basic strategy in Black-box optimization

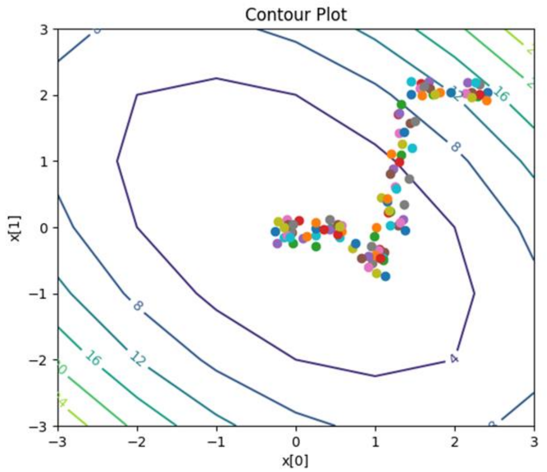
Black-box Optimierungs-software

Input Parameters x(t)

Simulator/Evaluator

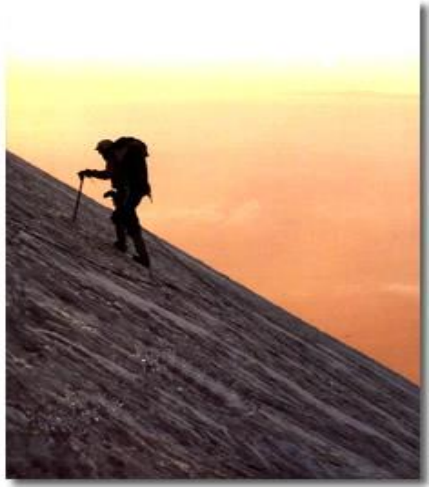
Zielfunktionswerte, Restriktionsverletzungen
f(x(t))+penalty(r(x(t))

1. Stochastic Hillclimbing
2. Gradient Descent
3. Newton Method
4. Simulated Annealing
5. Evolutionary Algorithm
6. Bayesian Optimization
7. Etc.



Hill-climbing Methods for Single-Objective Optimization

Path oriented (hill climbers) can be defined by a general iterative formula:



$$\mathbf{x_{t+1}} = \mathbf{x_t} + \sigma_\mathbf{t} \mathbf{d_t}$$

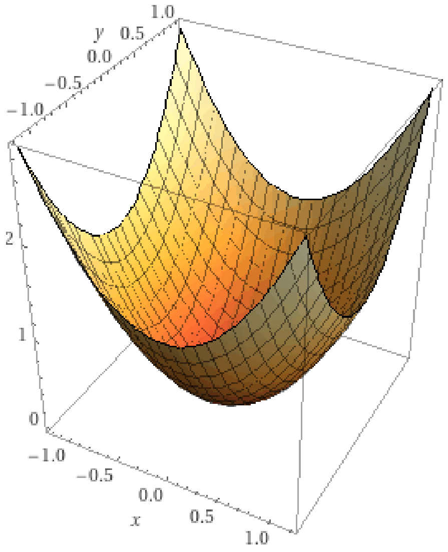
$\mathbf{x_t}$: Current search point

σ_t: Step size

$\mathbf{d}_t$: Current search direction

Hill-climbers generates a sequence of points $\{\mathbf{x}_t\}_{t=1,2,...}$ that gradually improve the value of the objective function.

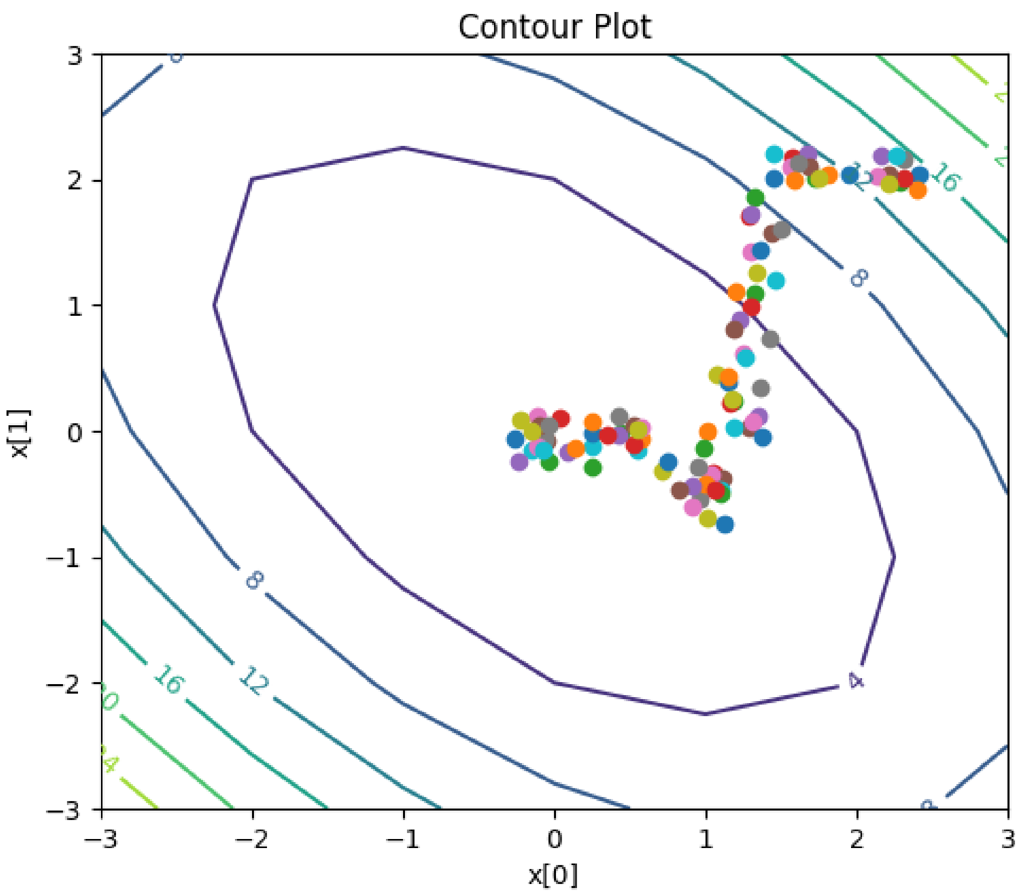
Simple 2-D stochastic hillclimber (Example of a single-objective optimizer, minimalistic)




```
# objective function
def objective(x):
    return x[0]**2+x[1]**2
```

```
#  black-box optimization software
def local_hillclimber(objective, bounds, n_iterations, step_size,init):
    # generate an initial point
    best = init
    # evaluate the initial point
    best_eval = objective(best)
    curr, curr_eval = best, best_eval   # current working solution
    scores = list()
    for i in range(n_iterations):
        # take a step
        candidate = [curr[0] +rand()*step_size[0]-step_size[0]/2.0,
                     curr[1]+rand()*step_size[1]-step_size[1]/2.0]
        print('>%d f(%s) = %.5f, %s' % (i, best, best_eval,candidate))
        #+ randn(len(bounds)) * step_size
        # evaluate candidate point
        candidate_eval = objective(candidate)
        # check for new best solution
        if candidate_eval < best_eval:
            # store new best point
            best, best_eval = candidate, candidate_eval
            # keep track of scores
            scores.append(best_eval)
            # report progress
            print('>%d f(%s) = %.5f' % (i, best, best_eval))
            # current best
            curr=candidate
    return [best, best_eval, scores]
```

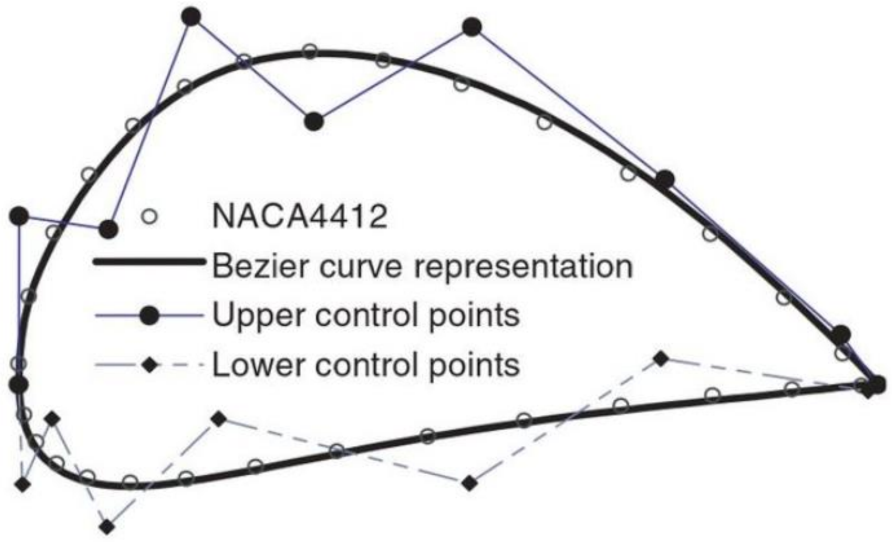
Plotting the history




```
bounds=asarray([[-3.0,3.0],[-3.0,3.0]])
step_size=[0.4,0.4]
n_iterations=100
init=[2.4,2.0]
best, score, points, scores,  = local_hillclimber(objective,
                                                 bounds, n_iterations,
                                                 step_size, init)


n, m = 7, 7
start = -3

x_vals = np.arange(start, start+n, 1)
y_vals = np.arange(start, start+m, 1)
X, Y = np.meshgrid(x_vals, y_vals)

print(X)
print(Y)
fig = plt.figure(figsize=(6,5))
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes([left, bottom, width, height])


Z = (X**2 + Y**2 + X*Y)
cp = ax.contour(X, Y, Z)
ax.clabel(cp, inline=True,
          fontsize=10)
ax.set_title('Contour Plot')
ax.set_xlabel('x[0]')
ax.set_ylabel('x[1]')
for i in range(n_iterations):
    plt.plot(points[i][0],points[i][1],"o")
plt.show()
```

Design Parameterization

- Design Parameterization is the problem of describing a geometrical shape by means of continuous parameter vectors
- Examples: Bezier points, superstructures



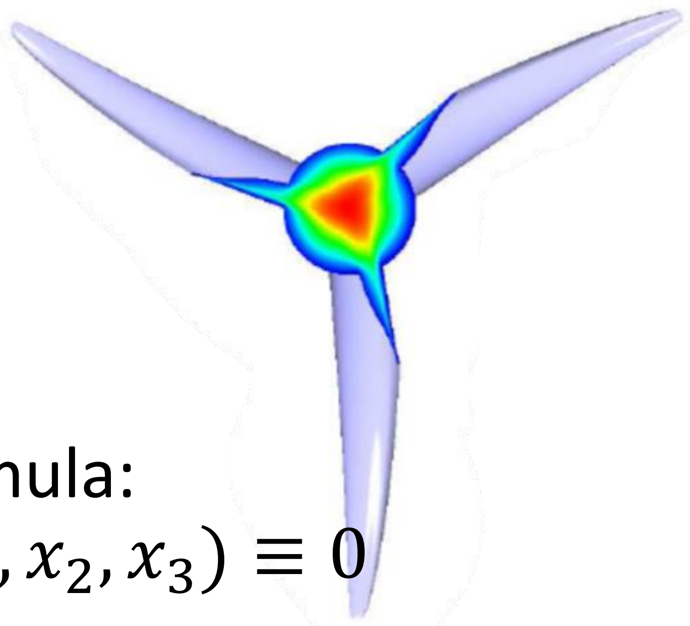
Figure

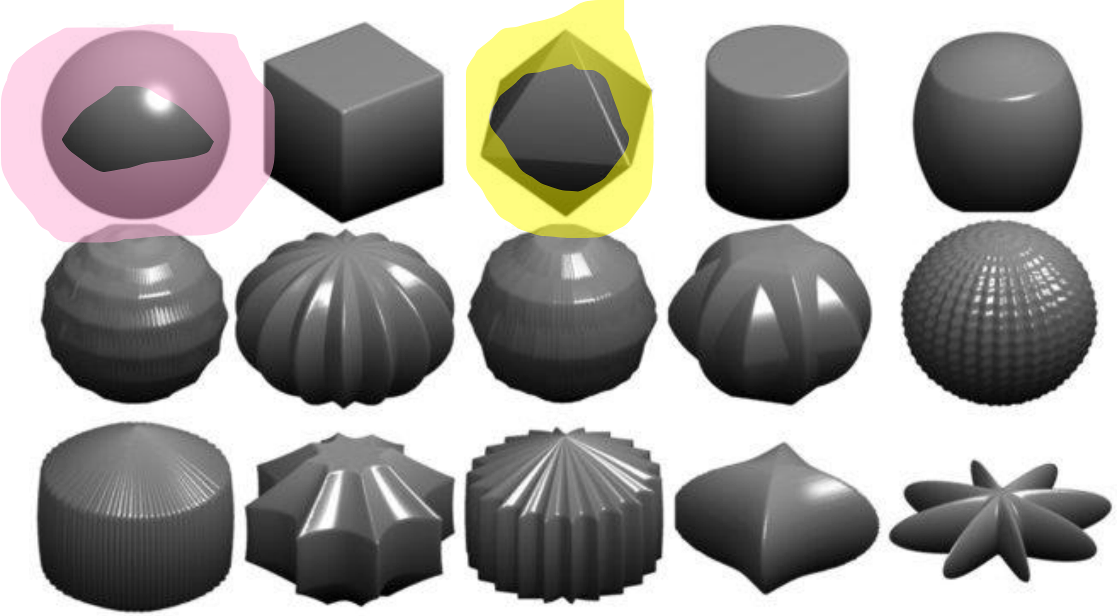
Caption

Fig. 1.8: Bezier curves representation of NACA4412 airfoil [36]

Source: https://arxiv.org/pdf/1204.4107.pdf

Parameterization
Gielis’ superformula





$$r(\varphi) = \left(\left| \frac{\cos\left(\frac{m\varphi}{4}\right)}{a} \right|^{n_2} + \left| \frac{\sin\left(\frac{m\varphi}{4}\right)}{b} \right|^{n_3} \right)^{-\frac{1}{n_1}} .$$

- The superformula: $f(p_1, \dots, p_n, x_1, x_2, x_3) \equiv 0$
- $x_1, \dots x_n$: variables
- $p_1, \dots, p_n$: parameters
- Example

$$f(x_1, x_2, x_3) = \sum |x_i|^{p_i}$$

$$p_i = 1, i = 1, \dots, 3$$

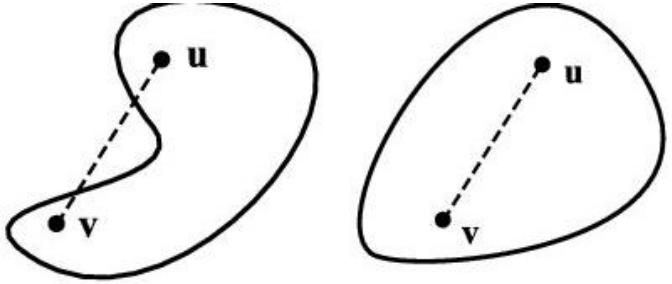
$$p_i = 2, i = 1, \dots, 3$$

Convexity

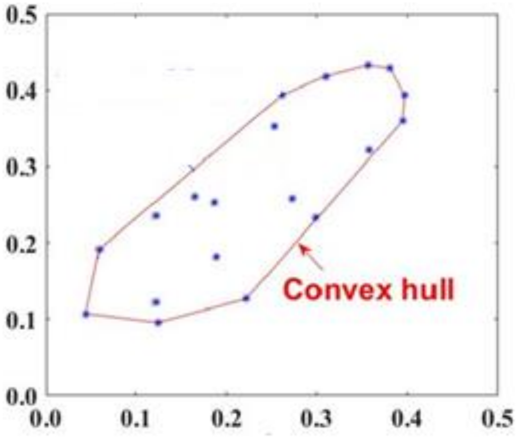
Idea
Represent buildings as convex shapes

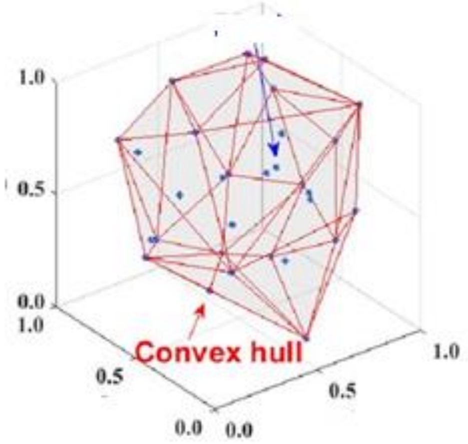
Source: Wikipedia, Creative commons license

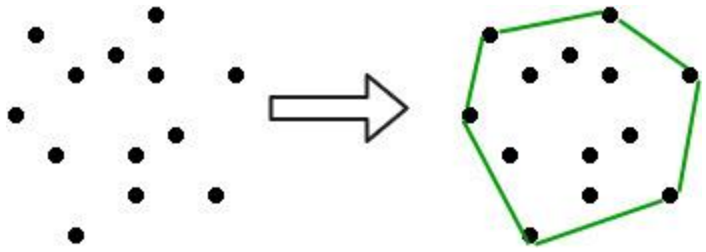
- A set $S \subseteq \mathbb{R}^d$ is called a convex set if it for any two points, $u \in S$ and $\mathrm{v} \in S$ the line-segment connecting u and v is fully contained in S.
- These shapes have no-cavities (this is why buildings are often convex, excepting those with 'swimming pools' on the roof)



Convex hull representation

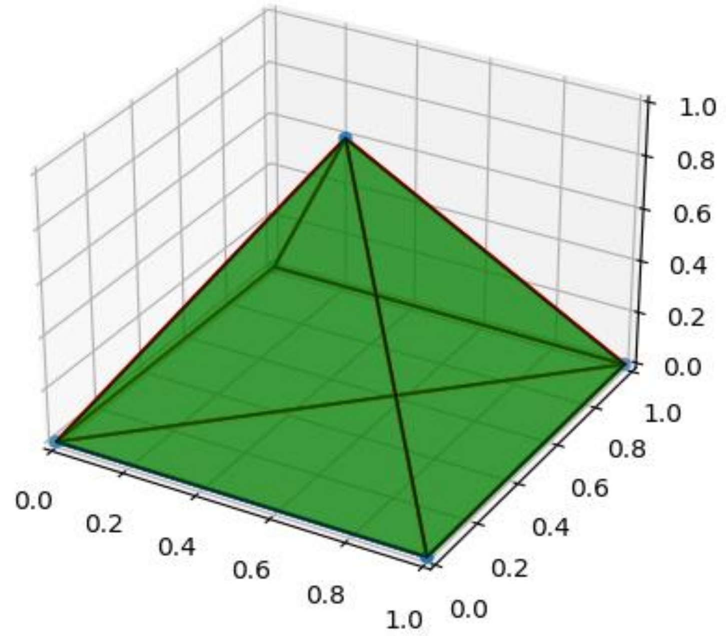


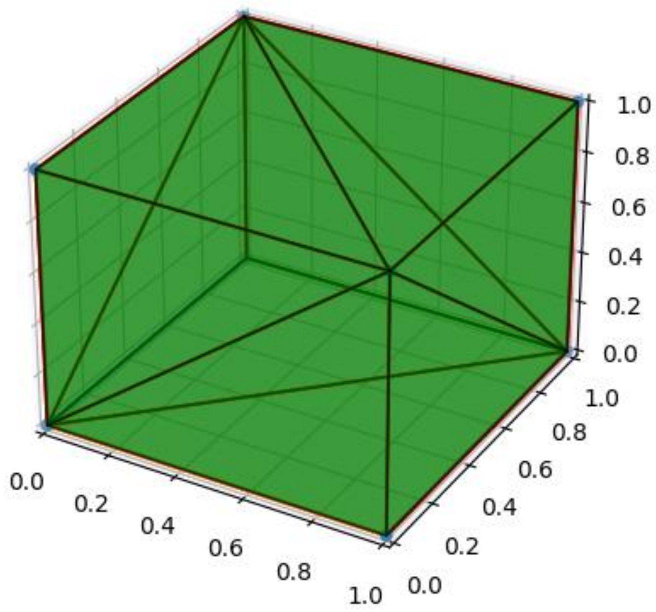




- The convex hull is the smallest convex set that contains all points
- In 2-D you can imagine a rubber-band around the set
- Convex hulls can be used to represent the set of all convex shapes by means of point sets
- ‘Active’ points form corners of this sets. Inactive points are redundant and can be removed.

Design optimization – 3D Shapes



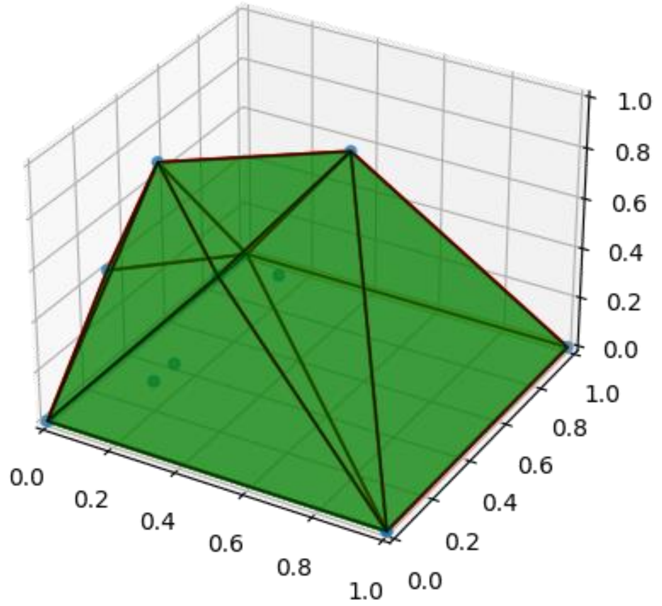



```
# Pyramid example
print("Making a pyramid")
# Define the points first
pyramid_points = np.array([
    [0,0,0], [1,0,0], [0,1,0], [1,1,0], # floor corners
    [.5, .5, 1] # pyramidion / capstoneS
])
```

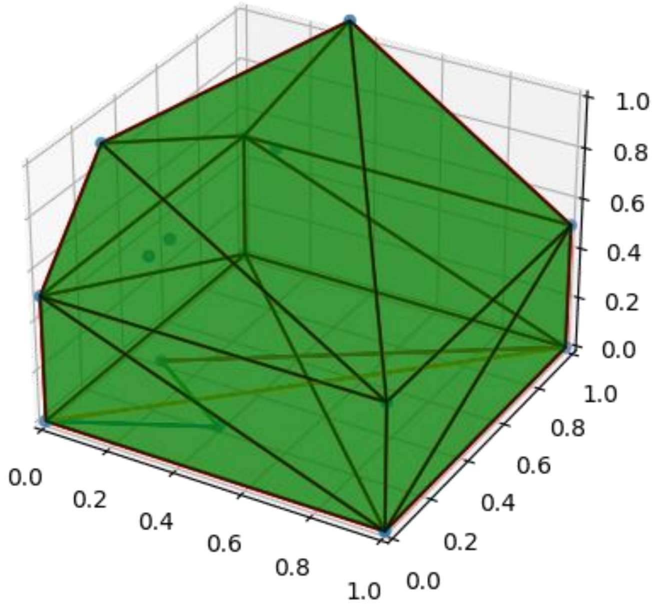
```
# Box example:
print("Making a box")
# Define the points
box_points = np.array([
    [0,0,0], [1,0,0], [0,1,0], [1,1,0], # floor corners
    [0,0,1], [1,0,1], [0,1,1], [1,1,1] # Ceiling/roof corners
])
```

Walls, floors, and roofs

```
random_points = np.array([
    [0,0,0], [0,1,0],[1,1,0], [1,0,0], # floor
    [0, 0.3, 0.4+0.5], [0.4, 0.2, 0.6+0.5], [0.1, 0.4, 0.8],
    [0.4, 0.5, 0.4], [.5, 0.7, 0.8],
    [.2, 0.3, 0.1], [.2, 0.2, 0.1]
])
box3 = Tent(random_points)
box3.plot()
```

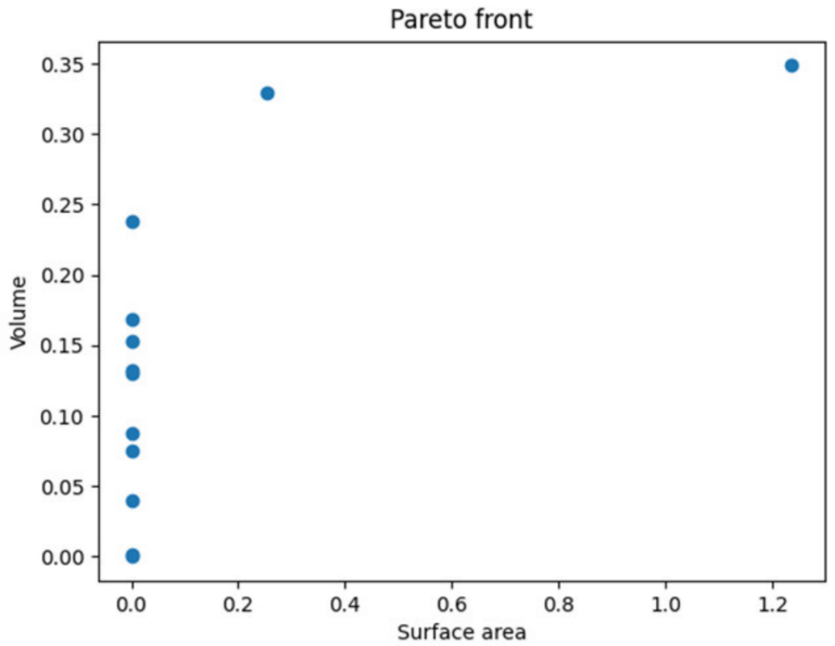


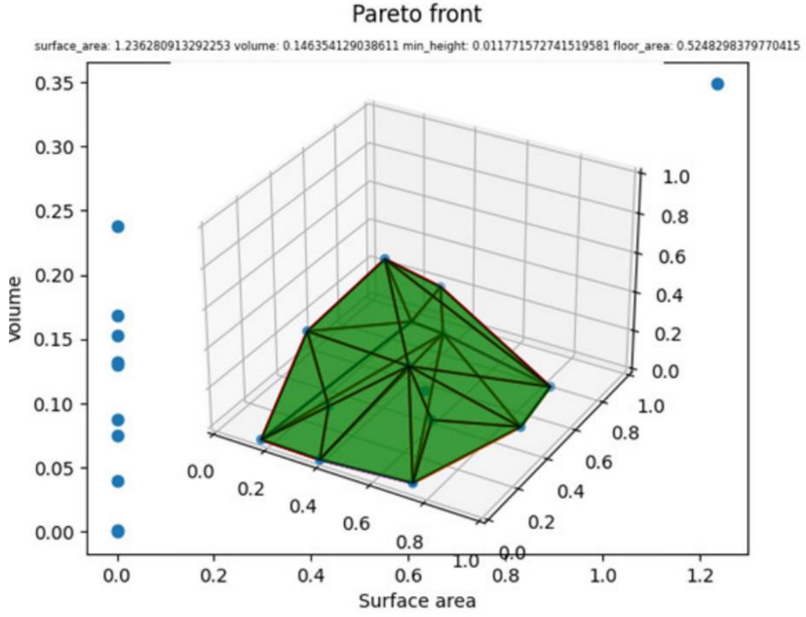

```
random_points = np.array([
    [0,0,0], [0,1,0],[1,1,0], [1,0,0], # floor
    [0, 0, 0.5], [0, 1, 0.5], [1, 1, 0.5],[1, 0, 0.5],  # floor+0.5
    [0, 0.3, 0.4+0.5], [0.4, 0.2, 0.6+0.5], [0.1, 0.4, 0.8+0.5],
    [0.4, 0.5, 0.4+0.5], [.5, 0.7, 0.8+0.5],
    [.2, 0.3, 0.1+0.5], [.2, 0.2, 0.1+0.5]
])
box3 = Tent(random_points)
box3.plot()
```

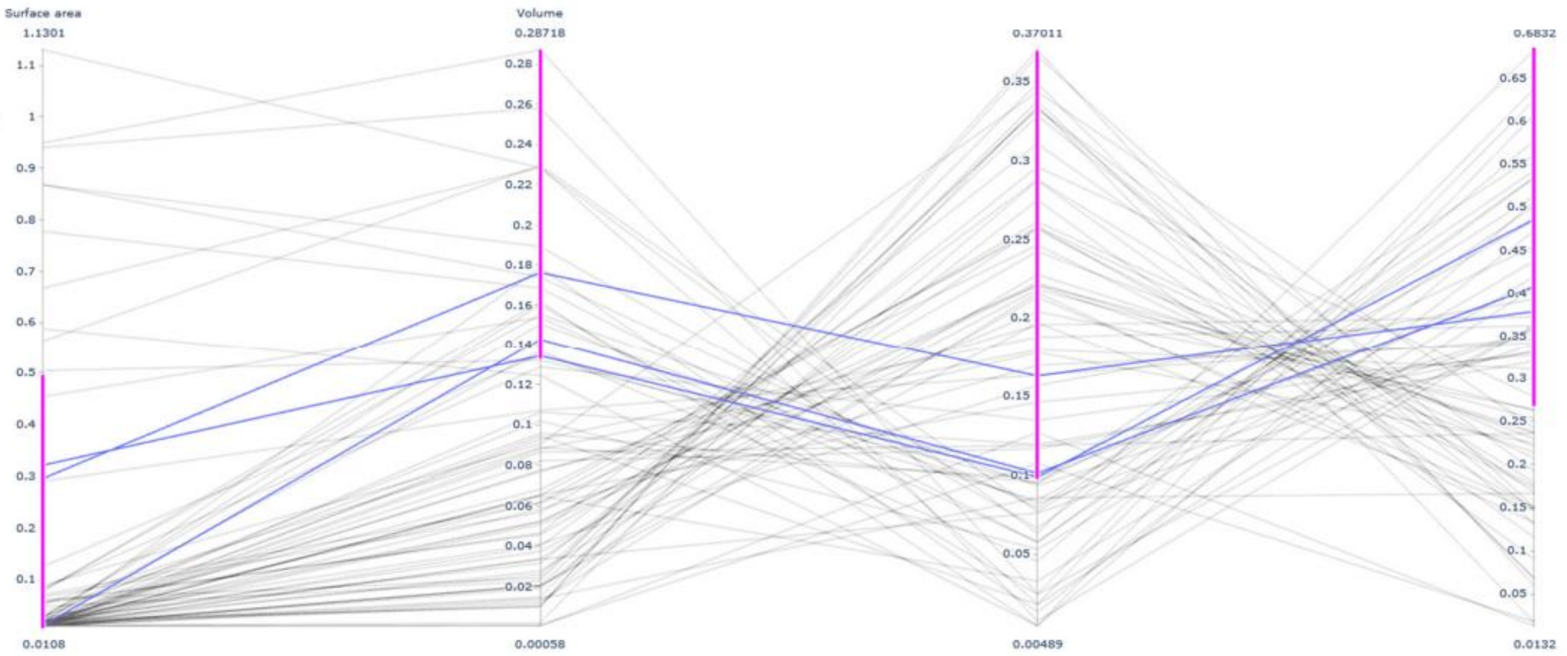


Optimizing Tents (homework)

- Maximize Volume
- Minimize Surface Area of Roof
- Maximize Height
- Maximize Floor Area







To be continued …