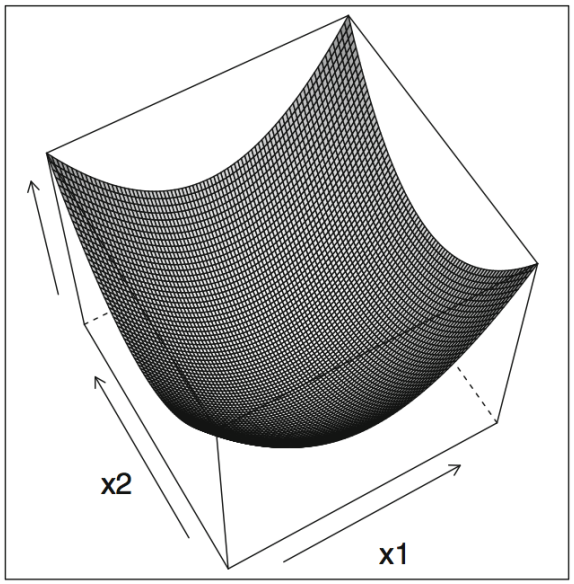
# Otimização

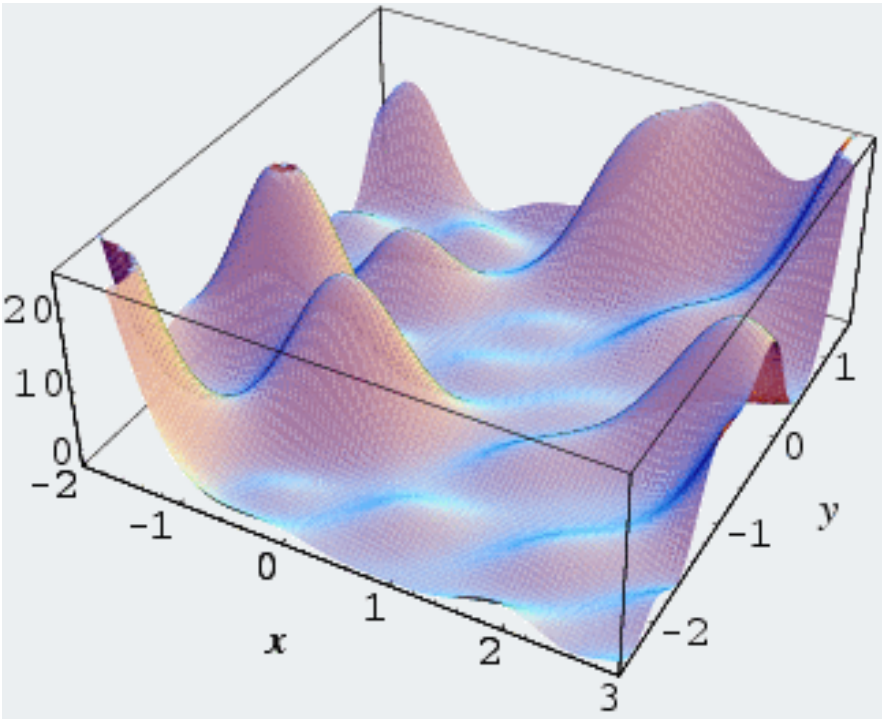
**OBJECTIVOS:**

- Introdução à Resolução de Problemas que envolvam uma otimização

# Otimização

•pretende-se “otimizar” (minimizar ou maximizar) um ou mais **objetivos**. Cada solução válida está associada a um ou mais objetivos, existindo soluções óptimas (locais ou globais) e soluções de qualidade diversa (interessantes, razoáveis, fracas, etc...)



## Objetivos:

- Assume-se que cada transição de estado implica uma determinada quantidade ou quantidades (custo, benefício, gasto, distância, ...).
- Dependendo do tipo de quantidade, o objetivo é **maximizar** ou **minimizar** algo.

Exemplos:

-Custo unitário para cada movimento de um bloco no problema das caixas (Blocks) – objetivo: minimizar movimentos.

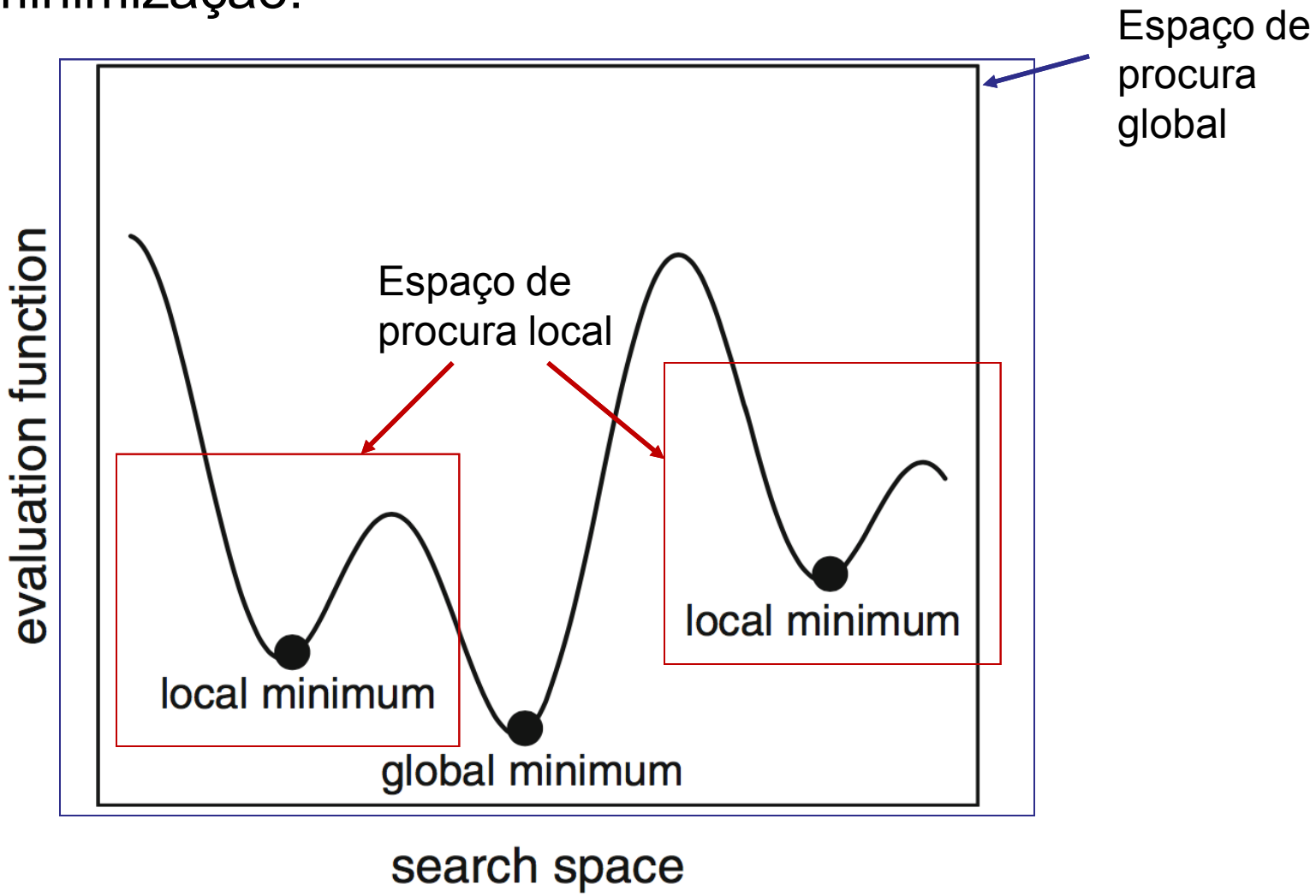
-Encontrar caminho entre 2 cidades da Roménia – objetivo: minimizar distância, minimizar custo de combustível, maximizar o valor turístico da viagem, ...

-Elaborar horários, minimizando a ocupação de salas e maximizando a compactação de aulas (evitando furos).

-Sentar convidados num casamento, maximizando relações boas e minimizando conflitos.

## Objetivos:

- Procura/ótimo local versus procura/ótimo global, exemplo de minimização:



# Exemplo: Problema max sin


Seja $x_i$ uma sequencia de números binários (0 ou 1) com comprimento de D. Pretende-se maximizar o valor de f, de acordo com:

$$x' = \sum_{i=1}^{D} x_i 2^{i-1}$$

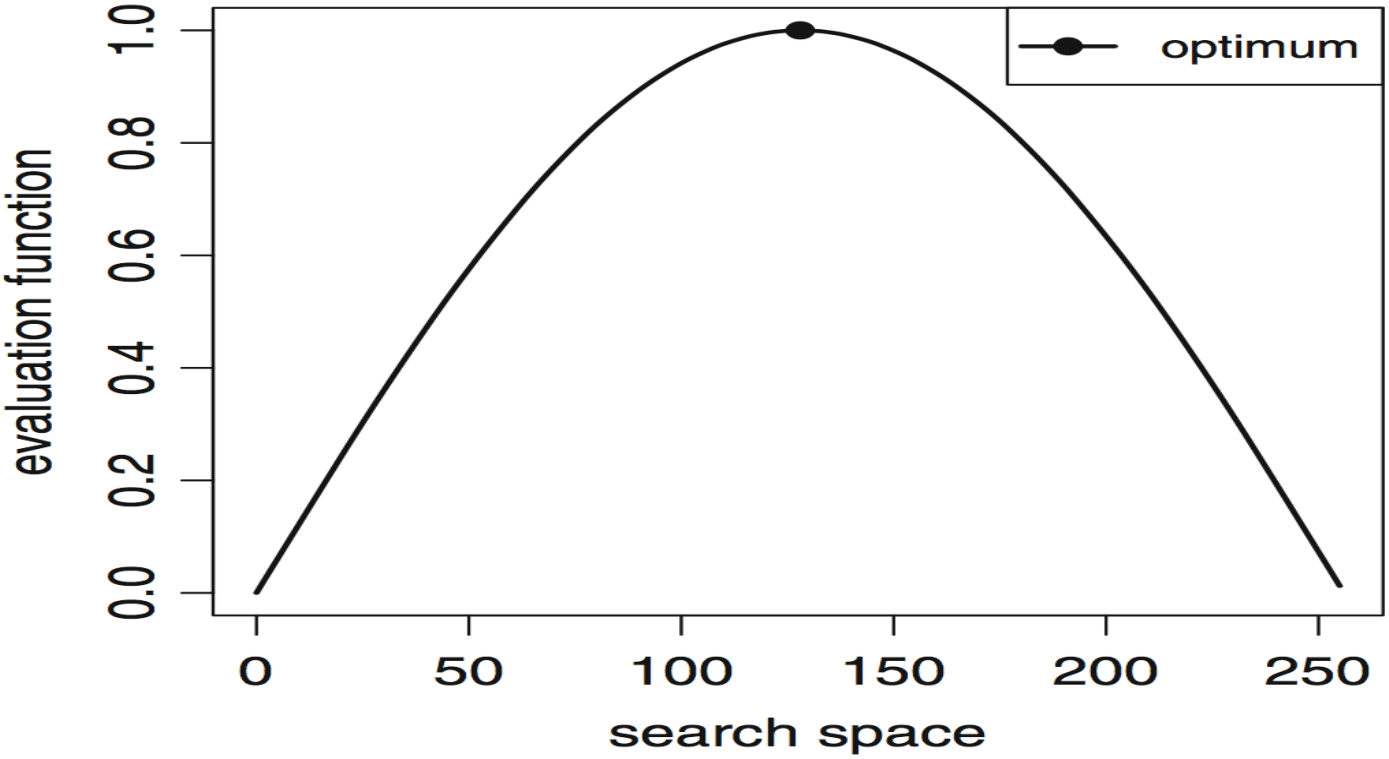
$$f_{\max \sin}(\mathbf{x}) = \sin\left(\pi \frac{x'}{2^D}\right)$$

Onde x' é a representação inteira de $x_i$ (por exmplo: para $x_i$=[0,1,1,1], D=4 e x'=7.

**Q1: Qual sequência binária ótima para D=4?**

# Exemplo: Problema max sin

Para D=8, se eixo dos x for x', tem-se:




O valor ótimo é: xi=[1,0,0,0,0,0,0,0], x'=128, f(x')=1.0.

## Métodos/Algoritmos de Otimização:

- Baseados em **Procura Cega**: depth-first, breadth-first, Monte-Carlo, ... (analisados na aula passada)
- Baseados em **Procura e Heurísticas**: best-first search, A*, IDA*, RBFS
- **Iterativos**:
  - baseados numa **única solução** a melhorar (procura local): Hill-climbing, Stochastic Hill-climbing, Simulated Annealing, Tabu Search, ...
  - baseados **numa população de soluções** (procura global): Evolutionary Computation (Genetic and Evolutionary Algorithms), Differential Evolution, Particle Swarm Optimization, Estimation of Distribution Algorithm, ...

# Métodos Iterativos

- Também conhecidos por **Otimização Moderna** ou **Metaheurísticas**.
- Parte-se de **uma solução ou população** de soluções iniciais.
- Cada solução **S** tem uma dada forma de representação (conjunto de números, lista de estados, árvore,
- Para cada solução, é possível medir a qualidade (objetivo) via uma função de **avaliação/fitness/eval**: f(**S**).
- A pesquisa **é iterativa**, via uma procura guiada, onde novas soluções são criadas a partir da solução ou população atual.

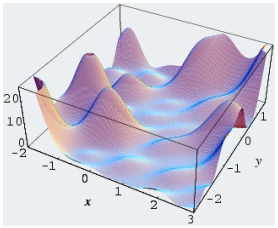
# Métodos Iterativos de Otimização Moderna

**Algorithm 1** Generic modern optimization method

1: **Inputs:** $f, C$ ▷ $f$ is the evaluation function, $C$ includes control parameters
2: $S \leftarrow initialization(C)$ ▷ $S$ is a solution or population
3: $i \leftarrow 0$ ▷ $i$ is the number of iterations of the method
4: **while** not $termination\_criteria(S, f, C, i)$ **do**
5: $S' \leftarrow change(S, f, C, i)$ ▷ new solution or population
6: $B \leftarrow best(S, S', f, C, i)$ ▷ store the best solution
7: $S \leftarrow select(S, S', f, C, i)$ ▷ solution or population for next iteration
8: $i \leftarrow i + 1$
9: **end while**
10: **Output:** $B$ ▷ the best solution

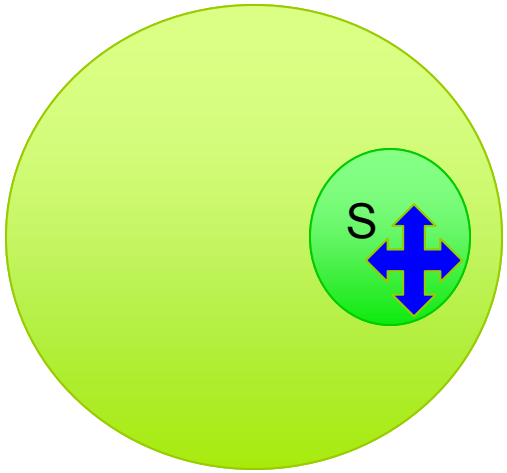
# Métodos Iterativos

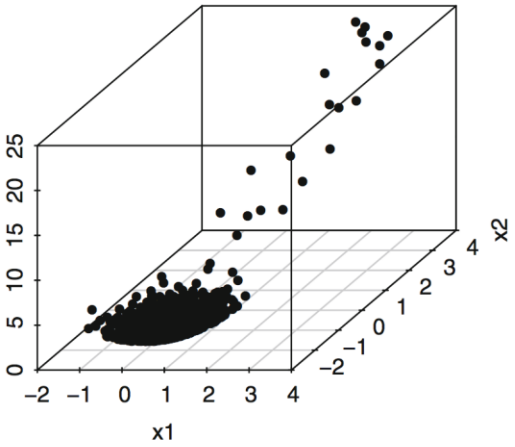
- **Single-state**: procura sobretudo local, partem de uma solução inicial que é melhorada
- **Population-based:** procura mais global, partem de uma população de soluções.
- **Vantagens**: **métodos genéricos**, usam pouco conhecimento do domínio (tipicamente de modo indireto, via a função de avaliação), facilmente aplicáveis a qualquer problema, tendem a encontrar boas soluções (ou de qualidade) com um uso razoável de esforço computacional.
- **Desvantagens**: não garantem que a solução ótima (global) é encontrada.



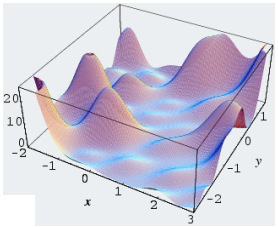
# Hill Climbing

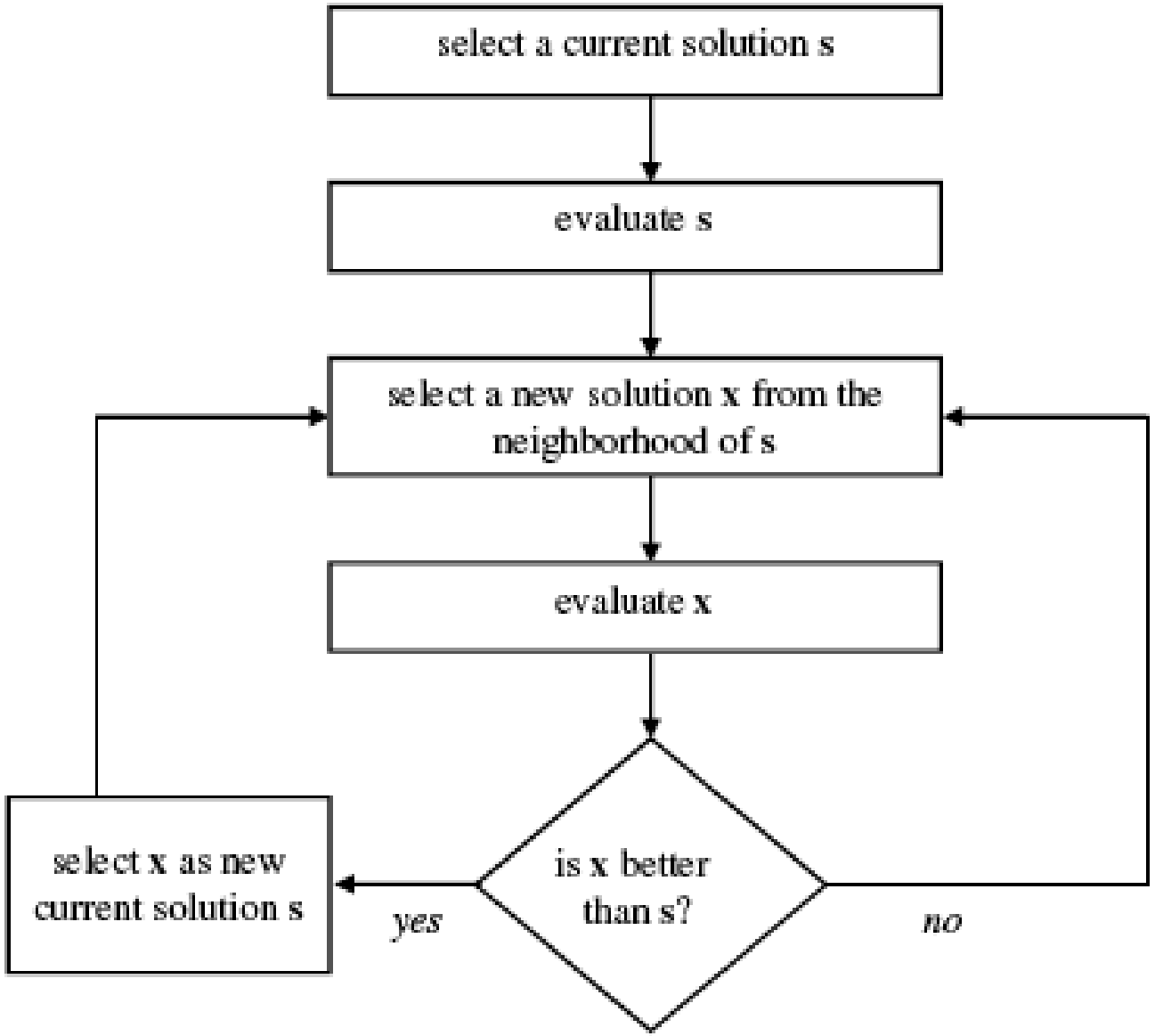
- **performs local search,** searches from a given initial point (S)
- **+** easy to implement;
- **- may get stuck in local minima/maxima;**
- **Algorithms/Variants**:
  - Gradient descent;
  - Stocastic Hill-climbing;
  - Simulated Annealing;
  - Tabu search
  - …



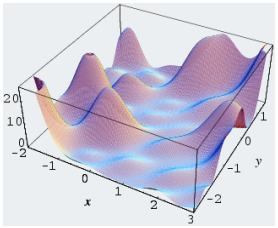



# Simple Hill Climbing





# Simple Hill Climbing



**Algorithm 2** Pure hill climbing optimization method

1: **Inputs:** $S, f, C$ ▷ $S$ is the initial solution, $f$ is the evaluation function, $C$ includes control parameters
2: $i \leftarrow 0$ ▷ $i$ is the number of iterations of the method
3: **while** not $termination\_criteria(S, f, C, i)$ **do**
4: $S' \leftarrow change(S, C)$ ▷ new solution
5: $B \leftarrow best(S, S', f)$ ▷ best solution for next iteration
6: $S \leftarrow B$ ▷ deterministic select function
7: $i \leftarrow i + 1$
8: **end while**
9: **Output:** $B$ ▷ the best solution

# Complexidade Algorítmica

- O método de hill climbing tem uma complexidade proporcional ao número de iterações: **O(N)** x O(Eval(S))

Valor em geral constante

- Não garante a obtenção do valor óptimo.
- O pure hill climbing pode ficar preso em mínimos locais.
- Variantes: **Stochastic Hill Climbing**, com o parâmetro Prob, onde a nova solução é aceite se **Prob<*random*(0,1)** ou se é melhor do que a solução atual.
- O Stochastic Hill Climbing é menos propenso a ficar preso em mínimos locais.

## Criação de um SBC para Optimização via Hill Climbing

1. Pensar numa forma Prolog para representação de uma solução **S**.
2. Definir uma função de avaliação da qualidade da solução: **eval(S,Q)**
3. Definir uma função que com base numa solução atual cria outra na sua vizinhança local: **change(S1,S2)**

## Representação de uma solução S

- A ideia é com base em **S** conseguir ter uma solução que faça sentido para o problema a analisar.
- Diversas possibilidades: números binários, inteiros, reais, mistura de binários com inteiros e reais, árvores, gramáticas, ....

# Representação de uma solução S

- Representação binária com uma dimensão de D bits

$$\underbrace{B_1\ B_2\ B_3\ B_4\ \ldots\ B_D}_{\text{Total de D bits.}}$$

onde Bi é 0 ou 1.

Exemplo:

- Escolher uma equipa qualquer de seres humanos para um dado projeto, de entre a Ana, Carlos, Iva e Rui (D=4),exemplos de soluções:
  - 1,0,1,0 (Ana e Iva)
  - 1,0,1,1 (Ana, Iva e Rui)
  - 0,1,1,1 (Carlos, Iva e Rui)

# Representação de uma solução S

- Representação com inteiros e dimensão D

$$\underbrace{N_1\ N_2\ N_3\ N_4\ \ldots\ N_D}_{\text{Total de D inteiros.}}$$

onde Ni=0,1,2,3,...

Exemplo:

- Escolher o preço (em euros) para D=5 produtos, preço entre 1 e 100:
  - 5,10,83,99,4
  - 100,1,90,45,34

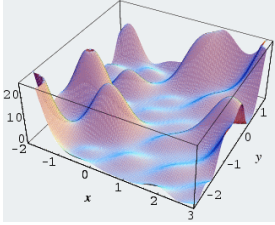
# Função de Avaliação da Solução eval(S)

- Função que devolve um valor numérico com a qualidade da solução **S**, algo a minimizar ou maximimizar.
- Em muitos problemas, tem-se:

$$\text{eval}(S) = \text{objetivo}(S) - \text{penalização}(S)$$

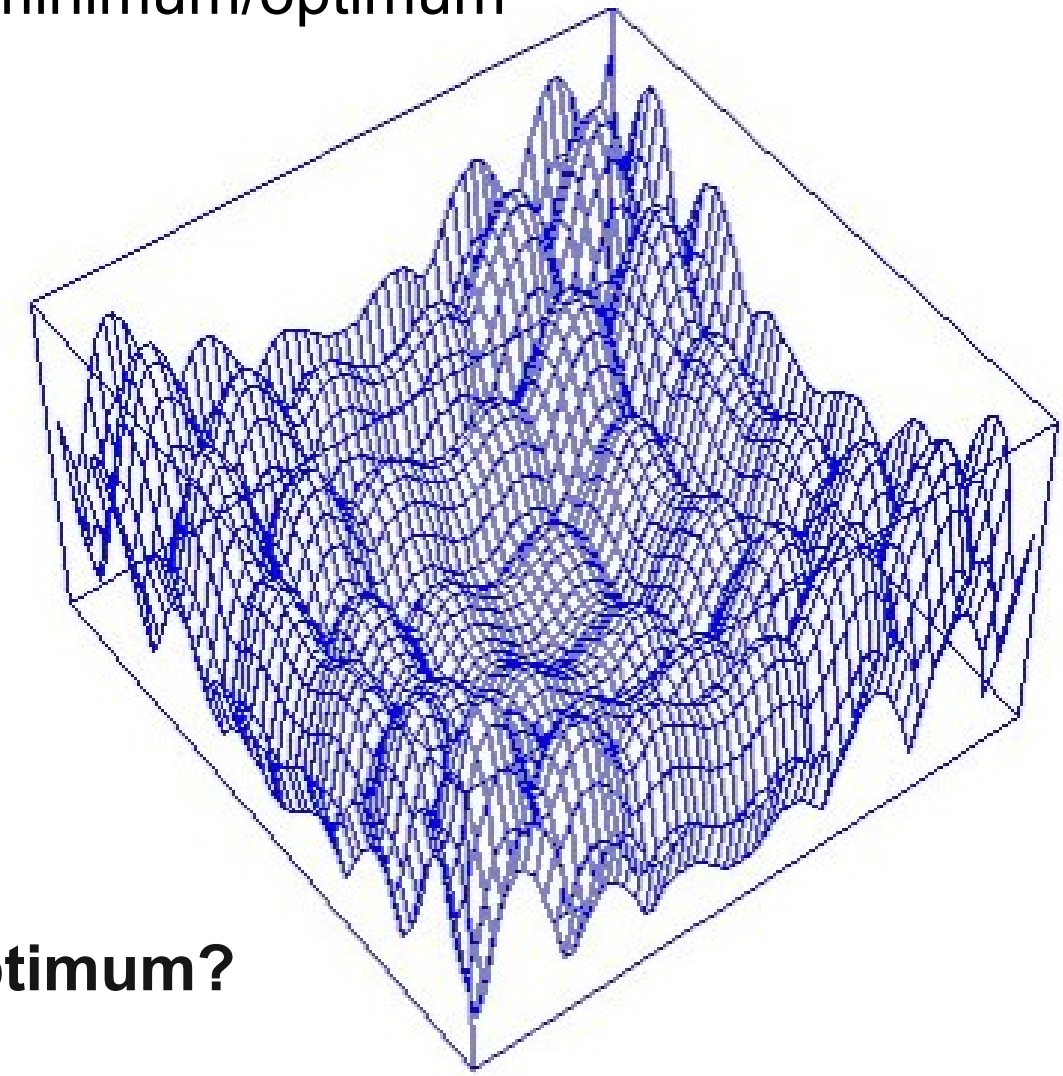
- Exemplo:

$$\text{eval}(S) = \text{lucro}(S) = \text{ganho}(S) - \text{custo}(S)$$



# > Evaluation function

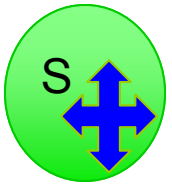
- Local vs Global minimum/optimum



- **What is the optimum?**

# Soluções inválidas (restrições)

- Por vezes uma solução pode não ser válida. Para lidar com estas soluções existem 2 possibilidades (entre outras):
  - **Death Penalty:** atribuir uma avaliação proibitiva (demasiado má).
  - **Reparar:** transformar uma solução inválida em válida (**melhor**)
- Exemplo: Escolher visitar até 3 cidades, de entre um total de 5. Se a representação for binária (0 – não visita, 1 visita), podemos ter estas soluções
  - 1,0,0,0,0 -> solução válida
  - 1,0,1,1,0 -> solução válida.
  - 1,0,1,1,1 -> solução inválida. O que fazer?
    - eval(1,0,1,1,1) = **1.0Inf** (“infinito”, se objetivo é minimizar).
    - Uma estratégia de reparação: selecionar as 3 primeiras cidades a contar da esquerda para a direita:
      **1,0,1,1**,1 -> **reparar** -> **1,0,1,1**,0




# Vizinhança local: change(S1,S2)

- Função que com base em S1, cria uma nova solução S2 que está na ”vizinhança” de S1.
- Função determinística (alteração sempre igual) ou estocástica (perturbação aleatória de um parâmetro de S1).

Exemplos estocásticos:

- Solução binária S1: escolher um bit de modo aleatório e inverter o seu valor.
- Solução de inteiros S1: escolher um inteiro de modo aleatório e com 50% de probabilidade aumentar ou diminuir esse inteiro em 1 unidade.
- Solução de reais S1: escolher um valor real de modo aleatório e perturbá-lo, adicionando um valor aleatório.

# Hill Climbing

```
% hill climbing
% Prob=0 is pure hill climbing, Prob>0 means Stochastic Hill Climbing
% S0 is the initial solution, Control is a list with the number of iterations,
verbose in console, probability and type of optimization.
hill_climbing(S0,[Iter,Verbose,Prob,Opt],S1):-
        eval(S0,E0),
    show(0,Verbose,S0,E0,_,_),
        hill_climbing(S0,E0,0,Iter,Verbose,Prob,Opt,S1).

hill_climbing(S,_,Iter,Iter,_,_,_,S).
hill_climbing(S1,E1,I,Iter,Verbose,Prob,Opt,SFinal):-
        change(S1,SNew),
        best(Prob,Opt,S1,E1,SNew,_,S2,E2),
        I1 is I+1,
    show(I1,Verbose,S1,E1,S2,E2),
        hill_climbing(S2,E2,I1,Iter,Verbose,Prob,Opt,SFinal).
```

# Hill Climbing

```
demo:- hill_climbing([a,b,c],[20,1,0.5,max],S),write(S).

% assumes eval(Solution,Result)
% assumes change(S1,S2)

% return SR, the best value of S1 and S2: SR (solution) and ER (eval)
best(Prob,Opt,S1,E1,S2,E2,SR,ER):-
                eval(S2,E2),
                best_opt(Prob,Opt,S1,E1,S2,E2,SR,ER).

best_opt(Prob,_,_,_,S2,E2,S2,E2):-
                random(X), % random from 0 to 1,
                X< Prob. % accept new solution

best_opt(_,Opt,S1,E1,S2,E2,SR,ER):- % else, select the best one
   ( (Opt=max,max_list([E1,E2],ER));(Opt=min,min_list([E1,E2],ER)) ),
   ( (ER==E1,SR=S1); (ER==E2,SR=S2) ).

% show evolution:
show(final,Verbose,S1,E1,_,_):-
                 Verbose>0,
                 write('final:'),write(' S:'),write(S1),write(' E:'),write(E1),nl.
show(0,Verbose,S1,E1,_,_):-
                Verbose>0,
                write('init:'),write(' S0:'),write(S1),write(' E0:'),write(E1),nl.
show(I,Verbose,S1,E1,S2,E2):-
                0 is I mod Verbose,
                write('iter:'),write(I),write(' S1:'),write(S1),write(' E1:'),
                write(E1),write(' S2:'),write(S2),write(' E2:'),write(E2),nl.
show(_,_,_,_,_,_).
```