Local and Global Search Algorithms

- Motivation: local vs global optimization
- General structure of the local search algorithms
- Local Search Deterministic Methods:
  - Pattern Search
  - Nelder Mead
- Local Search Random Methods :
  - Matyas
  - Solis-Wets
- Metaheuristics for global search:
  - Local search with random restarts
  - Iterated local search
  - Simulated Annealing

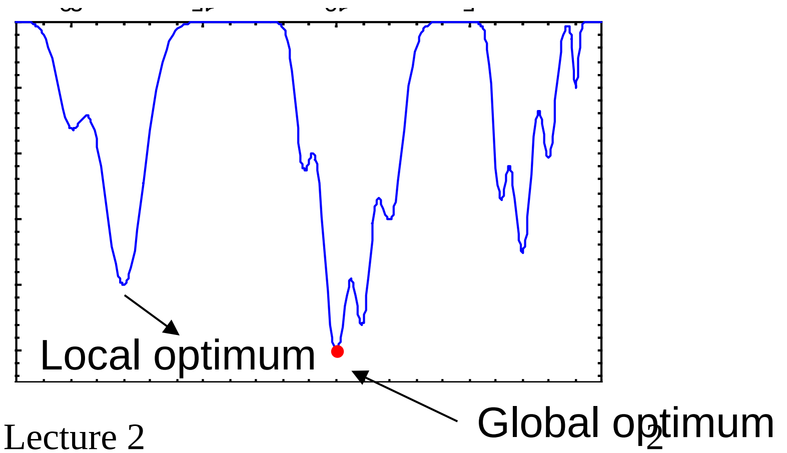
Local vs Global Optimization

Local optimization (minimization): find x* such that f(x*)<=f(x) for all x in V(x*) (V(x*)=neighborhood of x);

Rmk: it requires the knowledge of an initial approximation

Global optimization:

- Find x* such that f(x*)<=f(x), for any x (from the entire search domain)
- If the objective function has local optima then the local search methods (e.g. Gradient methods) can be stucked in such a local optimum



Local Optimization

Discrete search space:

- The neighborhood of an element is a finite set which can be completely explored

Particular case (permutation-like solutions):

- $s=(s_1,s_2,\ldots,s_n)$ s_i from $\{1,\ldots.,n\}$
- $V(s)=\{s'|s'$ can be obtained from s by interchanging two elements$\}$
- Card $V(s)=n(n-1)/2$

Example (n=4)

s=(2,4,1,3)

s'=(1,4,2,3)

Continuous search space:

a) The objective function is differentiable

- Gradient method
- Newton-like methods

b) The objective function is not differentiable (or even discontinuous)

- Direct search methods(ex: Nelder Mead)
- Methods based on small random perturbations

Local search: general structure

Notations:

S – search space

f – objective function

S_* - set of local/global optima

$s=(s_1,s_2,\ldots, s_n)$: element of S/ configuration/ candidate solution

s_* = the best element discoverd up to the current step

s^* = optimal solution

Local search algorithm:

```
s = initial approximation
repeat
   s'=perturb(s)
   if f(s')<f(s) then
      s=s'
until <stopping condition>
```

Remarks:

1. The initial approximation can be selected randomly or constructed based on a simple heuristic (e.g. greedy)
2. The perturbation can be deterministic (e.g. gradient based) or random
3. The replacement of s with s' can be done also when f(s')=f(s) (the condition is in this case f(s')<=f(s))
4. Stopping condition:
   (a) No improvement during the previous K iterations;
   (b) Maximal number of iterations or of objective function evaluations

Local search: variants (I)

Local search algorithm:

```
s = initial approximation
repeat
  s'=perturb(s)
  if f(s')<f(s) then
    s=s'
until <stopping condition>
```

More candidates:

```
s = initial approximation
repeat
  s'=perturb(s)
  for k=1:m
    s'' = perturb(s)
    if f(s'')<f(s)  then s'=s''
  end
  if f(s')<f(s) then
    s=s'
until < stopping condition >
```

Remarks:

1. The search is more explorative – at each iteration there are several candidates which are analyzed
2. Each objective function evaluation should be counted (if the stopping condition uses the number of evaluations)

Local search: variants (II)

Local search algorithm:

```
s = initial approximation
repeat
   s'=perturb(s)
   if f(s')<f(s) then
      s=s'
until <stopping condition>
```

More candidates:

```
s = initial approximation
best = s
repeat
   s'=perturb(s)
   for k=1:m
      s'' = perturb(s)
      if f(s'')<f(s)   then s'=s''
   end
   s=s'
   if f(s)<f(best) then best=s
until < stopping condition >
```

Remarks:

1. The best out of the m candidate solutions is unconditionally accepted
2. The best candidate solution obtained up to the current moment is preserved (ensuring the elitism of the searching process; elitism = we cannot lose the a good configuration once that it has been found)

Local search: perturbation variants

- Aim of the perturbation: constructing a new candidate solution starting from the existing one

- Perturbation types (depending on the nature of the perturbation):
  - Deterministic
  - Random

- Perturbation types (depending on the perturbation intensity):
  - Local
  - Global

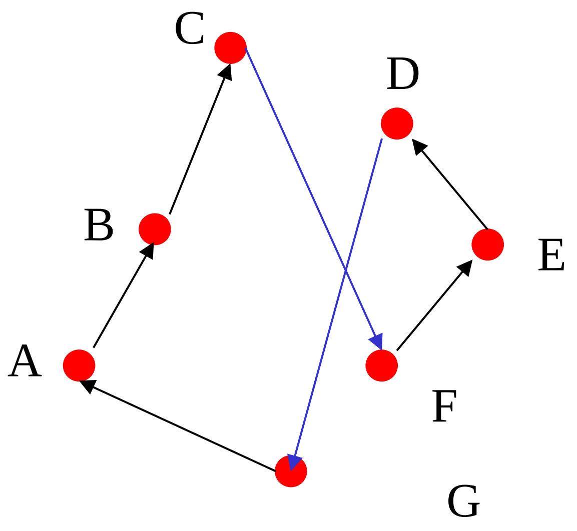
- Perturbation types (depending on the search space):
  - Discrete search space (replacement of one or several components)
  - Continuous search space (adding a perturbing term to the current configuration)

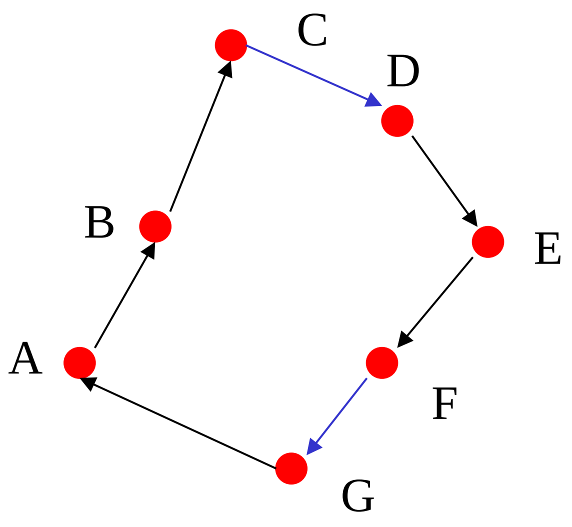
Local search: perturbation variants

Combinatorial optimization problems: the new configuration is chosen in the neighborhood of the current one by applying some transformations which are typical to the problem to be solved

Example 1: TSP (Travelling Salesman Problem)

- Generating a new configuration (2-opt transformation)





Implementation:

1. Random choice of two positions
2. Reverse the order of elements between the two selected positions

ABCFEDG $\longrightarrow$ ABCFEDG $\longrightarrow$ ABCDEFG

Local search: perturbation variants

Combinatorial optimization problems: the new configuration is chosen in the neighborhood of the current one by applying some transformations which are typical to the problem to be solved

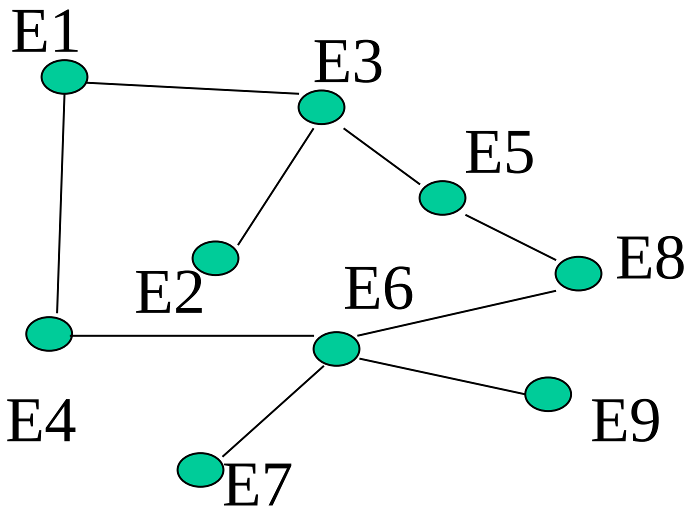
Example 2: Timetabling

- Remove conflicts (violated constraints) by moving or exchanging elements

- Current configuration perturbation:
  - Move an event which violates a constraint in a free slot

	S1	S2	S3
T1	E1	E3	E9
T2	E4		E8
T3	E6	E5	
T4	E2		E7

	S1	S2	S3
T1	E1		E9
T2	E4	E3	E8
T3	E6	E5	
T4	E2		E7



Conflicts graph

Local search: perturbation variants

Combinatorial optimization problems: the new configuration is chosen in the neighborhood of the current one by applying some transformations which are typical to the problem to be solved

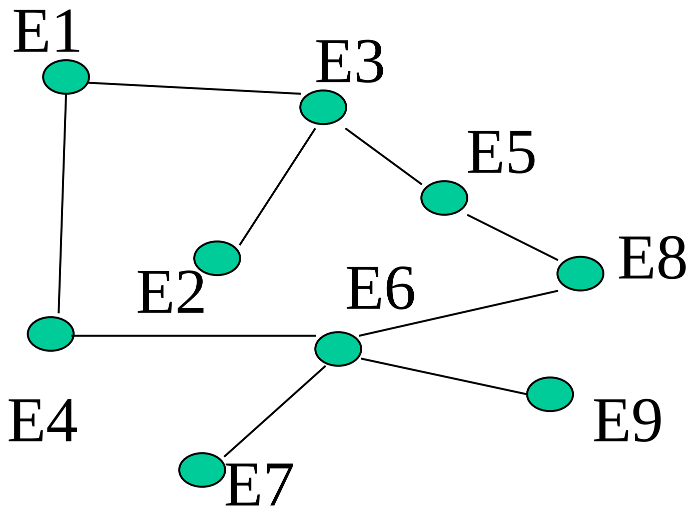
Example 2: Timetabling

- Remove conflicts (violated constraints) by moving or exchanging elements

- Current configuration perturbation:
  - Exchange two events

	S1	S2	S3
T1	E1		E9
T2	E4	E3	E8
T3	E2 $\updownarrow$	E5	
T4	E6		E7

	S1	S2	S3
T1	E1		E9
T2	E4	E3	E8
T3	E6	E5	
T4	E2		E7



Conflicts graph

Local search: perturbation variants

Optimization in continuous domains

Random perturbation

```
Perturb(s,p,inf,sup,r)
   for i=1:n
      if rand(0,1)<=p then
         repeat
            n=rand(-r,r)
         until inf<=$s_i$+n<=sup
         $s_i$=$s_i$+n
      end
   end
   return s
```

Notations:

s=the candidate solution to be perturbed

p=perturbation probability

r=perturbation „radius”

rand(a,b) = random value uniformly distributed on [a,b]

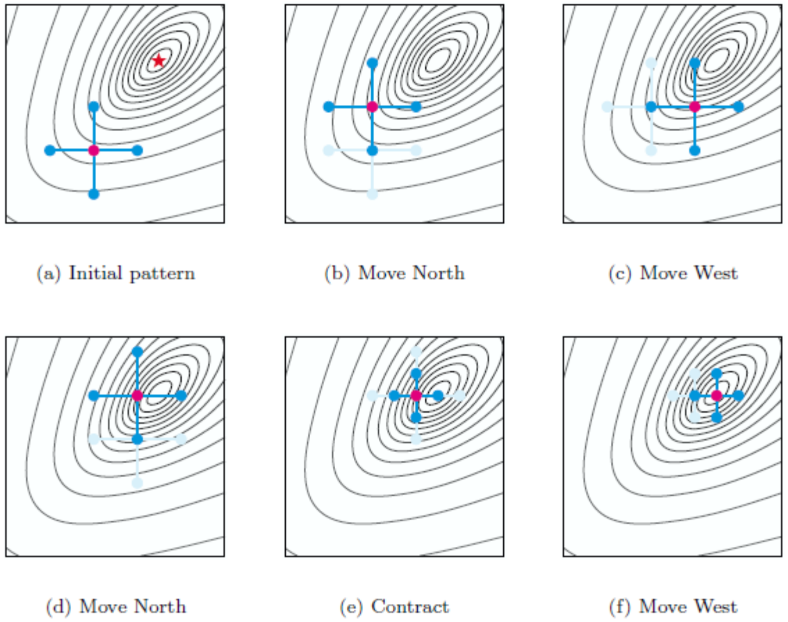
Deterministic perturbation by direct search (it does not use derivatives)

- Pattern Search (Hooke -Jeeves)
- Nelder - Mead

Local search: pattern search

Idea: successive modifications of the components of the current configuration

```
PatternSearch(s,r)
  s=initial approximation
  r=initial value
  best=s
  repeat
    s'=s
    for i=1:n
      if $f(s+r*e_i) < f(s')$ then $s'=s+r*e_i$ end
      if $f(s-r*e_i) < f(s')$ then $s'=s-r*e_i$ end
    end
    if s==s' then r=r/2
             else s=s'
    end
    if f(s)<f(best) then best=s
  until <stopping condition>
```


(a) Initial pattern (b) Move North (c) Move West
(d) Move North (e) Contract (f) Move West

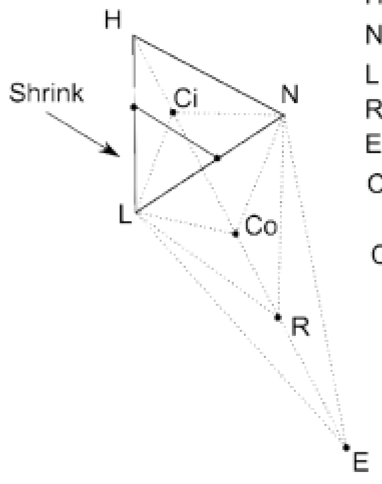
T.G. Kolda et al., Optimization by direct search: new perspectives on some classical and modern methods, SIAM Review, 45(3), 385-482, 2003

Remark:

1. $e_i=(0,0,...,0,1,0,...,0)$ (1 on position i)
2. At each iteration are constructed 2n candidates out of which the best one is selected

Local search: Nelder-Mead algorithm

Idea: the search is based on a simplex in R^n (set of (n+1) points in R^n) and on some transformations which allow to „explore" the search space



H = Highest
N = Next to highest
L = Lowest
R = Reflexion
E = Expansion
Ci = Contraction (inside)
Co = Contraction (outside)

The transformations are based on:

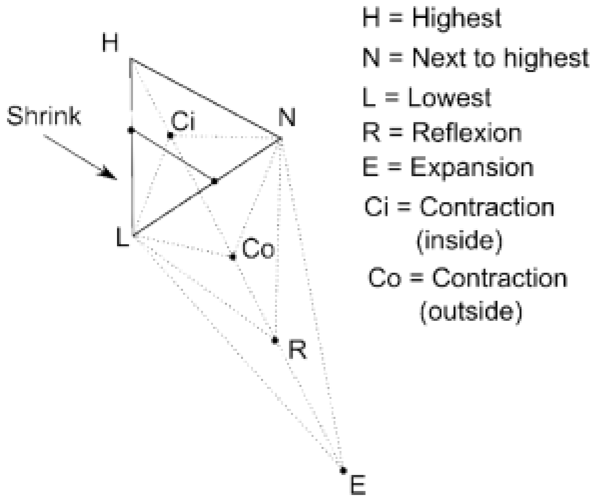
1. Sort the simplex elements increasingly by the objective function value (for a minimization problem)
2. Compute the average, $M(x_1,\ldots,x_n)$, of the best n elements from the simplex
3. Successive construction of new elements by: reflexion, expansion, contraction (interior, exterior), shrinking

J.G. Lagarias et.al; Convergence properties of the Nelder-Mead simplex method in low dimensions, SIAM J. Optim., 1998

Local search: Nelder-Mead algorithm

```
Select (n+1) points from $R^n$: $(x_1,x_2,..., x_{n+1})$
Repeat
    compute $(f_1,f_2,..., f_{n+1})$, $f_i=f(x_i)$
    sort $(x_1,x_2,..., x_{n+1})$ such that $f_1<=f_2<=...<=f_{n+1}$
    $M=(x_1+x_2+...+x_n)/n$

    Step1 (reflexion - R):
        $xr=M+r(M-x_{n+1})$;
        if $f_1<=f(xr)<f_{n+1}$ accept xr; continue;
        else goto Pas 2
    Step 2 (expansion - E):
        if $f(xr)< f_1$ then
            $xe=M+e(xr-M)$
            if $f(xe)<f(xr)$ then accept xe; continue
            else goto Pas 3
```



Local search: Nelder-Mead algorithm

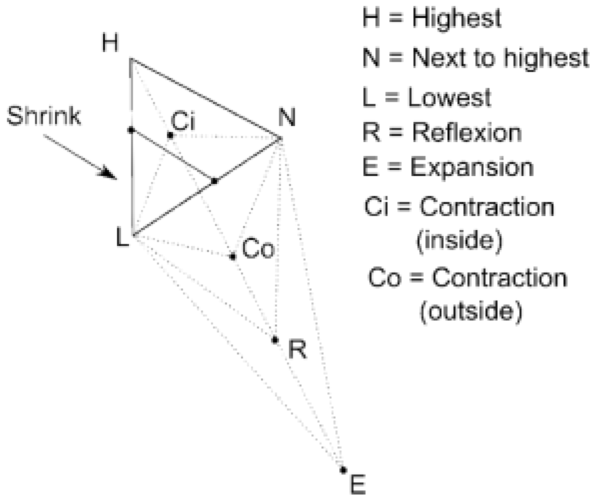
Step 4 (contraction exterior/interior – Co/Ci):

```
if $f_n$<=f(xr)<$f_{n+1}$ then
    xc=M+c(xr-M)
    if f(xc)<f(xr) accept xc; continue
    else goto Pas 5
if f(xr)>=$f_{n+1}$ then
    xcc=M-c(M-$x_{n+1}$)
    if f(xcc)<$f_{n+1}$ then accept xcc; continue
    else goto Pas 5
```

Step 5 (Shrinkink):

construct a new simplex:

$x_1, v_2, \ldots, v_{n+1}$ unde $v_i = x_i + s(x_i - x_1)$



Parameters: r=1, e=2, c=1/2, s=1/2

From local to global optimization

Perturbation: use (ocasionally) some large perturbations

Example: use a infinite support probability distribution (e.g. Normal or Cauchy distribution – algoritm Matyas, Solis-Wets)

Random restart: start a new search process from a random initial configuration

Example: local search with random restarts

Exploration of the local optima set: the current local optimum is perturbed and used as a starting point for a new search process

Example: iterated local search

Selection: accept (ocasionally) poorer configurations

Example: simulated annealing

Example: Matyas algorithm(1960)

```
s(0) = initial configuration
k=0    // iteration counter
e=0    // failure counter
repeat
    generate a random vector with normally
     distributed components ($z_1,\ldots z_n$)
    IF  f(s(k)+z)<f(s(k)) THEN s(k+1)=s(k)+z
                                  e=0
                         ELSE s(k+1)=s(k)
                                  e=e+1
    k=k+1
UNTIL (k==kmax) OR (e==emax)
```

Rmk. The random perturbation is usually applied to one of the components (e.g. the vector z has only one non-zero component)

Problem: how should be chosen the parameters of the distribution used to perturb the current value?

Example: N(0,sigma)

Reminder: simulation of random variables with normal distribution

Box-Muller algorithm

```
u=rand(0,1)   // random value uniformly distributed on (0,1)
v=rand(0,1)
r=sqrt(-2*ln(u));
z1=r*cos(2*PI*v)
z2=r*sin(2*PI*v)
RETURN z1,z2
    // z1 and z2 can be considered as values of two independent random
    variables with normal distribution
```

Reminder: simulation of random variables with normal distribution

Other variant of the Box-Muller algorithm:

```
repeat
  u=rand(0,1) v=rand(0,1)
  $w=u^2+v^2$
until $0<w<1$
y=sqrt(-2ln(w)/w)
z1=u*y
z2=v*y
RETURN z1,z2
```

Rmk: to obtain values corresponding to a non-standard normal distribution N(m,sigma) one have to apply the transformation: m+z*sigma

Example: Solis-Wets algorithm (1981)

```
s(0) = initial configuration
k=0;  m(0)=0  // the average of the perturbation vector is adaptive
repeat
    generate a vector $(z_1,\ldots z_n)$ having components distributed according toN(m(k),1)
    IF  f(s(k)+z)<f(s(k)) THEN s(k+1)=s(k)+z;
                                   m(k+1)=0.4*z+0.2*m(k)

    IF f(s(k)-z)<min{f(s(k)),f(s(k)+z)}  THEN s(k+1)=s(k)-z;
                                                m(k+1)=m(k)-0.4*z

    IF f(s(k)-z)>f(s(k)) AND f(s(k)+z)>f(s(k)) THEN
                                                    s(k+1):=s(k)
                                                    m(k+1):=0.5*m(k)
    k:=k+1
UNTIL (k==kmax)
```

Search with random restarts

Idea:

- The search process is repeated starting from random initial configurations
- The best final configuration is chosen as solution

Remarks:

- The stopping condition of the local search can be based on a random decision (e.g. The allocated time can be random)
- The search processes are independent – none of the information collected at the previous search threads is used

Random Restart

```
s=initial configuration
best=s
Repeat
    repeat
        r=perturb(s)
        if f(r)<=f(s) then s=r
    until <conditie oprire căutare
        locală>
    if f(s)<f(best) then best =s
    s=other initial configuration
        (random)
until  <stopping condition>
return best
```

Iterated Local Search

Idea:

- It is based on some successive local search stages which are correlated
- The initial configuration from the next stage is chosen in a neighborhood of the local optimum identified at the current stage

Remark:

- The initial configuration of a new search stage is based on a more „aggressive" perturbation than the perturbation used for local search

Iterated Local Search (ILS)

```
s=initial configuration
s0=s; best=s
Repeat
    repeat
       r=perturbSmall(s)
       if f(r)<=f(s) then s=r
    until <local stopping condition>
    if f(s)<f(best) then best =s
    s0=choose(s0,s)
    s=perturbLarge(s0)
until  <stopping condition>
return best
```

Simulated Annealing

Idea:

- accept, with some probability, also perturbations which lead to an increase of the objective function (in the case of minimization problems)

Inspiration:

- SA algorithms are inspired by the process of restructuring the internal configuration in a solid which is annealed (e.g. crystallization process):
  - The solid is heated (up to the melting point): its particles are randomly distributed.
  - The material is the slowly cooled down: its particles are reorganized in order to reach a low energy state

Contributors: Metropolis(1953), Kirkpatrick, Gelatt, Vecchi (1983), Cerny (1985)

Simulated Annealing

Analogy:

Physical process:		Minimization problem:
• System energy	→	Objective function
• System state	→	Configuration (candidate solution)
• Change of the system state	→	Perturbation of the current configuration
• Temperature	→	Parameter which controls the optimization process

Simulated Annealing

Some physics:

- Each state of the system has a corresponding probability
- The probability corresponding to a given state depends on the energy of the state and on the system temperature (Boltzmann distribution)

$$P_T(s) = \frac{1}{Z(T)} \exp\left(-\frac{E(s)}{k_B T}\right)$$

$$Z(T) = \sum_{s \in S} \exp\left(-\frac{E(s)}{k_B T}\right)$$

E(s) = energy of state s
T = temperature
Z(T)=partition function
(normalization factor)
k_B = Boltzmann constant

Simulated Annealing

Some physics:

- Large values of T (T goes to infinity): the argument of exp is almost 0 => the states have all the same probability

- Small values of T (T goes to 0): only the states with non-zero energy will have non-zero probabilities

$$P_T(s) = \frac{1}{Z(T)} \exp\left(-\frac{E(s)}{k_B T}\right)$$

$$Z(T) = \sum_{s \in S} \exp\left(-\frac{E(s)}{k_B T}\right)$$

E(s) = energy of state s
T = temperature
Z(T)=partition function
(normalization factor)
k_B = Boltzmann constant

Simulated Annealing

How can we use these results from physics to solve an optimization problem ?

- It would be enough to generate configurations according to the Boltzmann distribution for smaller and smaller values of the temperature.

- **Problem:** it is difficult to compute the partition function Z(T) (it means to compute a sum over all possible configurations in the search space which is practically impossible for real-world problems – it would correspond to an exhaustive search)
- **Solution:** the distribution is approximated by simulating the evolution of a stochastic process (Markov chain) having as stationary distribution the Boltzmann distribution => Metropolis algorithm

Simulated Annealing

Metropolis algorithm (1953)

```
Init x(0)
k:=0
REPEAT
  x':=perturb(x(k))
   IF  f(x')<f(x(k)) THEN x(k+1):=x'  (unconditionally)
                          ELSE x(k+1):=x'
                                 with probability min{1,exp(-(f(x')-f(x(k))/T)}

   k:=k+1
UNTIL "a stopping condition is satisfied"
```

Simulated Annealing

Properties of the Metropolis algorithm

- Another acceptance probability:

  $$P(x(k+1)=x') = 1/(1+\exp((f(x')-f(x(k))/T))$$

- Implementation issue: assigning a value with a given probability is based on generating a random value in (0,1)

```
u:=Random(0,1)
IF u<P(x(k+1)=x') THEN x(k+1)=x'
                  ELSE x(k+1)=x(k)
```

- Large values for T -> high acceptance probability for any configuration (pure random search)

  Small values for T -> High acceptance probabilities only for the states with low energy values (greedy search - similar to a gradient descent method)

Simulated Annealing

Properties of the Metropolis algorithm

- The rules used to generate new configurations depend on the problem to be solved

Optimization in continuous domains

$x' = x + z$

$z = (z_1, \ldots, z_n)$

z_i : generated according to the distribution:

- $N(0,T)$
- Cauchy(T) (Fast SA)
- etc

Combinatorial optimization

The new configuration is selected deterministically or randomly from the neighborhood of the current configuration

Example: TSP – 2-opt transformation

Simulated Annealing

Simulated Annealing = repeated application of the Metropolis algorithm for decreasing values of the temperature

General structure

```
Init x(0), T(0)
i:=0
REPEAT
    apply Metropolis  (for kmax iterations)
    compute T(i+1)
    i:=i+1
UNTIL T(i)<eps
```

Problem: How to choose the cooling scheme ?

Simulated Annealing

Cooling schemes:

$T(k)=T(0)/(k+1)$

$T(k)=T(0)/\ln(k+c)$

$T(k)=aT(k-1)$ ($a<1$, ex: $a=0.995$)

Remark. T(0) should be chosen such that during the first iterations almost all new configurations are accepted (this ensures a good exploration of the search space)

Simulated Annealing

Convergence properties:

If the following properties are satisfied:

- $Pg(x(k+1)=x'|x(k)=x)>0$ for any x and x' (the transition probability between any two configurations is non-zero)
- $Pa(x(k+1)=x'|x(k)=x)=\min\{1,\exp(-(f(x')-f(x))/T)\}$ (Metropolis acceptance probability)
- $T(k)=C/\lg(k+c)$ (logarithmic cooling schedule)

then $P(f(x(k))=f(x^*)) \to 1$ (x(k) is convergent in probability to the global minimum x^*)

Simulated Annealing

Variant: another acceptance probability (Tsallis)

$$P_a(x') = \begin{cases} 1, & \Delta f \le 0 \\ (1-(1-q)\Delta f / T)^{1/(1-q)}, & \Delta f > 0,\ (1\text{-}q)\Delta f \le 1 \\ 0, & \Delta f > 0,\ (1\text{-}q)\Delta f > 1 \end{cases}$$

$$\Delta f = f(x') - f(x)$$

$$q \in (0,1)$$

Simulated Annealing

Example: Travelling Salesman Problem (TSP)

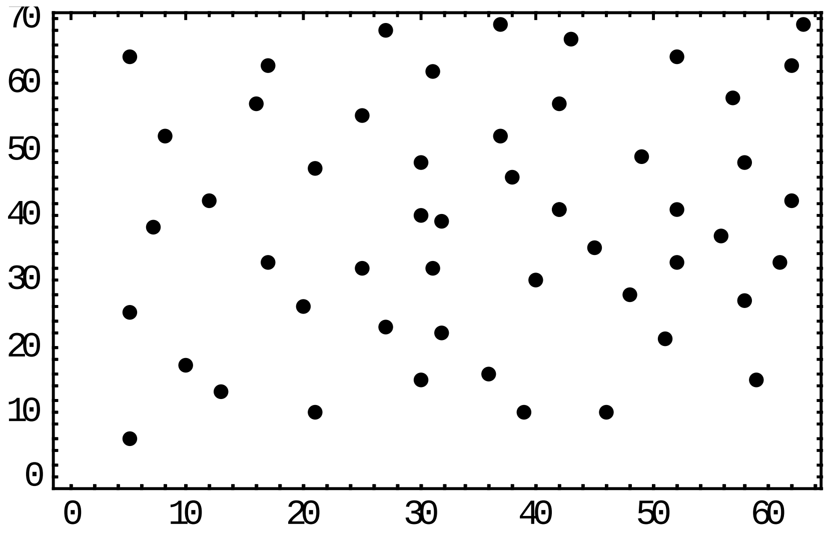
(TSPLib: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95)

Test instance: eil51 – 51 towns

Parameters:

- 5000 iterations; T is changed at each 100 iterations
- T(k)=T(0)/(1+log(k))

Location of towns

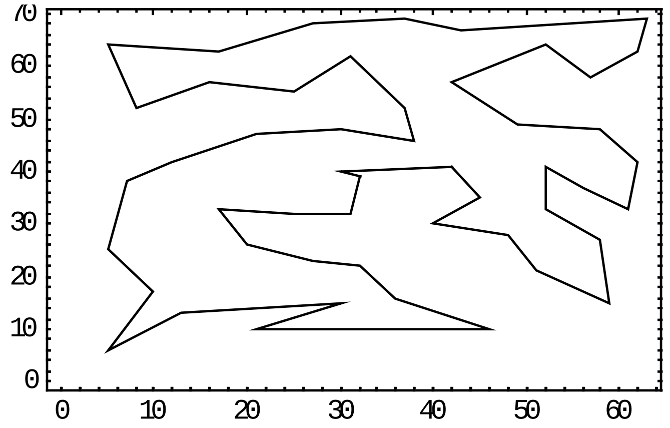


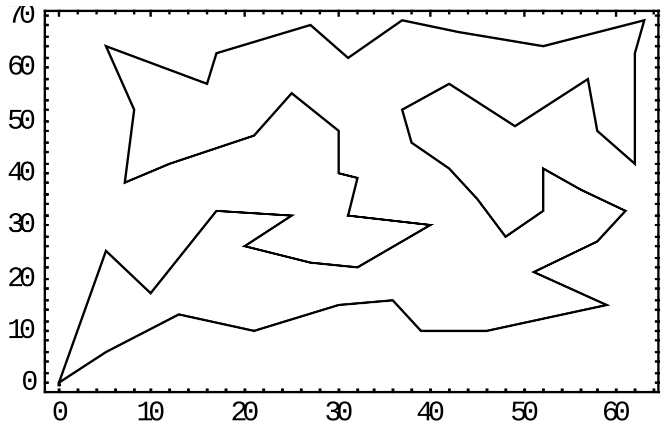
Simulated Annealing

Example: TSP

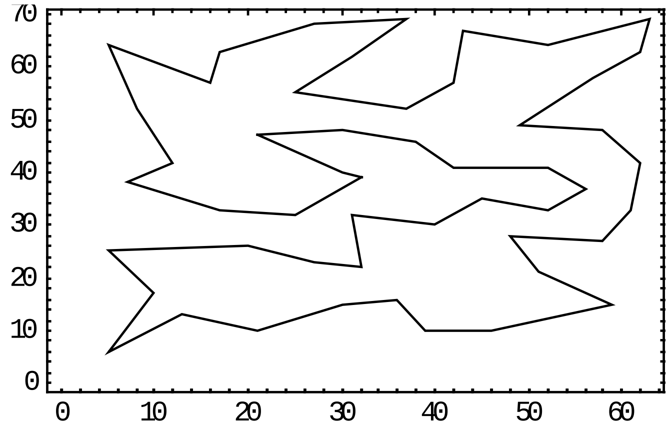
Test instance: eil51 (TSPlib)

T(0)=5, cost=474.178





T(0)=1, cost=481.32



T(0)=10,cost=478.384

Minimal cos: 426