Trajectory based Search Algorithms (I)

- 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

Local vs Global Optimization

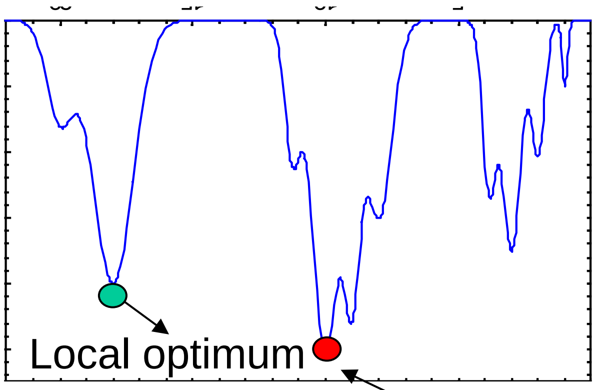
Local optimization (minimization): find x* such that f(x*)<=f(x) for all x in V(x*)

(V(x*)=neighborhood of x);

Rmk: it requires the knowledge of an initial approximation and the search will focus on the neighborhood of this 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 get stuck 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):

- $s=(s_1,s_2,...,s_n)$ s_i from {1,....,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 – the search direction is established based on the changes in the objective function -> direction of increase (minimization) or decrease (maximization)

- Gradient method (first order derivatives-> first order methods)
- Newton-like methods (second order derivatives -> second order methods)

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

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

(no derivatives are used -> zero-order methods)

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 discovered 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 number 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
   [s1,…, sm]= MultiplePerturb(s)
   s'=bestOf([s1,…, sm])
   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)

More candidates:

```
s = initial approximation
repeat
   [s1,…, sm]= MultiplePerturb(s)
   s'=bestOf([s1,…, sm])
   if f(s')<f(s) then s=s'
until <stopping condition>
Return s
```

More candidates – other variant:

```
s = initial approximation
best = s
repeat
   [s1,…, sm]=MultiplePerturb(s)
   s=bestOf([s1,…, sm])
   if f(s)<f(best) then best=s
until <stopping condition>
return best
```

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 = the best configuration so far is saved – if a good configuration is found it cannot be lost)

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 (e.g. hill climbing = choose the best configuration in the neighborhood)
  - Random (e.g. random walk = choose a random configuration from the neighborhood)

- Perturbation types (depending on the perturbation intensity):
  - Local (small) -> exploitation (intensification of search)
  - Global (large) -> exploration (diversification of the search)

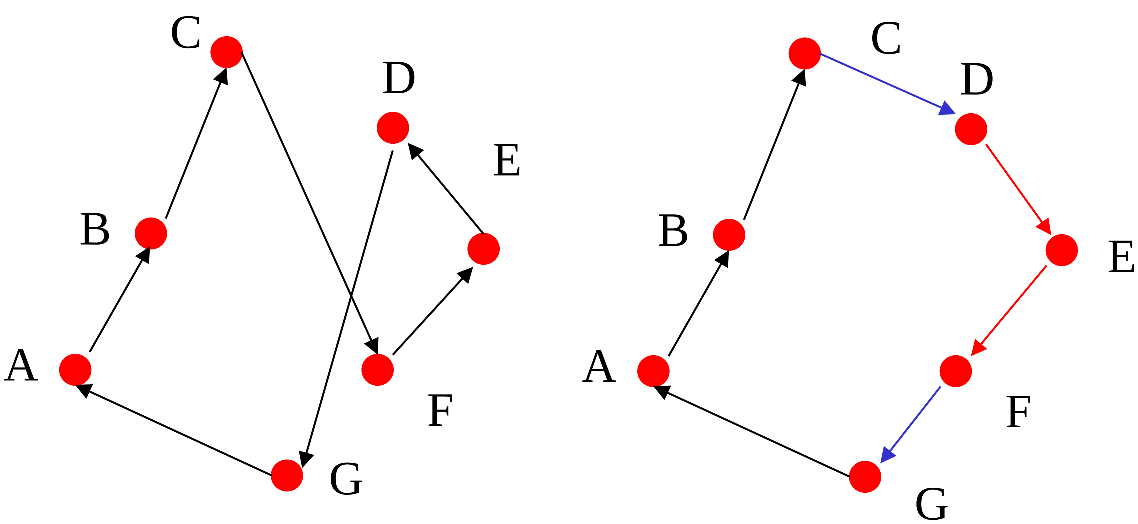
- Perturbation types (depending on the search space):
  - Discrete search space (replacement of one or several components)
  - Continuous search space (adding a perturbing term to the current configuration)

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

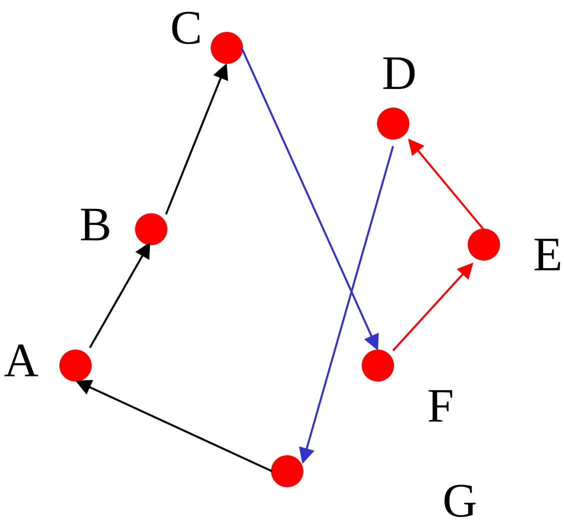
ABCFEDG → ABCFEDG → ABCDEFG

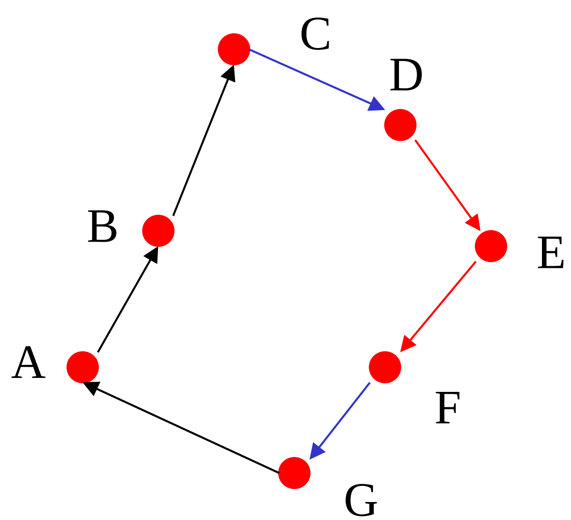
Local search: perturbation variants

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 1: TSP (Travelling Salesman Problem)

- Generating a new configuration (2-opt transformation)





What kind of perturbation?

- Random
- Local (?)
- Based on a finite neighborhood (discrete search space)

ABCFEDG → ABCFEDG → 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

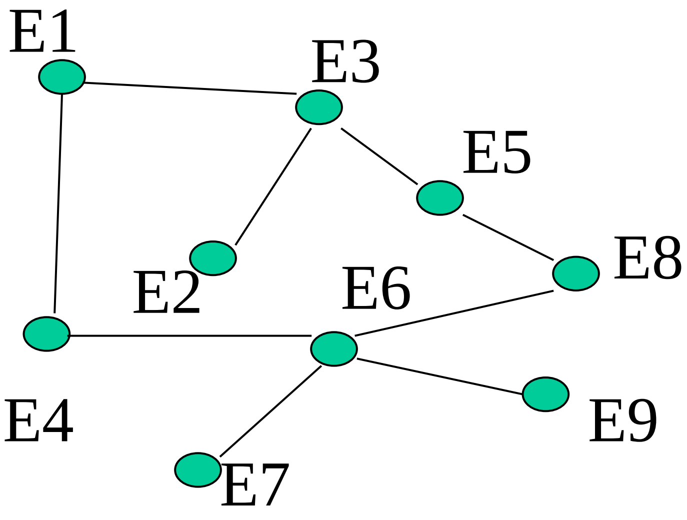
Example 2: Timetabling

- 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	E2	E5	
T4	E6		E7

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

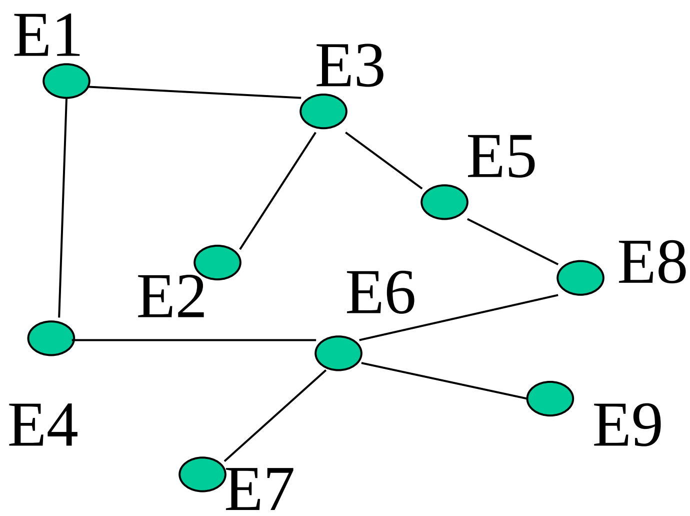
Example 2: Timetabling

- 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	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
      pert=rand(-r,r)
    until inf<=s_i+pert<=sup
    s_i=s_i+pert
   end if
 end for
 return s
```

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

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

Notations:

s=the candidate solution to be perturbed

p=perturbation probability

r=perturbation „radius”

[inf, sup] = search range

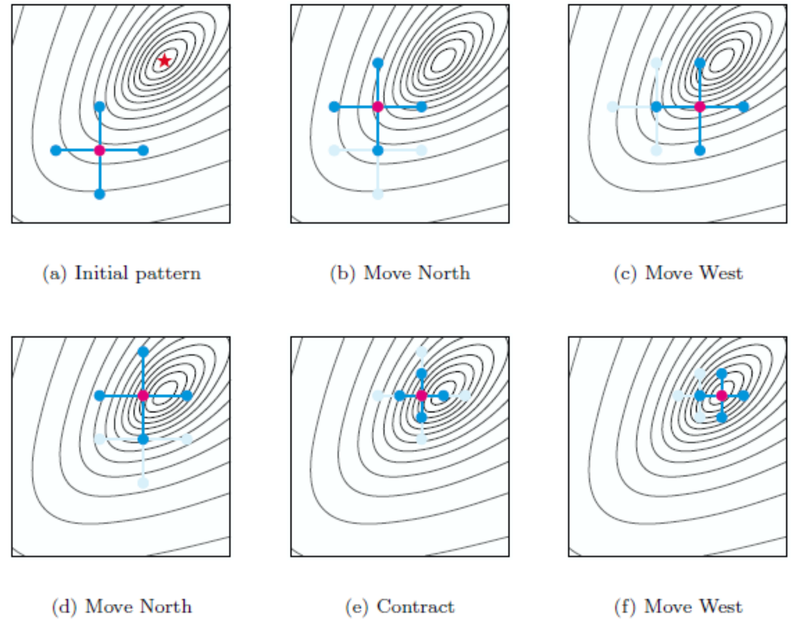
n = problem size (number of components of a solution

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

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
     if f(s-r*e_i)< f(s') then s'=s-r*e_i
    end for
    if s==s' then r=r/2
             else s=s'
    end
    if f(s)<f(best) then best=s
  until <stopping condition>
  return best
```



(a) Initial pattern (b) Move North (c) Move West

(d) Move North (e) Contract (f) Move West

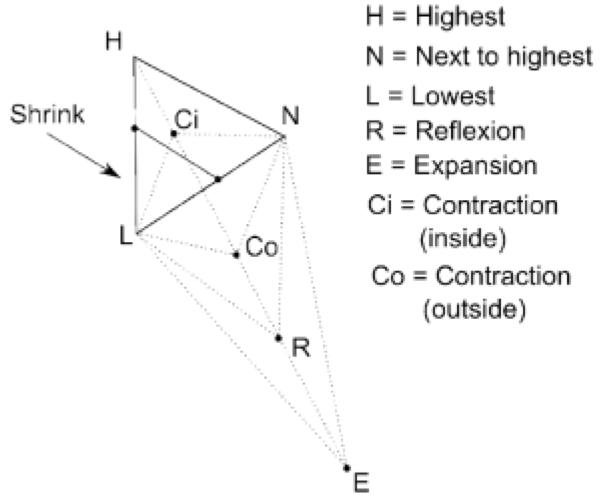
T.G. Kolda et al., Optimization by direct search: new perspectives on some classical and modern methods, SIAM Review, 45(3), 385-482, 2003

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

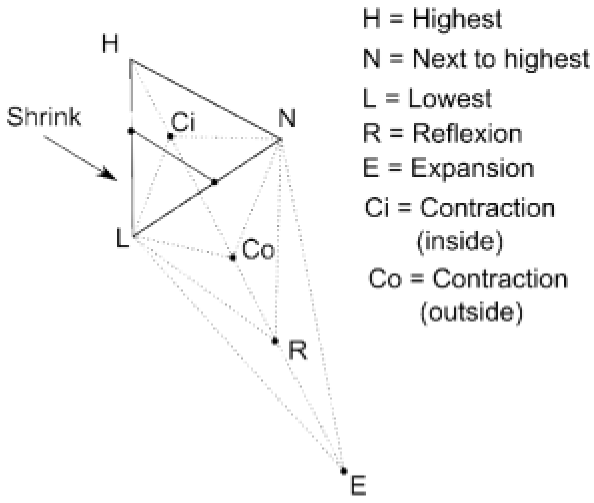


The transformations are based on:

1. Sort the simplex elements increasingly by the objective function value (for a minimization problem)
2. Compute the average, $M(x_1,...,x_n)$, 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 $f_1 <= f(xr) < f_n$ then accept xr; continue;

else goto Step 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

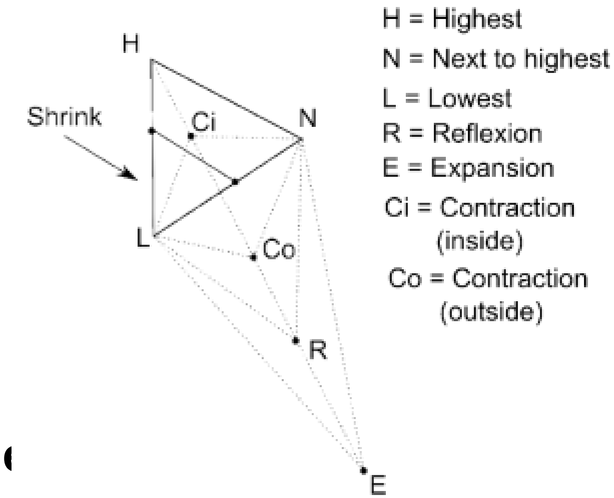
else goto Step 3

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) then accept xc; continue
                    else goto Step 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 Step 5
```

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



Step 5 (Shrinking) construct a new simplex:

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

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 = initial configuration
k=0    // iteration counter
e=0    // failure counter
repeat
  //generate a random vector z with
  //normally distributed components
  //(z1,…zn)
   z=random vector
   if  f(s+z)<f(s)
       then s=s+z
            e=0
       else 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 standard normal
   // distribution (N(0,1))
```

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=u²+v²
until 0<w<1
y=sqrt(-2*ln(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 //the average of the perturbation vector is adaptive
repeat
  //generate a vector (z1,…zn) with components from N(m,1)
  z=random vector
  if f(s+z)<f(s) then s=s+z   // accept the perturbation
                      m=0.4*z+0.2*m   // adjust the mean
  if f(s-z)<min{f(s),f(s+z)} then s=s-z // accept the perturb.
                                  m=m-0.4*z
  if f(s-z)>f(s) AND f(s+z)>f(s) then m=0.5*m
  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 <local search stopping
    condition>
  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
```

Iterated Local Search

Remarks [T. Stutzle – Tutorial on Iterated Local Search, 2003]

- The perturbation used to construct the new starting configuration (perturbLarge) should be chosen such that it is not easily undone by the local search (perturbSmall)
- ILS defines a biased walk in the search space

Summary

- Trajectory based search – keeps track of only one candidate solution
- Local search – small perturbation of the current configuration
  - Deterministic: choose the best element in the neighborhood
  - Random: choose an arbitrary element from the neighborhood
- Global search – avoid local optima by
  - Restarting the search
  - Iterating the search
  - Combining deterministic and random perturbation
  - Changing the neighborhood size (e.g. Variable Neighborhood Search)
  - Controlling the set of visited configuration and of search intensification and diversification (e.g. Tabu Search)
  - Escaping from local optima by non-greedy acceptance (e.g. Simulated Annealing)

Next Lecture

Other global search methods:

- Variable Neighborhood Search
- Tabu Search
- Simulated Annealing
- Greedy Randomized Adaptive Search