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
Local vs Global Optimization

Local optimization (minimization): find $x^*$ such that $f(x^*) \leq f(x)$ 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 $f(x^*) \leq f(x)$, 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 \( \{1, \ldots, n\} \)
- \( V(s) = \{s' | s' \text{ can be obtained from } s \text{ by interchanging two elements}\} \)
- \( \text{Card } V(s) = n(n-1)/2 \)

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=(s₁,s₂,..., sₙ) : 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 objective function evaluations
Local search: variants (I)

Local search algorithm:

\[ s = \text{initial approximation} \]
\[ \text{repeat} \]
\[ s' = \text{perturb}(s) \]
\[ \text{if } f(s') < f(s) \text{ then} \]
\[ s = s' \]
\[ \text{until } \langle \text{stopping condition} \rangle \]

More candidates:

\[ s = \text{initial approximation} \]
\[ \text{repeat} \]
\[ s' = \text{perturb}(s) \]
\[ \text{if } f(s') < f(s) \text{ then} \]
\[ s = s' \]
\[ [s_1, \ldots, s_m] = \text{MultiplePerturbation}(s) \]
\[ s' = \text{bestOf}([s_1, \ldots, s_m]) \]
\[ \text{if } f(s') < f(s) \text{ then} \]
\[ s = s' \]
\[ \text{until } \langle \text{stopping condition} \rangle \]

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:

\[ s = \text{initial approximation} \]

repeat
\[ s' = \text{perturb}(s) \]
if \( f(s') < f(s) \) then
\[ s = s' \]
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)

More candidates:

\[ s = \text{initial approximation} \]
best = \( s \)
repeat
\[ [s_1, \ldots, s_m] = \text{MultiplePerturbation}(s) \]
\[ s = \text{bestOf}([s_1, \ldots, s_m]) \]
if \( f(s) < f(\text{best}) \) then \( \text{best} = s \)
until < stopping condition >
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 → 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

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>E1</td>
<td>E3</td>
</tr>
<tr>
<td>T2</td>
<td>E4</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>E6</td>
<td>E5</td>
</tr>
<tr>
<td>T4</td>
<td>E2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>E1</td>
<td>E9</td>
</tr>
<tr>
<td>T2</td>
<td>E4</td>
<td>E3</td>
</tr>
<tr>
<td>T3</td>
<td>E6</td>
<td>E5</td>
</tr>
<tr>
<td>T4</td>
<td>E2</td>
<td>E7</td>
</tr>
</tbody>
</table>

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

```
<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>E1</td>
<td>E9</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>E4</td>
<td>E3</td>
<td>E8</td>
</tr>
<tr>
<td>T3</td>
<td>E2</td>
<td>E5</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>E6</td>
<td>E7</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>E1</td>
<td>E9</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>E4</td>
<td>E3</td>
<td>E8</td>
</tr>
<tr>
<td>T3</td>
<td>E6</td>
<td>E5</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>E2</td>
<td>E7</td>
<td></td>
</tr>
</tbody>
</table>
```

Conflicts graph
Local search: perturbation variants

Optimization in continuous domains

Random perturbation

\[
\text{Perturb}(s, p, \text{inf}, \text{sup}, r) \\
\text{for } i = 1:n \\
\quad \text{if } \text{rand}(0,1) \leq p \text{ then} \\
\quad \quad \text{repeat} \\
\quad \quad \quad n = \text{rand}(-r, r) \\
\quad \quad \quad \text{until } \text{inf} \leq s_i + n \leq \text{sup} \\
\quad \quad s_i = s_i + n \\
\quad \text{end} \\
\text{end} \\
\text{return } s
\]

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"
- \(\text{rand}(a, b)\) = 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*ei end
if f(s-r*ei)< f(s′) then s′=s-r*ei end
endif

s==s′ then r=r/2
else s=s′
end
if f(s)<f(best) then best=s

until <stopping condition>

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 $\mathbb{R}^n$ (set of $(n+1)$ points in $\mathbb{R}^n$) 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, $M(x_1,\ldots,x_n)$, 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 \( \mathbb{R}^n \): \((x_1, x_2, ..., x_{n+1})\)

Repeat

- compute \((f_1, f_2, ..., f_{n+1})\), \(f_i = f(x_i)\)
- sort \((x_1, x_2, ..., x_{n+1})\) such that \(f_1 \leq f_2 \leq ... \leq f_{n+1}\)
- \(M = \frac{x_1 + x_2 + ... + x_n}{n}\)

Step 1 (reflection - R):

- \(x_r = M + r(M - x_{n+1})\);
- if \(f_1 \leq f(x_r) < f_n\) accept \(x_r\); continue;
- else goto Pas 2

Step 2 (expansion - E):

- if \(f(x_r) < f_1\) then
  - \(x_e = M + e(x_r - M)\)
  - if \(f(x_e) < f(x_r)\) then accept \(x_e\); continue
  - else goto Pas 3
Local search: Nelder-Mead algorithm

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

- if $f_n \leq f(x_r) < f_{n+1}$ then
  - $x_c = M + c(x_r - M)$
  - if $f(x_c) < f(x_r)$ accept $x_c$; continue
  - else goto Pas 5
- if $f(x_r) \geq f_{n+1}$ then
  - $x_{cc} = M - c(M - x_{n+1})$
  - if $f(x_{cc}) < f_{n+1}$ then accept $x_{cc}$; continue
  - else goto Pas 5

Step 5 (Shrinking):

- 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 (occasionally) some large perturbations
- Example: use a infinite support probability distribution (e.g. Normal or Cauchy distribution – algorithm 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 (occasionally) poorer configurations
- Example: simulated annealing
Example: Matyas algorithm (1960)

\[ s(0) = \text{initial configuration} \]
\[ k=0 \quad // \text{iteration counter} \]
\[ e=0 \quad // \text{failure counter} \]
repeat
\[ \text{generate a random vector with normally distributed components } (z_1, \ldots z_n) \]
\[ \text{IF } f(s(k)+z) < f(s(k)) \text{ THEN } s(k+1) = s(k) + z \]
\[ e=0 \]
\[ \text{ELSE } s(k+1) = s(k) \]
\[ e=e+1 \]
\[ k=k+1 \]
UNTIL \((k==k_{\text{max}}) \text{ OR } (e==e_{\text{max}})\)

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

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

\textbf{Example:} N(0,\sigma)
Reminder: simulation of random variables with normal distribution

**Box-Muller algorithm**

\[
\begin{align*}
u &= \text{rand}(0,1) \quad \text{// random value uniformly distributed on (0,1)} \\
v &= \text{rand}(0,1) \\
r &= \sqrt{-2 \ln(u)}; \\
z_1 &= r \cos(2\pi v) \\
z_2 &= r \sin(2\pi v) \\
\end{align*}
\]

RETURN \( z_1, z_2 \)

// \( z_1 \) and \( z_2 \) 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)

\[ s(0) = \text{initial configuration} \]
\[ k=0; \ m(0)=0 \ // \text{the average of the perturbation vector is adaptive} \]
\[ \text{repeat} \]
\[ \quad \text{generate a vector } (z_1, \ldots, z_n) \text{ having components distributed according to } N(m(k), 1) \]
\[ \text{IF } f(s(k)+z) < f(s(k)) \text{ THEN } s(k+1)=s(k)+z; \]
\[ \quad \mbox{m}(k+1)=0.4z+0.2\mbox{m}(k) \]
\[ \text{IF } f(s(k)-z) < \min\{f(s(k)), f(s(k)+z)\} \text{ THEN } s(k+1)=s(k)-z; \]
\[ \quad \mbox{m}(k+1)=\mbox{m}(k)-0.4z \]
\[ \text{IF } f(s(k)-z) > f(s(k)) \text{ AND } f(s(k)+z) > f(s(k)) \text{ THEN} \]
\[ \quad s(k+1):=s(k) \]
\[ \quad \mbox{m}(k+1):=0.5\mbox{m}(k) \]
\[ k:=k+1 \]
\[ \text{UNTIL } (k==k_{\text{max}}) \]
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)
\[ s = \text{initial configuration} \]
\[ s_0 = s; \ best = s \]
Repeat
  repeat
    \[ r = \text{perturbSmall}(s) \]
    if \( f(r) \leq f(s) \) then \( s = r \)
  until <local stopping condition>
  if \( f(s) < f(\text{best}) \) then \( \text{best} = s \)
  \[ s_0 = \text{choose}(s_0, s) \]
  \[ s = \text{perturbLarge}(s_0) \]
until <stopping condition>
return best
Next Lecture

Other global search methods:

• Simulated Annealing
• Variable Neighborhood Search
• Tabu Search
• Greedy Randomized Search