A Course on Meta-Heuristic Search Methods for Combinatorial Optimization Problems

Santosh Kumar Mandal, *Ph.D research fellow*

AutOrI LAB,
DIA, Roma Tre
Email: mandal@dia.uniroma3.it

January 16, 2014
Outline

1. Simulated annealing algorithm
   - An example
   - Assignment-I Tips
   - Variants
2. Iterated local search
3. Variable neighbourhoods
   - Variable neighbourhood decent
   - Variable neighbourhood search
4. Guided local search
5. GRASP
6. Large neighbourhood search
7. Path relinking
8. Vehicle routing problem
\[ \max f(x) = x^3 - 60 \times x^2 + 900 \times x + 100 \] (1.1)

*Global maxima: \( f(x) = 4100 \) at \( x = 10 \) (1.2)*

- Neighbourhood operator: random flipping of a bit
- \( T_{\text{max}} = 500K \)
- Initial solution = 10011 (5 bits) with \( f(x) = 2399 \)
- Cooling schedule: \( T = 0.9 \times T \)
- \( \Delta f = f(x) - f'(x) \) (for maximization problem)

<table>
<thead>
<tr>
<th>( T )</th>
<th>Move</th>
<th>Solution</th>
<th>( f )</th>
<th>( \Delta f )</th>
<th>Move?</th>
<th>New Neighbor Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Yes</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Yes</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Yes</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Yes</td>
<td>01110</td>
</tr>
<tr>
<td>328</td>
<td>4</td>
<td>01100</td>
<td>3998</td>
<td>&lt;0</td>
<td>Yes</td>
<td>01100</td>
</tr>
<tr>
<td>295.2</td>
<td>3</td>
<td>01000</td>
<td>3972</td>
<td>16</td>
<td>Yes</td>
<td>01000</td>
</tr>
<tr>
<td>265.7</td>
<td>4</td>
<td>01010</td>
<td>4100</td>
<td>&lt;0</td>
<td>Yes</td>
<td>01010</td>
</tr>
<tr>
<td>239.1</td>
<td>5</td>
<td>01011</td>
<td>4071</td>
<td>29</td>
<td>Yes</td>
<td>01011</td>
</tr>
<tr>
<td>215.2</td>
<td>1</td>
<td>11011</td>
<td>343</td>
<td>3728</td>
<td>No</td>
<td>01011</td>
</tr>
</tbody>
</table>
Parameter settings

- **$T_{\text{max}}$:** Check 100 - 500 K
- **$T_{\text{min}}$:** = 0.01 K
- **Length of inner loop:** Check 100 - 500
- Cooling schedule:
  - Linear.
    \[ T = T - \beta \quad \beta < 1 \]
  - Geometric
    \[ T = T \times \alpha \quad \alpha < 1 \]
  - Decide values of $\alpha$ and $\beta$ on your own.
- Acceptance probability for bad solutions:
  \[ \exp\left(\frac{-\Delta}{T}\right) > \text{rand(.)} \quad \text{rand(.)}: \text{a random number between 0 and 1} \]
Binary to decimal

100101

\[(1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 37\]

- Each binary digit represents an increasing power of 2, with the rightmost digit representing \(2^0\), the next representing \(2^1\), then \(2^2\), and so on.
- To determine the decimal representation of a binary number, simply take the sum of the products of the binary digits and the powers of 2 which they represent.

1101 ??
Homogeneous: $T$ is kept constant in the inner loop. It is decreased only in the outer loop.

Non-homogeneous: There is only temperature loop.
Threshold accepting [Dueck and Scheuer (1990)]:

- Acceptance probability

\[ P(s', s) = \begin{cases} 
1 & \text{if } \triangledown E \leq Q_{value} \\
0 & \text{otherwise} 
\end{cases} \]

- Update \( Q_{value} \) according to an annealing schedule.
- Faster than Simulated annealing
**Record-to-record**

*Record-to-record travel algorithm:*

*RECORD*: objective value of the best found solution.

---

**Template of the record-to-record travel algorithm.**

**Input:** Deviation $D > 0$.

`s = s_0 ; /* Generation of the initial solution */`

`RECORD = f(s) ; /* Starting RECORD */`

**Repeat**

- Generate a random neighbor $s'$ :
  - If $f(s') < RECORD + D$ Then $s = s'$ ; /* Accept the neighbor solution */
  - If $RECORD > f(s')$ Then $RECORD = f(s')$ ; /* RECORD update */

**Until** Stopping criteria satisfied

**Output:** Best solution found.
Great deluge algorithm [Dueck (1993)]:

- The climber will try to reach at the top (global optima position).
- The climber will try to keep his/her foot above the water level.
Input: Water Level;
\( s = s_0 \)  
Generation of the initial solution ;
Choose the rain speed \( UP \);
Choose the initial water level ;

**Repeat**

Generate a neighbour solution \( s' \);
if \( f(s') > Level \), then \( s = s' \);
Level = Level + UP;

**Until** (stopping criteria)

Output: Best found solution

**Table:** Pseudocode of Great deluge algorithm

Input: $s^{(o)}$ - the initial solution;
$s = \text{local search} \ (s^{(o)});$  
Repeat
$ s' = \text{Mutate} \ (s);$  
$ s'' = \text{Local Search} \ (s');$  
if $f(s'') < f(s)$
  set $s = s''$ ;  
end if ;  
Until (stopping condition is satisfied) ;  
Return $s$;

Table: Pseudocode of Iterated Local Search

Figure: Perturbation principle
Key issues

- Define an initial solution.
- Mutation technique.
- Local search method.
Mutation/Perturb

- For binary string: flip operator.
- For VRPs: 2-pot, reinsert, swap.
Input: Neighborhood structures $N_l \quad l = 1, 2, 3, \ldots, l_{\text{max}}$;
Generate the initial solution $s^{(o)}$;
set: $s = s^{(o)}$;
$l = 1$;
while ($l \leq l_{\text{max}}$)
  Find the best neighbor $s'$ of $s$ in $N_l$;
  if $f(s') < f(s)$
    set $s = s'$; $l = 1$;
  else
    $l = l + 1$;
  end if;
end while;
Return $s$;

Table: Pseudocode of Variable Neighborhood Decent

Figure: Variable neighbourhoods
Template of the basic variable neighborhood search algorithm.

**Input:** a set of neighborhood structures $N_k$ for $k = 1, \ldots, k_{\text{max}}$ for shaking.

$x = x_0$ ; /* Generate the initial solution */

**Repeat**

$k = 1$

**Repeat**

Shaking: pick a random solution $x'$ from the $k^{th}$ neighborhood $N_k(x)$ of $x$ ;

$x'' = \text{local search}(x')$

**If** $f(x'') < f(x)$ **Then**

$x = x''$

Continue to search with $N_1 ; k = 1$

**Otherwise** $k = k + 1$

**Until** $k = k_{\text{max}}$

**Until** Stopping criteria

**Output:** Best found solution.
Components

Shaking:

- Nested neighbourhood structures:
  - n-flip \((n = 1, 2, \ldots)\).
  - k-opt \((k = 2, 3, \ldots)\)
  - \(\lambda\)-interchange \((\lambda = 1, 2, \ldots)\)

- Usually, neighbourhoods are ordered from smallest to largest.

Diversification: by shaking
Intensification: by local search
Variations

The order of the neighbourhoods:

- Forward VNS: start with $k = 1$ and increase.
- Backward VNS: start with $k = k_{\text{max}}$ and decrease.
- start with $k = k_{\text{min}}$, and increase $k$ by $k_{\text{step}}$ if no improvement.
Variations

Accepting worse solutions:

- With some probability.
- Skewed VNS: Accept if $f(x'') - \alpha \times d(x, x'') < f(x)$ [$d(x, x'')$ measures the distance between solutions].
Variations

Others:

- Reduced VNS: same as the Basic VNS, but no Local Search procedure.
- Variable Neighbourhood Decomposition Search: fix some components of the solution, and perform Local Search on the remaining free components.
Template of the guided local search algorithm.

**Input:** S-metaheuristic LS, λ, Features f, Costs c.

\[ s = s_0 \] /* Generation of the initial solution */

\[ p_i = 0 \] /* Penalties initialization */

**Repeat**

Apply a S-metaheuristic LS; /* Let \( s^* \) the final solution obtained */

**For** each feature \( i \) of \( s^* \) **Do**

\[ u_i = \frac{c_i}{1+p_i} \] /* Compute its utility */

\[ u_j = \max_{i=1,\ldots,m}(u_i) \] /* Compute the maximum utilities */

\[ p_j = p_j + 1 \] /* Change the objective function by penalizing the feature \( j \) */

**Until** Stopping criteria /* e.g. max number of iterations or time limit */

**Output:** Best solution found.

\[
    f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i I_i(s)
\]

\[
    I_i(s) = \begin{cases} 
        1 & \text{if the feature } f t_i \in s \\
        0 & \text{otherwise}
    \end{cases}
\]
Components

\( \lambda \): It dictates the influence of the penalty on the extended move evaluation function

- Low value: intensification
- High value: diversification

\[
\lambda = \frac{f(s^*)}{\text{avg. no. of features in } s^*}
\] (4.1)

Features and costs (For VRPs): Edges and their lengths.
Greedy Randomized Adaptive Search Procedure

A template of the greedy randomized adaptive search procedure.

**Input:** Number of iterations.

**Repeat**

\[ s = \text{Random-Greedy}(\text{seed}) \] /* apply a randomized greedy heuristic */

\[ s' = \text{Local} - \text{Search}(s) \] /* apply a local search algorithm to the solution */

**Until** Stopping criteria /* e.g. a given number of iterations */

**Output:** Best solution found.
Greedy construction

RCL:
- Consider p-best elements
- Consider an element $e_i$ with cost $c_i$ if

\[ c_i \leq c_{\text{min}} + \alpha \times (c_{\text{max}} - c_{\text{min}}) \]
\( \alpha \) parameter \( \in [0, 1] \):
- \( \alpha = 0 \) : pure greedy
- \( \alpha = 1 \) : pure random

\[ c_i \leq c_{min} + \alpha \times (c_{max} - c_{min}) \quad e_i \in RCL \] (5.1)
• It is also known as *Ruin and Create*.
• It is capable of exploring a large number of neighbour solutions.
• Given a solution, LNS *Destroys* some part of it and then *Repairs* greedily.
Components

Destroy:

- Can be done randomly.
- Design a heuristic method.

Repair:

- GRASP concept can be used.
- Design another heuristic procedure.

How to choose heuristics if more than one operator is being used?

- Pick any one randomly
- Use the search information [Pisinger and Ropke (2006)]:

\[
    w_{i,j+1} = w_{i,j} \times (1 - r) + r \times \frac{\pi_i}{\theta_i}
\]

\(w_{i,j}\) is the weight associated with heuristic \(i\) at \(j^{th}\) segment of the search. The \(\pi_i\) is score of heuristic \(i\) obtained during the last segment of the search, \(r\) is a reaction factor and \(\theta_i\) counts the number of times heuristic \(i\) was used.
Hybrid

---

Input: $s^{(o)}$ - the initial solution;
$s = \text{local search} \left( s^{(o)} \right)$;
Repeat
  Choose a destroy and a repair heuristic;
  $s' = \text{Repair} \left( \text{Destroy} \left( s \right) \right)$;
  $s'' = \text{Local Search} \left( s' \right)$;
  if $f(s'') < f(s)$
    set $s = s''$ ;
  end if ;
Until (stopping condition is satisfied) ;
Return $s$;

**Table:** Pseudocode of Iterated Local Search with LNS
Path relinking

Initiating solution

Guiding solution

Original path

Relinked path

Figure: Relinking solutions
Path relinking

Figure: Multiple guiding solutions
Path relinking

Path selection rules:

- Minimizing the distance between solutions
  - solution quality
  - Search history

Re-linking methods:

- Forward (start $\rightarrow$ final)
- Backward (final $\rightarrow$ start)
- Back and forward (start $\leftrightarrow$ final)
- Mixed (start $\leftrightarrow$ final, but with an intermediate guiding solution)
Path relinking

Figure: Path relinking research

<table>
<thead>
<tr>
<th>Years</th>
<th>Number of Articles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990-1995</td>
<td>1</td>
</tr>
<tr>
<td>1996-2000</td>
<td>5</td>
</tr>
<tr>
<td>2001-2004</td>
<td>12</td>
</tr>
</tbody>
</table>

Santosh Kumar Mandal, Ph.D research fellow

Meta-Heuristics
Path relinking

Applications:

- GRASP with evolutionary path relinking (http://www.sciencedirect.com/science/article/pii/S0305054811003029)
- GRASP with path relinking (http://www2.research.att.com/~mgcr/doc/sgrasprr.pdf)
- Scatter search with path relinking (http://leeds-faculty.colorado.edu/glover/SS-PR%20Template.pdf)

Future research:

- Selection of initiating and guiding solutions.
- Application of local search to intermediate solutions
- Testing of standalone PR procedures.
Vehicle Routing Problem (VRP)

Given a fleet of vehicles based at a central depot, the basic VRP [Dantzig and Ramser (1959)] aims to design a set of tours for the vehicles to service a given set of geographically dispersed customers.

- **Objectives:**
  - Minimization of the overall tour cost
  - Minimization of the makespan

- **Constraints:**
  - Each vehicle tour should start and end at the depot node.
  - Vehicle loading capacity should not violate on any tour.
  - Each customer must be served only once by a single vehicle.
VRP: solution representation

Complete VRP solution

Giant tour

Santosh Kumar Mandal, Ph.D research fellow

Meta-Heuristics
VRP: neighbourhood operators

**Figure:** 2-opt illustration

**Figure:** Reinsert illustration


