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

Santosh Kumar Mandal, *Ph.D research fellow*

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

January 16, 2014

Outline

$$max\ f(x) = x^3 - 60 \times x^2 + 900 \times x + 100 \tag{1.1}$$

$$Global\ maxima : f(x) = 4100\ at\ x = 10 \tag{1.2}$$

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

T	Move	Solution	f	Δf	Move?	New Neighbor Solution
500	1	00011	2287	112	Yes	00011
450	3	00111	3803	<0	Yes	00111
405	5	00110	3556	247	Yes	00110
364.5	2	01110	3684	<0	Yes	01110
328	4	01100	3998	<0	Yes	01100
295.2	3	01000	3972	16	Yes	01000
265.7	4	01010	**4100**	<0	Yes	01010
239.1	5	01011	4071	29	Yes	01011
215.2	1	11011	343	3728	No	01011

Parameter settings

- T_{max}: Check 100 - 500 K
- T_{min}: $= 0.01$ K
- *Length of inner loop*: Check 100 - 500
- Cooling schedule:
  - Linear.
    $T = T - \beta \qquad \beta < 1$
  - Geometric
    $T = T \times \alpha \qquad \alpha < 1$
  - Decide values of α and β on your own.
- Acceptance probability for bad solutions:
  $\exp(\frac{-\triangle}{T}) > rand(.)$ $rand(.)$: 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

Threshold accepting [Dueck and Scheuer (1990)]:

- Acceptance probability

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

- 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.

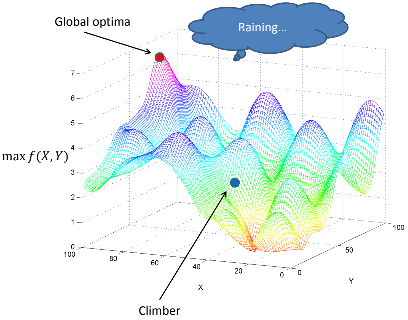
[redacted] Template of the record-to-record travel algorithm.

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

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.



Continue...

Input: Water Level;
$s = s_o$ 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;
$s =$ local search $(s^{(o)})$;
Repeat
$s' =$ Mutate (s);
$s'' =$ 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

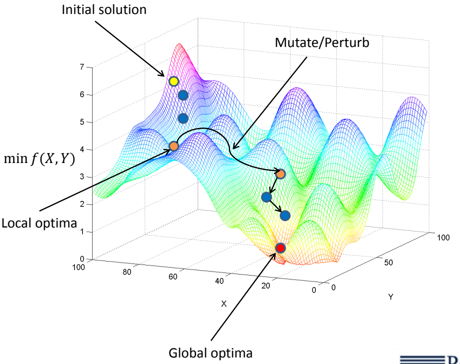


Figure: Perturbation principle

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 $\quad l = 1, 2, 3, \ldots, l_{max}$;
Generate the initial solution $s^{(o)}$;
set: $s = s^{(o)}$;
$l = 1$;
while $(l \leq l_{max})$
    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

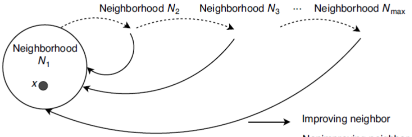


Figure: Variable neighbourhoods

[redacted] Template of the basic variable neighborhood search algorithm.

Input: a set of neighborhood structures N_k for $k = 1, \ldots, k_{max}$ 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'' =local search(x') ;
    If $f(x'') < f(x)$ **Then**
      $x = x^{''}$;
      Continue to search with N_1 ; $k = 1$;
    Otherwise k=k+1 ;
  Until $k = k_{max}$
Until Stopping criteria
Output: Best found solution.

Components

Shaking:

- Nested neighbourhood structures:
  - n-flip ($n = 1, 2, \dots$).
  - k-opt ($k = 2, 3, \dots$)
  - λ-interchange ($\lambda = 1, 2, \dots$)
- 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 $k = k_{max}$ and decrease.
- start with $k = k_{min}$, and increase k by k_{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.

working process

[redacted] Template of the guided local search algorithm.

Input: S-metaheuristic LS, λ, Features I, 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 */
  $u_j = max_{i=1,..m}(u_i)$; /* 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.

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

$$I_i(s) = \begin{cases} 1 & \text{if the feature } ft_i \in s \\ 0 & \text{otherwise} \end{cases}$$

Components

λ: It dictates the influence of the penalty on the extended move evaluation function

- Low value: intensification
- High value: diversification

$$\lambda = \frac{f(s^*)}{avg.\ no.\ of\ features\ in\ s^*} \tag{4.1}$$

Features and costs(For VRPs): Edges and their lengths.

Greedy Randomized Adaptive Search Procedure

[redacted] Template of the greedy randomized adaptive search procedure.

Input: Number of iterations.
Repeat
  s = Random-Greedy(seed) ; /* apply a randomized greedy heuristic */
  $s' = Local - Search(s)$; /* 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

$$c_i \leq c_{min} + \alpha \times (c_{max} - c_{min})$$

[redacted] Template of the greedy randomized algorithm.

$s = \{\}$; /* Initial solution (null) */
Evaluate the incremental costs of all candidate elements ;
Repeat
  Build the restricted candidate list *RCL* ;
  /* select a random element from the list *RCL* */
  e_i = Random-Selection(*RCL*) ;
  If $s \cup e_i \in F$ **Then** /* Test the feasibility of the solution */
  $s = s \cup e_i$;
  Reevaluate the incremental costs of candidate elements ;
Until Complete solution found.

Components

α parameter $\in [0, 1]$:

- $\alpha = 0$: pure greedy
- $\alpha = 1$: pure random

$$c_i \leq c_{min} + \alpha \times (c_{max} - c_{min}) \qquad e_i \in RCL \tag{5.1}$$

- 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.

http://www.diku.dk/~sropke/Papers/PDPTW_techRep.pdf

Components

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

$\gg$ Pick any one randomly

$\gg$ Use the search information [Pisinger and Ropke (2006)]:

$$w_{i,j+1} = w_{i,j} \times (1 - r) + r \times \frac{\pi_i}{\theta_i} \tag{6.1}$$

$w_{i,j}$ is the weight associated with heuristic i at j^{th} segment of the search. The π_i is score of heuristic i obtained during the last segment of the search, r is a reaction factor and θ_i counts the number of times heuristic i was used.

Hybrid

Input: $s^{(o)}$ - the initial solution;
$s =$ local search $(s^{(o)})$;
Repeat
Choose a destroy and a repair heuristic;
$s' =$ Repair (Destroy (s));
$s'' =$ 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 with LNS

Path relinking

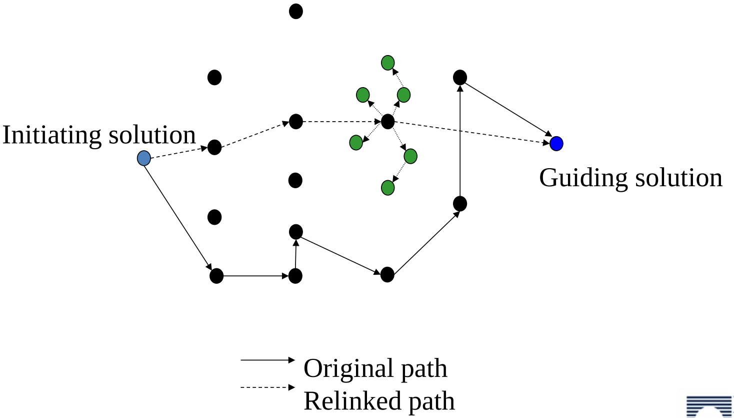


Figure: Relinking solutions

Path relinking

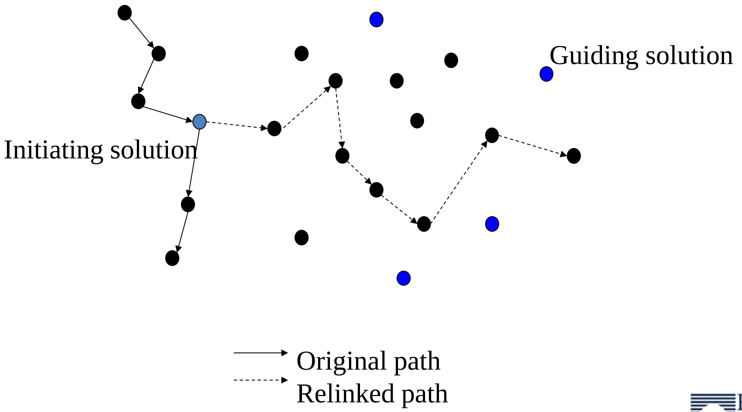


Figure: Multiple guiding solutions

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

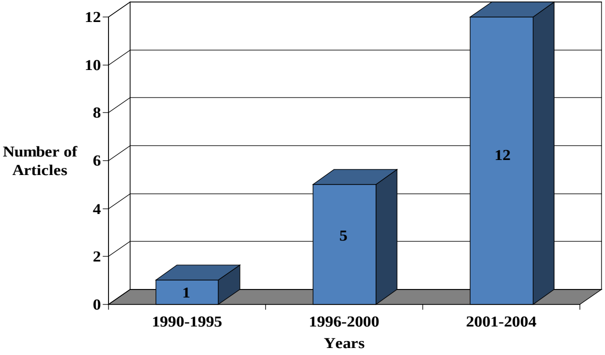


Figure: Path relinking research

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/sgrasppr.pdf)
- Scatter search with path relinking (http://leeds-faculty.colorado.edu/glover/SS-PR%20Template.pdf)

Future research:

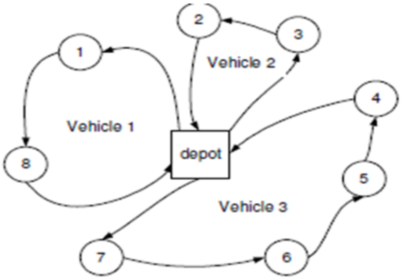
- Selection of initiating and guiding solutions.
- Application of local search to intermediate solutions
- Testing of standalone PR procedures.

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



0	1	8	0	3	2	0	7	6	5	4	0

Complete VRP solution

↓

1	8	3	2	7	6	5	4

Giant tour

VRP: neighbourhood operators

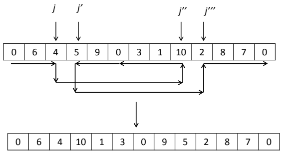


Figure: 2-opt illustration

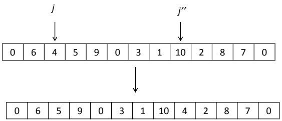


Figure: Reinsert illustration

Dantzig, G. B. and Ramser, J. M. (1959). The truck dispatching problem. *Management Science*, 6(1):81–91.

Dueck, G. (1993). The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92.

Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175.

Pisinger, D. and Ropke, S. (2006). An adaptive large neighborhood search heuristic for the pick up and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Taillard, É. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673.