University of Ljubljana, Faculty of Computer and Information Science

Metaheuristics

Prof Dr Marko Robnik-Šikonja

Analysis of Algorithms and Heuristic Problem Solving
Version 2023

What is a metaheuristic?

- procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem

- Examples:
  - tabu search
  - guided local search
  - variable neighborhood search

Literature

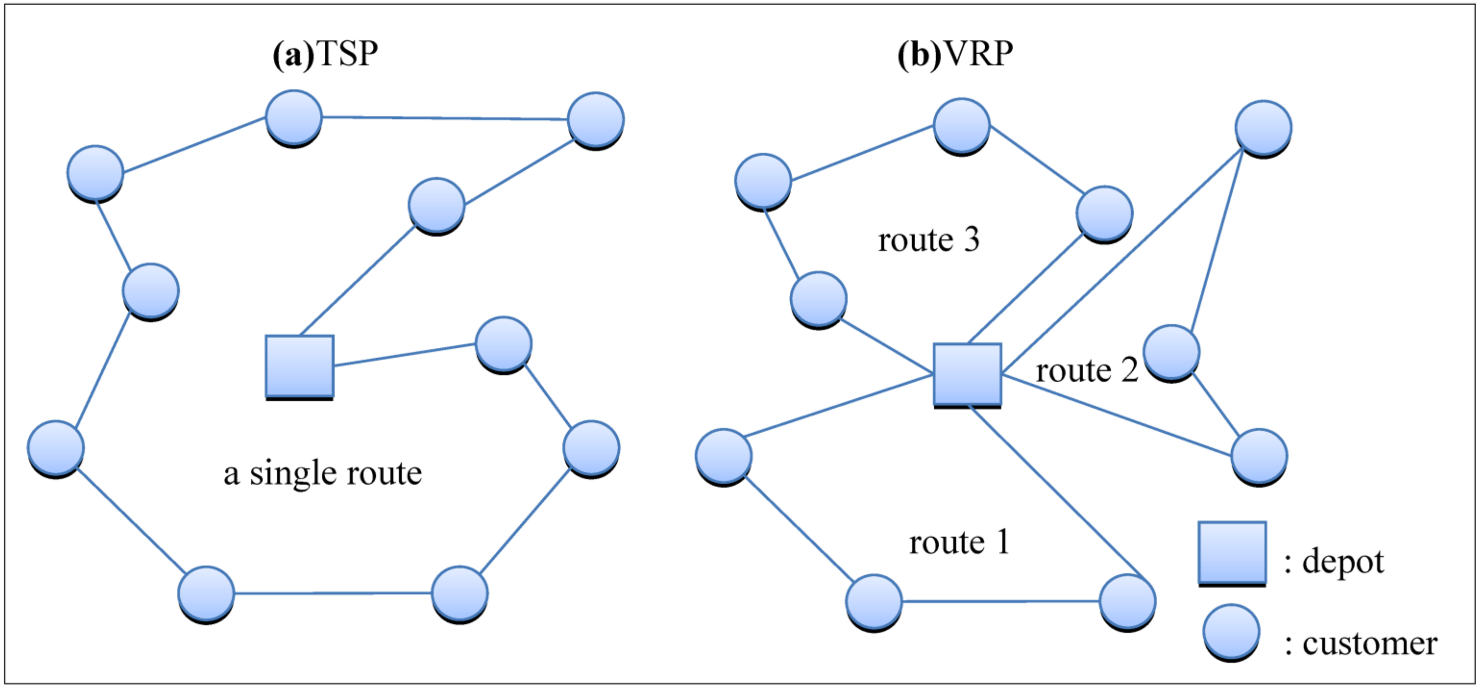
- Blum, Christian, and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison." *ACM computing surveys (CSUR)* 35, no. 3 (2003): 268-308.
- M. Gendreau, J.-Y. Poitvin (Eds): Handbook of Metaheuristics. Springer Verlag, 2010

Classification of metaheuristics

- Nature-inspired vs. non-nature inspired
- Population-based vs. single point search
- Dynamic vs. static objective function
- One vs. various neighborhood structures
- Memory usage vs. memory-less methods

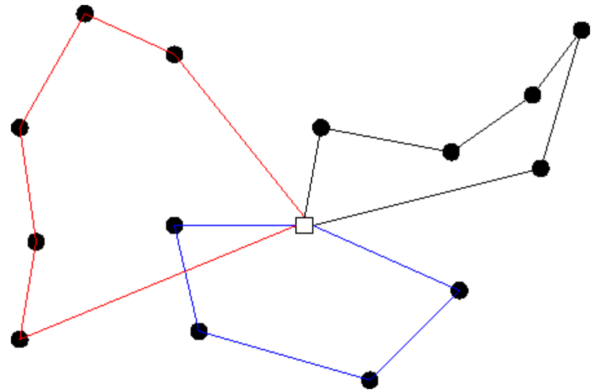
Vehicle routing problem (VRP)

- generalization of TSP
- G=(V,E)
  - one vertex is depot, where a fleet of m identical vehicles of capacity Q is based
  - other vertices are customers which need to be served



Vehicle routing problem

- each vertex has associated demand q_i and service time t_i
- each edge has a cost $c_{i,j}$ and travel time $t_{i,j}$
- Task is to find a set of routes such that minimizes the cost of all routes subject to
  - each route begins and ends at the depot
  - each customer is visited only once by only one route
  - the demand on each route does not exceed Q
  - the total duration of each route (travel + service) does not exceed L



Extensions of VRP

- Vehicle Routing Problem with Pickup and Delivery (VRPPD): A number of goods need to be moved from certain pickup locations to other delivery locations. The goal is to find optimal routes for a fleet of vehicles to visit the pickup and drop-off locations.
- Vehicle Routing Problem with stack: Similar to the VRPPD, except an additional restriction is placed on the loading of the vehicles: at any delivery location, the item being delivered must be the item most recently picked up. This scheme reduces the loading and unloading times at delivery locations because there is no need to temporarily unload items other than the ones that should be dropped off.
- Vehicle Routing Problem with Time Windows (VRPTW): The delivery locations have time windows within which the deliveries (or visits) must be made.
- Capacitated Vehicle Routing Problem: CVRP or CVRPTW. The vehicles have limited carrying capacity of the goods that must be delivered.
- Vehicle Routing Problem with Multiple Trips (VRPMT): The vehicles can do more than one route.
- Open Vehicle Routing Problem (OVRP): Vehicles are not required to return to the depot.

Tabu search

- prevent cycling and retuning back to the same local extreme
- idea: suppress solutions or parts of solutions, i.e. add them to the tabu list

Tabu search pseudocode

```
s ← GenerateInitialSolution()
TabuList ← ∅
while termination conditions not met do
    s ← ChooseBestOf(N(s) \ TabuList)
    Update(TabuList)
endwhile
```

```
s ← GenerateInitialSolution()
InitializeTabuLists(TL_1, ..., TL_r)
k ← 0
while termination conditions not met do
    AllowedSet(s, k) ← {s' ∈ N(s) | s does not violate a tabu condition,
                                     or it satisfies at least one aspiration condition}
    s ← ChooseBestOf(AllowedSet(s, k))
    UpdateTabuListsAndAspirationConditions()
    k ← k + 1
endwhile
```

Variants and improvements of tabu search

- probabilistic TS:
  - use sampling from neighbourhood, or
  - probabilistically activate the tabu criteria
- intensification:
  - intensify search in the neighbourhood of good solutions
  - e.g., in VRP,
  - maintain an intermediate memory for presence of edges, fix long present edges, thoroughly search through the others
  - change the neighbourhood, etc.
- diversification:
  - broaden the search
  - long term memory for presence of parts of solutions
  - use restart or a punishment term for the long-lasting components
- allow infeasible solutions,
  - relaxation of the problem and/or adding penalty terms for violation of constraints
- surrogate objectives
  - if fitness function is computationally costly
- auxiliary objectives
  - to bias search, e.g., towards less vehicles, or to add preference for low number of clients on a route
- hybridization (combination with other techniques)

Guided local search

- metaheuristics which guides local search and helps it to avoid local extremes
- define properties (attributes) of solutions
- penalize attributes, which occur too often in local extrema
- define auxiliary objective function

$$h(s) = g(s) + \lambda \times \sum_{i \text{ is a feature}} (p_i \times I_i(s))$$

- h(s) = auxiliary fitness function
- g(s) = fitness function
- p_i = punishment for i-th property
- $I_i(s)$ = an indicator function for attribute i and state s
- λ = weight of punishments
- determine the utility of punishments

GLS: utility of punishments

- utility of punishment for property i in local extreme s*

$$\text{util}_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i}$$

- where c_i is cost
- p_i is current punishment for property i
- in local extreme, we punish the property with the largest utility util_i, i.e. we increment p_i by 1

Pseudo code for GLS

p=problem
M=number of features

```
procedure GuidedLocalSeach(p, g, λ, [I_1, ..., I_M], [c_1, ..., c_M], M)
begin
    k ← 0;
    s_0 ← ConstructionMethod(p);
    /* set all penalties to 0 */
    for i ← 1 until M do
        p_i ← 0;
    /* define the augmented objective function */
    h ← g + λ * Σ p_i * I_i;
    while StoppingCriterion do
    begin
        s_{k+1} ← ImprovementMethod(s_k, h);
        /* compute the utility of features */
        for i ← 1 until M do
            util_i ← I_i(s_{k+1}) * c_i / (1 + p_i);
        /* penalize features with maximum utility */
        for each i such that util_i is maximum do
            p_i ← p_i + 1;
        k ← k + 1;
    end
    s* ← best solution found with respect to objective function g;
    return s*;
end
```

Guided Fast Local Search

- define different neighborhoods, and label them active (1) or inactive (0)
- only investigate active neighborhoods
- make a feature from activity of neighborhoods
- use GLS on these features

Workforce scheduling problem

- assign a number of engineers to a set of jobs minimizing a total cost
- job (Location , duration, type)
- engineer: (location, start time, end time, overtime limit, skill factor)
- cost = traveling cost + overtime cost + job cost

Variable neighborhood search

- idea: define several neighborhood structures and change neighborhood when reaching local extreme in one of them
- order neighborhoods by the efficiency of computation

VND pseudocode

Algorithm 2 Variable neighborhood descent

Function VND (x, k_{max})

$k \leftarrow 1$

repeat

  $x' \leftarrow arg\min_{y \in N_k(x)} f(y)$ // Find the best neighbor in $N_k(x)$

  $x, k \leftarrow$ `NeighborhoodChange` (x, x', k) // Change neighborhood

until $k = k_{max}$

return x

Algorithm 1 Neighborhood change

Function `NeighborhoodChange` (x, x', k)

if $f(x') < f(x)$ **then**

  $x \leftarrow x'$ // Make a move

  $k \leftarrow 1$ // Initial neighborhood

else

  $k \leftarrow k + 1$ // Next neighborhood

return x, k

General VNS pseudocode

Algorithm 4 Shaking function

Function Shake(x, k)
- $w \leftarrow \lceil 1 + \texttt{Rand}(0,1) \times |\mathcal{N}_k(x)| \rceil$
- $x' \leftarrow x^w$

return x'

Algorithm 8 General VNS

Function GVNS $(x, \ell_{max}, k_{max}, t_{max})$
- **repeat**
  - $k \leftarrow 1$
  - **repeat**
    - $x' \leftarrow \texttt{Shake}(x, k)$
    - $x'' \leftarrow \texttt{VND}(x', \ell_{max})$
    - $x, k \leftarrow \texttt{NeighborhoodChange}(x, x'', k)$
  - **until** $k = k_{max}$
  - $t \leftarrow \texttt{CpuTime()}$
- **until** $t > t_{max}$

return x

Scatter Search and Path-Relinking

- Scatter search: systematic diversification strategy
- Path-Relinking: systematic intensification strategy

Summary of metaheuristics

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- More advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Tips in LS and metaheuristics

- Learn the problem well
- Collect statistics on performance, neighborhoods
- Learn from statistics
- Consider penalizing constraints
- Consider different neighborhood structures
- Experiment with parameters
- Select a good set of benchmark instances
- Calibrate a method to your set of instances and tune parameters