Metaheuristics

Prof Dr Marko Robnik-Šikonja

Analysis of Algorithms and Heuristic Problem Solving
December 2019
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
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 end 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 tabu list
Tabu search pseudocode

```plaintext
s ← s0
sBest ← s
tabuList ← []
while (not stoppingCondition())
    candidateList ← []
    bestCandidate ← null
    for (sCandidate in sNeighborhood)
        if ((not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)))
            bestCandidate ← sCandidate
    end
    s ← bestCandidate
    if (fitness(bestCandidate) > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate);  // better: part of bestCandidate
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```
**Variants and improvements of tabu search**

* probabilistic TS:
  * use sampling from neighborhood, or
  * probabilistically activate the tabu criteria

* intensification:
  * intensify search in the neighborhood 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 neighborhood, etc

* diversification:
  * broaden the search
  * long term memory for presence of parts of solutions
  * Use restart or a punishment term for the longlasting 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,
  * 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 \(s\)
* \(\lambda\) = weight of punishments
* determine 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
procedure GuidedLocalSearch\((p, g, \lambda, [I_1, \ldots, I_M], [c_1, \ldots, c_M], M)\) begin

\[ k \leftarrow 0; \]
\[ s_0 \leftarrow \text{ConstructionMethod}(p); \]
/* set all penalties to 0 */
for \( i \leftarrow 1 \) until \( M \) do

\[ p_i \leftarrow 0; \]
/* define the augmented objective function */
\[ h \leftarrow g + \lambda \times \sum p_i \times I_i; \]
while StoppingCriterion do begin

\[ s_{k+1} \leftarrow \text{ImprovementMethod}(s_k, h); \]
/* compute the utility of features */
for \( i \leftarrow 1 \) until \( M \) do

\[ \text{util}_i \leftarrow I_i(s_{k+1}) \times c_i / (1 + p_i); \]
/* penalize features with maximum utility */
for each \( i \) such that \( \text{util}_i \) is maximum do

\[ p_i \leftarrow p_i + 1; \]
\[ k \leftarrow k + 1; \]
end
\[ s^* \leftarrow \text{best solution found with respect to objective function } g; \]
return \( s^*; \)
end
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
Algorithm 2 Variable neighborhood descent

Function $\text{VND}(x, k_{\text{max}})$

1. $k \leftarrow 1$
2. repeat
3. \hspace{1em} $x' \leftarrow \arg\min_{y \in N_k(x)} f(y)$ // Find the best neighbor in $N_k(x)$
4. \hspace{1em} $x, k \leftarrow \text{NeighborhoodChange}(x, x', k)$ // Change neighborhood
5. until $k = k_{\text{max}}$
6. return $x$

Algorithm 1 Neighborhood change

Function $\text{NeighborhoodChange}(x, x', k)$

1. if $f(x') < f(x)$ then
2. \hspace{1em} $x \leftarrow x'$ // Make a move
3. \hspace{1em} $k \leftarrow 1$ // Initial neighborhood
4. else
5. \hspace{1em} $k \leftarrow k + 1$ // Next neighborhood
6. return $x, k$
Algorithm 4 Shaking function

Function $\text{Shake}(x, k)$
1. $w \leftarrow [1 + \text{Rand}(0, 1) \times |\mathcal{N}_k(x)|]$
2. $x' \leftarrow x^w$
return $x'$

Algorithm 8 General VNS

Function $\text{GVNS}(x, \ell_{max}, k_{max}, t_{max})$
1. repeat
2. $k \leftarrow 1$
3. repeat
4. $x' \leftarrow \text{Shake}(x, k)$
5. $x'' \leftarrow \text{VND}(x', \ell_{max})$
6. $x, k \leftarrow \text{NeighborhoodChange}(x, x'', k)$
until $k = k_{max}$
7. $t \leftarrow \text{CpuTime}()$
until $t > t_{max}$
return $x$
Tips in LS and metaheuristics

* learn the problem well
* collect statistics on performance, neighborhoods
* learn fromm statistics
* consider penalizing constraints
* consider different neighborhood structures
* experiment with parameters
* select a good set of benchmark instances
* calibrate method to your set of instances and tune the parameters
* M. Gendreau, J.-Y. Poitvin (Eds): Handbook of Metaheuristics. Springer Verlag, 2010