Variable neighborhood search for the travelling salesman problem and its variants

Nenad Mladenović,

Brunel University-West London UK

FON, MI-SANU, University of Belgrade

- Introduction
- Variable neighborhood search algorithms
  - ▷ Variable metric algorithm;
  - ▷ VND;
  - ▷ Reduced VNS;
  - ▷ Basic VNS;
  - ▷ Skewed VNS;
  - ▷ General VNS;
  - ▷ More extensions
  - ▷ VN Decomposition search;
  - ▷ Primal-Dual VNS

Introduction

- Optimization problems (continuous-discrete, static-dynamic, deterministic-stochastic)
- Exact methods, Heuristics, Simulation (Monte-Carlo)
- Classical heuristics (constructive (greedy add, greedy drop), relaxation based, space reduction, local search, Lagrangian heuristics,...)
- Metaheurestics (Simulated annealing, Tabu search, GRASP, Variable neighborhood search, Genetic search, Evolutionary methods, Particle swarm optimization, ...)
- Variable neighborhood search

Optimization problems

A deterministic optimization problem may be formulated as

$$\min\{f(x)|x \in X, X \subseteq \mathcal{S}\}, \tag{1}$$

- where $\mathcal{S}, X, x$ and f denote the solution space, the feasible set, a feasible solution and a real-valued objective function, respectively.
- If $\mathcal{S}$ is a finite but large set, a combinatorial optimization problem is defined.
- If $\mathcal{S} = R^n$, we refer to continuous optimization.
- A solution $x^* \in X$ is optimal if

$$f(x^*) \leq f(x), \ \forall x \in X.$$

- An exact algorithm for problem (1), if one exists, finds an optimal solution x^*, together with the proof of its optimality, or shows that there is no feasible solution, i.e., $X = \emptyset$, or the solution is unbounded.
- For continuous optimization, it is reasonable to allow for some degree of tolerance, i.e., to stop when sufficient convergence is detected.

Variable metric algorithm

Assume that the function $f(x)$ is approximated by its Taylor series

$$f(x) = \frac{1}{2}x^T A x - b^T x$$

$$x_{i+1} - x_i = -H_{i+1}(\nabla f(x_{i+1}) - \nabla f(x_i)).$$

Function `VarMetric(x)`
let $x \in R^n$ be an initial solution
$H \leftarrow I$; $g \leftarrow -\nabla f(x)$
for $i = 1$ `to` n **do**

$\alpha^* \leftarrow arg\min_\alpha f(x+\alpha \cdot Hg)$
$x \leftarrow x + \alpha^* \cdot Hg$
$g \leftarrow -\nabla f(x)$
$H \leftarrow H + U$
end

Local search

Function `BestImprovement(x)`

repeat

$x' \leftarrow x$

$x \leftarrow arg\min_{y \in N(x)} f(y)$

until $(f(x) \geq f(x'))$;

Function `FirstImprovement(x)`

```
repeat
```

$x' \leftarrow x$; $i \leftarrow 0$

```
repeat
```

$i \leftarrow i + 1$

$x \leftarrow arg\min\{f(x), f(x_i)\}$, $x_i \in N(x)$

`until` $(f(x) < f(x_i)$ `or` $i = |N(x)|)$;

`until` $(f(x) \geq f(x'))$;

Variable neighborhood search

- Let $\mathcal{N}_k$, $(k = 1, \ldots, k_{max})$, a finite set of pre-selected neighborhood structures,
- $\mathcal{N}_k(x)$ the set of solutions in the k^{th} neighborhood of x.
- Most local search heuristics use only one neighborhood structure, i.e., $k_{max} = 1$.
- An optimal solution x_{opt} (or global minimum) is a feasible solution where a minimum is reached.
- We call $x' \in X$ a local minimum with respect to $\mathcal{N}_k$ (w.r.t. $\mathcal{N}_k$ for short), if there is no solution $x \in \mathcal{N}_k(x') \subseteq X$ such that $f(x) < f(x')$.
- Metaheuristics (based on local search procedures) try to continue the search by other means after finding the first local minimum. VNS is based on three simple facts:
  - ▷ A local minimum w.r.t. one neighborhood structure is not necessarily so for another;
  - ▷ A global minimum is a local minimum w.r.t. all possible neighborhood structures;
  - ▷ For many problems, local minima w.r.t. one or several $\mathcal{N}_k$ are relatively close to each other.

Variable neighborhood search

- In order to solve optimization problem by using several neighborhoods, facts 1 to 3 can be used in three different ways:
  - (i) deterministic;
  - (ii) stochastic;
  - (iii) both deterministic and stochastic.

- Some VNS variants
  - Variable neighborhood descent (VND) (sequential, nested)
  - Reduced VNS (RVNS)
  - Basic VNS (BVNS)
  - Skewed VNS (SVNS)
  - General VNS (GVNS)
  - VN Decomposition Search (VNDS)
  - Parallel VNS (PVNS)
  - Primal Dual VNS (P-D VNS)
  - Reactive VNS
  - Backward-Forward VNS
  - Best improvement VNS
  - Exterior point VNS
  - VN Simplex Search (VNSS)

- VN Branching
- VN Pump
- Continuous VNS
- Mixed Nonlinear VNS (RECIPE), etc.

Neighborhood change

Function `NeighbourhoodChange` (x, x', k)
if $f(x') < f(x)$ **then**

  $x \leftarrow x'$; $k \leftarrow 1$ /* Make a move */
else

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

Reduced VNS

Function $\texttt{RVNS}\ (x, k_{max}, t_{max})$

```
repeat
    k ← 1
    repeat
        x' ← Shake(x, k)
        NeighborhoodChange (x, x', k)
    until k = k_max;
    t ← CpuTime()
until t > t_max;
```

- RVNS is useful in very large instances, for which local search is costly.
- It has been observed that the best value for the parameter k_{max} is often 2.
- The maximum number of iterations between two improvements is usually used as a stopping condition.
- RVNS is akin to a Monte-Carlo method, but is more systematic

- When applied to the p-Median problem, RVNS gave solutions as good as the Fast Interchange heuristic of Whitaker while being 20 to 40 times faster.

VND

Function `VND` (x, k'_{max})

repeat

  $k \leftarrow 1$

  repeat

    $x' \leftarrow arg\min_{y \in \mathcal{N}'_k(x)} f(x)$ /* Find the best neighbor in $\mathcal{N}_k(x)$ */

    `NeighbourhoodChange` (x, x', k) /* Change neighbourhood */

  until $k = k'_{max}$;

until no improvement is obtained;

Sequential VND

Function `Seq-VND`(x, ℓ_{max})
$\ell \leftarrow 1$ `// Neighborhood counter`
repeat
  $i \leftarrow 0$ `// Neighbor counter`
  repeat
    $i \leftarrow i + 1$
    $x' \leftarrow arg\min\{f(x), f(x_i)\}$, $x_i \in N_\ell(x)$ `// Compare`
  until $(f(x') < f(x)$ `or` $i = |N_\ell(x)|)$
  $\ell, x \leftarrow$ `NeighborhoodChange` (x, x', ℓ); `// Neighborhood change`
until $\ell = \ell_{max}$

- The final solution of `Seq-VND` should be a local minimum with respect to all ℓ_m neighborhoods.
- The chances to reach a global minimum are larger than with a single neighborhood structu
- The total size of `Seq-VND` is equal to the union of all neighborhoods used.

- If neighborhoods are disjoint (no common element in any two) then the following holds

$$|\mathcal{N}_{\mathtt{Seq-VND}}(x)| = \sum_{\ell=1}^{\ell_{max}} |\mathcal{N}_\ell(x)|, \ x \in X.$$

Nested VND

- Assume that we define two neighborhood structures ($\ell_{max} = 2$). In the nested VND we
fact perform local search with respect to the first neighborhood in any point of the second
- The cardinality of neighborhood obtained with the nested VND is product of cardinalities
neighborhoods included, i.e.,

$$|\mathcal{N}_{\texttt{Nest-VND}}(x)| = \prod_{\ell=1}^{\ell max} |\mathcal{N}_\ell(x)|, \ x \in X.$$

- The pure `Nest-VND` neighborhood is much larger than the sequential one.
- The number of local minima w.r.t. `Nest-VND` will be much smaller than the number of loc
minima w.r.t. `Seq-VND`.

Nested VND

Function `Nest-VND` (x, x', k)
Make an order of all $\ell_{max} \geq 2$ neighborhoods that will be used in the search
Find an initial solution x; let $x_{opt} = x$, $f_{opt} = f(x)$
Set $\ell = \ell_{max}$
repeat

- **if** all solutions from ℓ neighborhood are visited **then** $\ell = \ell + 1$
- **if** there is any non visited solution $x_\ell \in N_\ell(x)$ and $\ell \geq 2$ **then** $x_{cur} = x_\ell$, $\ell = \ell - 1$
- **if** $\ell = 1$ **then**
  - Find objective function value $f = f(x_{cur})$
  - **if** $f < f_{opt}$ **then** $x_{opt} = x_{cur}$, $f_{opt} = f_{cur}$

until $\ell = \ell_{max} + 1$ (i.e., until there is no more points in the last neighborhood)

Mixed nested VND

- After exploring b (a parameter) neighborhoods, we switch from a nested to a sequent strategy. We can interrupt nesting at some level b ($1 \leq b \leq \ell_{max}$) and continue with t list of the remaining neighborhoods in sequential manner.
- If $b = 1$, we get `Seq-VND`. If $b = \ell_{max}$ we get `Nest-VND`.
- Since nested VND intensifies the search in a deterministic way, boost parameter b may seen as a balance between intensification and diversification in deterministic local search wi several neighborhoods.
- Its cardinality is clearly

$$|\mathcal{N}_{\texttt{Mix-VND}}(x)| = \sum_{\ell=b}^{\ell_{max}} |\mathcal{N}_\ell(x)| + \prod_{\ell=1}^{b-1} |\mathcal{N}_\ell(x)|, \;\; x \in X.$$

Basic VNS

The Basic VNS (BVNS) method [?] combines deterministic and stochastic changes of neighbourhood. Its steps are given in Algorithm 8.

Function VNS (x, k_{max}, t_{max})
repeat

    $k \leftarrow 1$
    repeat

        $x' \leftarrow$ `Shake(`x, k`)` `/* Shaking */`
        $x'' \leftarrow$ `FirstImprovement(`x'`)` `/* Local search */`
        `NeighbourhoodChange(`x, x'', k`)` `/* Change neighbourhood */`
    until $k = k_{max}$;

    $t \leftarrow$ `CpuTime()`
until $t > t_{max}$;

General VNS

```
Function GVNS (x, k'_max, k_max, t_max)
repeat
    k ← 1
    repeat
        x' ← Shake(x, k)
        x'' ← VND(x', k'_max)
        NeighborhoodChange(x, x'', k)
    until k = k_max;
    t ← CpuTime()
until t > t_max;
```

Skewed VNS

Function `NeighbourhoodChangeS`(x, x'', k, α)
if $f(x'') - \alpha\rho(x, x'') < f(x)$ **then**

$x \leftarrow x''$; $k \leftarrow 1$
else

$k \leftarrow k + 1$
end

Function `SVNS` $(x, k_{max}, t_{max}, \alpha)$

repeat

$k \leftarrow 1$; $x_{best} \leftarrow x$

repeat

$x' \leftarrow$ `Shake`(x, k)

$x'' \leftarrow$ `FirstImprovement`(x')

`KeepBest` (x_{best}, x)

`NeighbourhoodChangeS`(x, x'', k, α)

until $k = k_{max}$;

$x \leftarrow x_{best}$

$t \leftarrow$ `CpuTime`()

until $t > t_{max}$;

Extensions

```
Function BI-VNS (x, k_max, t_max)
repeat
    k ← 1  x_best ← x
    repeat
        x' ← Shake(x, k)
        x'' ← FirstImprovement(x')
        KeepBest(x_best, x'')
        k ← k + 1
    until k = k_max;
    x ← x_best
    t ← CpuTime()
until t > t_max;
```

Extensions

Function `FH-VNS` (x, k_{max}, t_{max})
repeat

    $k \leftarrow 1$
    repeat

        for $\ell = 1$ to k **do**

            $x' \leftarrow$ `Shake`(x, k)
            $x'' \leftarrow$ `FirstImprovement`(x')
            `KeepBest`(x, x'')
        end

        `NeighbourhoodChange`(x, x'', k)
    until $k = k_{max}$;

    $t \leftarrow$ `CpuTime()`
until $t > t_{max}$;