Variable Neighborhood Descent and Dynamic Local Search
Heuristic algorithms

Giovanni Righini

University of Milan
Department of Computer Science (Crema)
Carrying on local search without worsening the solution

Instead of repeating the local search, one can carry on the local search, after a local optimum has been reached.

Either we accept worsening solutions (hill climbing) or we change the definition of the neighborhood.

\[ x' := \arg \min_{x \in N(x)} z(x) \]

Two strategies allow for carry on the local search without worsening the solutions:

- **Variable Neighborhood Descent** changes the neighborhood \( N \);
- **Dynamic Local Search** changes the objective function \( z \).

_The former strategy guarantees loop-free search, the latter does not._
Variable Neighborhood Descent (VND)

Variable Neighborhood Descent exploit the observation that local optima are relative to the chosen neighborhood: therefore, changing the neighborhood, in general a local optimum is no longer such.

Define a sequence of neighborhoods $N_1, \ldots, N_{k_{\text{max}}}$

1. set $k := 1$
2. run a steepest descent exchange heuristic until a local optimum with respect to $N_k$ is found;
3. update $k$;
4. if the end test is not satisfied, go to step 2.

\begin{algorithm}
\textbf{VariableNeighborhoodDescent}(I, x^{(0)})
\begin{align*}
x & := \text{SteepestDescent}(x^{(0)}); x^* := x; \\
k & := 1; \\
\text{While} \ \text{Stop()} = \text{false} \ \text{do} \\
\quad x' & := \text{SteepestDescent}(x); \\
\quad x^* & := x'; \\
\quad k & := \text{Update}(k); \\
\text{EndWhile}; \\
\text{Return} \ (x^*, z(x^*));
\end{align*}
\end{algorithm}
**VND and VNS**

There is a tight relationship between *VND* and *VNS*.

The main differences: in *VND*

- at each step the current solution is also the best incumbent solution;
- neighborhoods are explored, not used for extracting random initial solutions;  
  hence, they do not grow a lot.
- neighborhoods are not necessarily nested in a hierarchy  
  hence, the update of \( k \) can be different.
- The algorithm terminates when reaching a solution that is a local optimum with respect to all neighborhoods.
**VND strategies**

There are two main types of VND algorithms:

- if neighborhoods are nested \((N_1 \subset \ldots \subset N_{k_{\text{max}}})\) we want
  - to deeply exploit the small neighborhoods;
  - to resort to the large ones only to escape from the local optima.

Hence the update of \(k\) works as in VNS:

  - when no improvements are found in \(N_k\), \(k\) is increased;
  - when improvements are found in \(N_k\), \(k\) is set back to 1;

- if neighborhoods are heterogeneous we want
  - to exploit the complementarity of different neighborhoods (e.g., exchanging vertices instead of edges...)

Hence, \(k\) scans all values from 1 to \(k_{\text{max}}\) (possibly permuting the sequence at each repetition).
Example: the CMSTP

Consider an instance of the CMSTP with \( n = 9 \) vertices, uniform weights (\( w_v = 1 \)), capacity \( W = 5 \) and the costs indicated in the figure (\( c_e \gg 3 \) for missing edges).

![Graph](image)

Considering the first solution:
- the subtree on the left cannot receive vertices: edges in the subtree on the right cannot be deleted
- if an edge of the subtree on the left is deleted, the overall cost increases: the solution is a local optimum in \( N_{S_1} \).

The neighborhood \( N_{V_1} \) (transfer of a vertex) allows for an improving move, bringing vertex 5 from the subtree on the left to the subtree on the right.
Dynamic Local Search (DLS)

Dynamic Local Search is complementary to VND:

- it keeps the initial neighborhood
- it changes the objective function

It is often used when the objective function is not so useful (wide plateaux).

The main idea is

- to associate a penalty function \( w : E \rightarrow \mathbb{N} \) with the ground set;
- to define an auxiliary function \( \tilde{z}(z(x), w(x)) \) combining the objective function \( z \) with the penalty function \( w \);
- to run a steepest descent exchange heuristic that optimizes \( \tilde{z} \);
- to update the penalty depending on the results and to restart the exchange heuristic.
Dynamic Local Search

Algorithm DynamicLocalSearch($I, x^{(0)}$)

$w := \text{InitialPenalties}(I);$  
$x^* := x^{(0)};$

While Stop() = false do

$(x, x_z) := \text{SteepestDescent}(x, z, w);$  
If $z(x_z) < z(x^*)$ then $x^* := x_z;$  
$w := \text{UpdatePenalties}(w, x, x^*);$  
EndWhile;

Return $(x^*, z(x^*))$;

The steepest descent heuristic  
• considers $w$, not only $z$;  
• provides two solutions:  
  1. a final solution $x$, locally optimal with respect to $\tilde{z}$;  
  2. the best incumbent solution $x_z$ with respect to $z$.  


There are many possible variations:

- different types of penalties
  - additive penalties:
    \[ \tilde{z}(x) = z(x) + \sum_{i \in x} w_i \]
  - multiplicative penalties (with additive objective \( z(x) = \sum_{i \in x} \phi_i \)):
    \[ \tilde{z}(x) = \sum_{i \in x} w_i \phi_i \]

- different penalty update policies
  - random update: data are perturbed by adding “noise”;
  - memory-based update, to favor intensification or diversification;

- the update can be done
  - at each exploration of the neighborhood;
  - when a local optimum is reached;
  - after a sequence of runs of the local search algorithm.
Example: *DLS* for the *Max Clique Problem*

Given a graph, find a maximum cardinality clique.

- The exchange heuristic is a *VND* with the following neighborhoods:
  1. $N_{A_1}$ (add a vertex): the solution always improves, but the neighborhood is very small and often empty;
  2. $N_{S_1}$ (exchange of two vertices):
     the neighborhood is larger but forms a *plateau* (the objective value is the same for all moves).

- The objective does not provide useful information in either case
- Associate a penalty $w_i$ (initially null) with each vertex $i$;
- The exchange heuristic minimizes the overall penalty (in the neighborhood);
- The penalty is updated
  1. when the exploration of $N_{S_1}$ is over: the penalty of the vertices currently in the clique is increased by 1
  2. after a given number of iterations: all non-zero penalties are decreased by 1.

The aim of the method is to try to expel the vertices from the clique, especially those that have been in the clique for longer.