Adaptive Large Neighborhood Search

Heuristic algorithms

Giovanni Righini

University of Milan

Department of Computer Science (Crema)

Università degli Studi
di Milano

VLSN and LNS

By Very Large Scale Neighborhood (VLSN) local search, we indicate a local search algorithm that explores a neighborhood of exponential size for each move.

Three typical ways:

- Variable depth (Lin-Kernighan, ejection chains, . . .)
- Network flow sub-problems (min cost cycle, shortest path, min cost assignment)
- Polynomially solvable sub-problems (problems with special structure)

Large Neighborhood Search (LNS) does the same, but in non of these three ways. Instead

- the search is defined by destroy and repair procedures;
- the neighborhood is often sampled rather then explored.

Notation

For a given discrete optimization problem:

- $\mathcal{I}_n$ is the (possibly infinite) set of instances of size n
- $I \in \mathcal{I}_n$ is an instance
- $X(I)$ is the feasible set
- c is the cost function, to be minimized
- $N(x)$ is the neighborhood of a solution $x \in X(I)$.

The size of $N(x)$ is $size(n) = \max\{|N(x)| : x \in X(I), I \in \mathcal{I}_n\}$.

For instance, the 2-opt neighborhood for the TSP has $size(n) \in O(n^2)$.

Destroy and repair

The destroy procedure selects some variables of the problem and deletes the values they have in x, i.e. it makes the variables free, leaving the others frozen to their current values.
Output: an infeasible (incomplete) solution $d(x)$.

The repair procedure optimizes the values of the free variables, leaving the others unchanged.
Output: a new feasible (complete) solution $r(d(x))$. The destroy

procedure is often non-deterministic: the exponential-size neighborhood is sampled.

The repair procedure can be either exact or heuristic.

Pseudo-code

procedure LNS($x^{(0)}$)
    $x^b \leftarrow x^{(0)}$
    repeat
        $x^t \leftarrow r(d(x))$ ▷ Tentative neighbor
        if *Accept*(x^t) **then**
            $x \leftarrow x^t$ ▷ Update current
        end if
        if $c(x^t) < c(x^b)$ **then**
            $x^b \leftarrow x^t$ ▷ Update best incumbent
        end if
    until *Stopcriterion* **return** x^b
end procedure

Destroy procedure

The destroy procedure is mainly defined by a degree of destruction, representing the fraction of variables affected by the procedure.
It affects diversification.

- It can be fixed
- It can be varied gradually at run-time
- It can be chosen at random from a range depending on the size of the instance, *n*

It makes sense to destroy sets of variables that are likely to highly interact with one another (for instance, arcs in the same portion of a Hamiltonian tour).

Property: It must be possible to destroy any part of the current solution.

Repair procedure

Depending on the size of the repair sub-problem, the repair procedure can be exact or heuristic.
It affects intensification.

If the repair sub-problem is not too large, a MIP solver can be used as a repair sub-routine.

Acceptance test

The acceptance test can be designed in different ways:

- Improve-only: x^t replaces x only when $c(x^t) < c(x^)$;
- Simulated annealing-like: if the move is an improvement it is always accepted; otherwise, it is accepted with a probability that depends on the difference between $c(x^t)$ and $c(x^)$ and a "temperature" parameter.

Adaptive LNS

In Adaptive LNS multiple destroy and repair methods are used, each one with an associated weight.

The weights are adjusted at run-time according to the observed effects.

Notation:

- Ω_-: set of destroy methods;
- Ω_+: set of repair methods;
- ρ_-: vector of weights of destroy methods;
- ρ_+: vector of weights of repair methods;

Pseudo-code

procedure LNS($x^{(0)}$,Ω^-,Ω^+)
  $x^b \leftarrow x^{(0)}$
  $\rho^- \leftarrow 1$
  $\rho^+ \leftarrow 1$
  repeat
    $(d(), r()) \leftarrow Select(\Omega^-, \Omega^+, \rho^-, \rho^+)$ ▷ Roulette wheel
    $x^t \leftarrow r(d(x))$ ▷ Tentative neighbor
    if $Accept(x^t)$ **then**
      $x \leftarrow x^t$ ▷ Update current
    end if
    if $c(x^t) < c(x^b)$ **then**
      $x^b \leftarrow x^t$ ▷ Update best incumbent
    end if
    $\Psi \leftarrow ComputeScore(c(x^t))$
    $\rho^- \leftarrow \lambda\rho^- + (1-\lambda)\Psi$ ▷ Update weights
    $\rho^+ \leftarrow \lambda\rho^+ + (1-\lambda)\Psi$ ▷ Update weights
  until *Stopcriterion* **return** x^b
end procedure

Relevant functions and parameters

The selection of the methods is done through the “roulette wheel”, by assigning each method a probability proportional to its weight:

$$Prob_j = \frac{\rho_j^-}{\sum_{i \in \Omega^-} \rho_i^-} \quad Prob_j = \frac{\rho_j^+}{\sum_{i \in \Omega^+} \rho_i^+}$$

The computation of the score for the selected methods can be done as follows:

- $\Psi \leftarrow \omega_1$ if $c(x^t) < c(x^b)$ (new global best)
- $\Psi \leftarrow \omega_2$ if $c(x^t) < c(x)$ (improving move)
- $\Psi \leftarrow \omega_3$ if $Accept(x^t) = true$ (accepted move)
- $\Psi \leftarrow \omega_4$ if $Accept(x^t) = false$ (rejected move)

with $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4$. Computing time can also be taken into account, not to bias the weights in favor of more precise but time consuming methods.

The weights update depends on parameter λ (reactivity of self-adaptation).

ALNS design

In ALNS multiple destroy/repair methods are allowed.

- LNS: select a destroy and a repair method that are expected to work well for a wide range of instances.
- ALNS: include destroy/repair methods that only are suitable in some cases.

Methods that are good for intensification and for diversification can be selected separately.

Destroy methods

Random destroy. Randomly select the parts of the solution that should be destroyed. This is usually done for diversification purposes.

Worst destroy or **Critical destroy.** Remove a certain number of variable assignments which are likely to be sub-optimal (e.g. variables with a high cost, crossing arcs in an Euclidean TP...).

Related destroy. Select some variables that are easy to interchange while keeping the solution feasible (e.g. pairs of customers close to each other in routing problems, items with the same weight in capacitated problems). This is done to make the repair operation easier and more likely to succeed.

History based destroy. Variables to be destroyed are selected on the basis of some information coming from previous iterations. (e.g. count how often setting a given variable to a specific value has led to a bad solution).

Repair methods

The repair methods are usually well-performing heuristics for the specific problem.

Greedy repair. Variants of the greedy paradigm:

- perform the best choice at each step
- perform the least bad choice at each step.

Exact repair. Exact algorithms can be used or possibly relaxed to obtain faster solution times (giving up the optimality guarantee).

Time consuming and fast repair methods can be mixed by suitably penalizing the former ones.

In problems whose solutions are made by sub-problems (VRP, BPP, PMSP,...):

- sub-problems solved sequentially (sequential heuristics);
- all subproblems solved at the same time (parallel heuristics).

More on diversification and intensification

In traditional LS, diversification is controlled by some specific parameters, such as the accept ratio, the tabu tenure, etc.

In ALNS noise and randomization are used in both destroy and repair methods for diversification.

A classic local search heuristic can be run from time to time, to better explore the solution space “close to” the current solution (intensification).

Coupled neighborhoods

A natural extension to the ALNS framework is to have **coupled destroy/repair**.

For each destroy method d, define a subset R_d of repair methods that can be coupled with it.

The roulette wheel selection of repair neighborhoods restricts the selection to R_d if d is used for destroy.

Special case: one may have $R_d = \emptyset$, meaning that d does both the destroy and repair steps.

Properties of ALNS

Robustness. ALNS uses several complementary (often operational) neighborhood definitions: good exploration of the search space.

Self-tuning. Self-adaptation of the weights implies limited tuning effort to calibrate the algorithm.

Inclusiveness. THere is no need to select neighborhoods: one can put all of them into the ALNS algorithm.

Similarities and differences

VND searches complementary neighborhoods according to a predefined sequence (usually according to their complexity).

VNS searches nested neighborhoods of increasing size and complexity.

ALNS searches complementary neighborhoods selecting them according to a probability, which is adapted during the search.

It can be seen as a special case of a *hyper-heuristic* (i.e. a heuristic to select heuristics).