Developing Variable neighborhood search

N. Mladenović

LAMIH, University of Valenciennes and MI SANU Belgrade

Nizni Novgorod, November 2014

A step-by-step procedure

This seminar is devoted to newcomers. Its purpose is to help students in making a first very simple version of VNS, which would not necessary be competitive with later more sophisticated versions. Most of the steps are common for implementation of other metaheuristics.

- **Getting familiar with the problem.**
  - Think about problem at hand;
  - in order to understand it better, make a simple numerical example and spend some time in trying to solve it by hand in your own way.
  - Try to understand why problem is hard and why heuristic is needed.

A step-by-step procedure

- **Read literature.** Read about problem and solution methods in the literature.
- **Test instances (read data).**
  - Use your numerical example as a first instance for testing your future code, but
  - if it is not large enough, take some from the web, or make routine for generating random instances.
  - In the second case, read how to generate events using uniformly distributed numbers from (0,1) interval (since each programming language has statement for getting such random numbers).

A step-by-step procedure

- **Data structure.** Think about how the solution of the problem will be represented in the memory.
- Consider two or more presentations of the same solution if they can reduce the complexity of some routines, i.e., analyze advantages and disadvantages of each possible presentation.
- **Initial solution.** Having a routine for reading or generating input data of the problem, the next step is to get initial solution.
- For the simple version, any random feasible solution may be used, but usual way is to develop some *greedy* constructive heuristic, which suppose to be not very hard to do.

A step-by-step procedure

- **Objective value.**
  - Make a procedure that calculates objective function value for a given solution.
  - Notice that at this stage, we already have all ingredients for Monte-Carlo method: generation of random solution and calculation of objective function value.
  - Get solution of your problem by Monte-Carlo heuristic (i.e., repeat steps 5 and 6 many times and keep the best one).

A step-by-step procedure

- **Shaking.**
  - Make a procedure for Shaking. This is a key step of VNS.
  - However, it is easy to implement and usually it has only a few lines of computer code.
  - For example, in solving clustering problems easiest perturbation of the current solution is to re-allocate randomly chosen entity ℓ from the cluster it belongs to another one, also chosen at random.
  - In fact, in this case, shaking step (or jump in the k^{th} neighborhood) could have only three lines of the computer code:

A step-by-step procedure

- **Shaking.**
  - For $i = 1$ to k
  - $a(1 + n \cdot Rnd1) = 1 + m \cdot Rnd2$
  - EndFor
  - Solution is saved in array $a(\ell) \in \{1, \ldots, m\}$ that denotes membership or allocation of entity ℓ ($\ell = 1, \ldots, n$);
  - $Rnd1$ and $Rnd2$ denote random numbers uniformly distributed from the (0,1) interval. Compare results of obtained Reduced VNS (take $k_{max} = 2$) with Monte-Carlo method.

A step-by-step procedure

- **Local search.**
  - Choose an off-the-shelf local search heuristic (or develop new one).
  - In building a new local search, consider several usual moves that define neighborhood of the solution *drop, add, swap, interchange*, etc.
  - Also, for the efficiency (speed) of the method, it is very important to pay special attention to *updating* of the incumbent solution. In other words, usually it is not necessary to use procedure for calculating objective function values for each point in the neighborhood, i.e., it is possible to get those values by very simple calculation.

A step-by-step procedure

- **Comparison.**
  - Include Local search routine into RVNS to get the basic VNS,
  - Compare it with other methods from the literature.

More tips

Sometimes basic VNS does not provide very good results.

1. **First vs. best improvement.** Compare experimentally *first* and *best improvement* strategies within local search. Previous experience suggest the following: if your initial solution is chosen at random, use first improvement, but if some constructive heuristic is used, use best improvement rule.
2. **Reduce the neighborhood.** The reason of bad behavior of any local search may be unnecessary visit to all solutions in the neighborhood. Try to identify "promising" subset of the neighborhood and visit only them; ideally, find a rule that automatically selects solutions from the neighborhood whose objective values are not better than the current one.

More tips

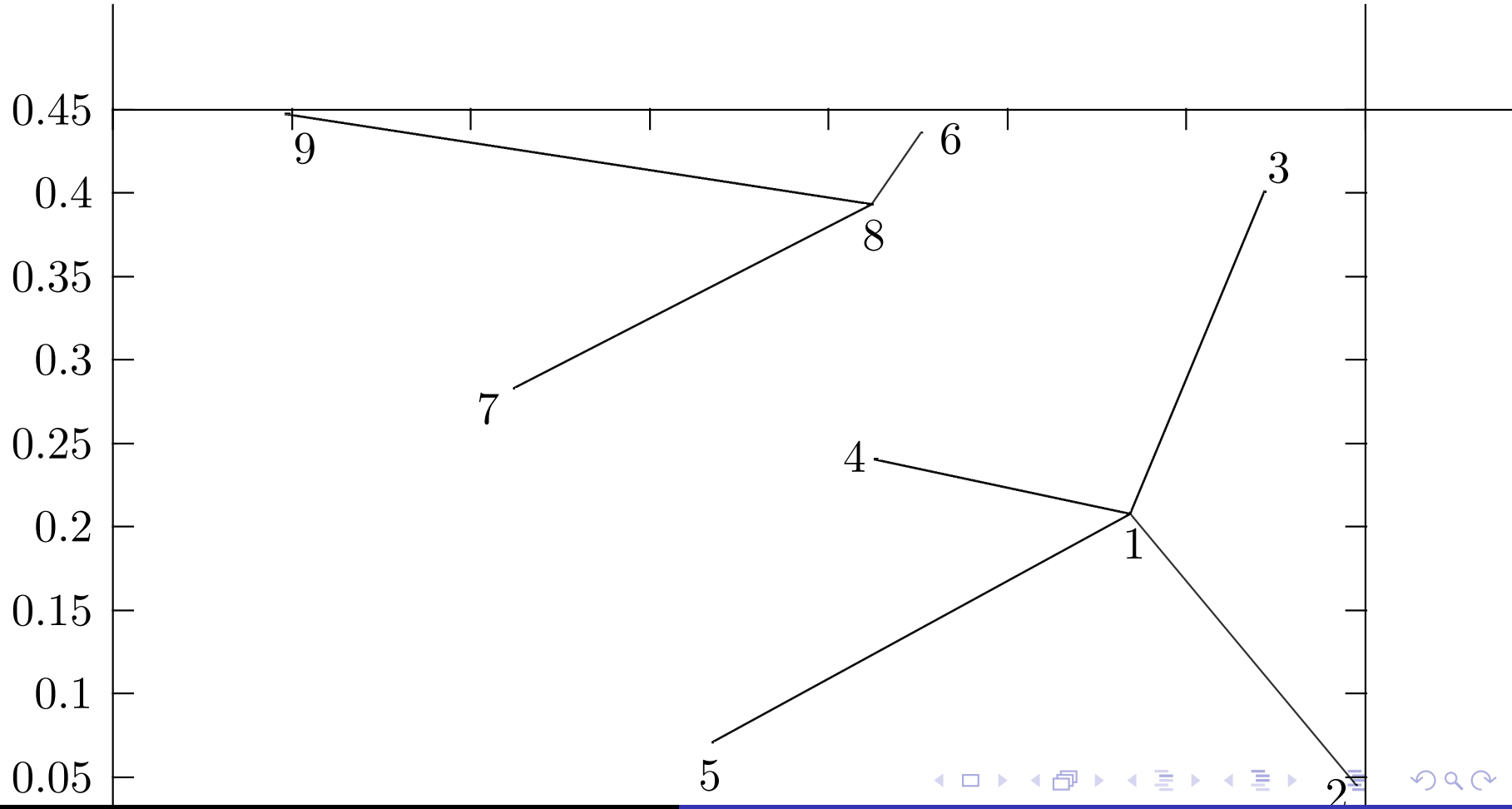
1. **Intensified shaking.** In developing more effective VNS, one must spend some time in checking how sensitive is the objective function on small change (shake) of the solution. The trade-off between intensification and diversification of the search in VNS is balanced in Shaking procedure. For some problem instances completely random jump in the k^{th} neighborhood is too diversified. In such cases, some *intensify shaking* procedure may increase intensification of the search. For example, k-interchange neighborhood may be reduced by repeating k times *random add* followed by *best drop* moves. (Special case of intensified shaking is so-called *Large neighborhood search*, where k randomly chosen attributes of the solutions are destroyed (dropped), and then the solution is re-build in the best way (by some constructive heuristic).)

More tips

1. **VND.** Analyze several possible neighborhood structures, estimate their size, make order of them, i.e., develop VND and replace Local search routine with VND to get general VNS.
2. **Experiments with parameter settings.** The single parameter of VNS is k_{max}, which should be estimated experimentally. However, usually the procedure is not very sensitive on k_{max} and, in order to make parameter free VNS, one can fix its value at the value of some input parameter, e.g., for the p-median (Example 3), $k_{max} = p$; for the minimum sum-of-square clustering (Example 2) $k_{max} = m$, etc.

Example

$n = 9$ points are chosen in the rectangle $[0, 1] \times [0, 0.5]$. Coordinates:
$a_1 = (0.8689, 0.2076)$, $a_2 = (0.9951, 0.0448)$, $a_3 = (0.9447, 0.4013)$,
$a_4 = (0.7257, 0.2407)$, $a_5 = (0.6351, 0.0708)$, $a_6 = (0.7527, 0.4366)$,
$a_7 = (0.5242, 0.2826)$, $a_8 = (0.7244, 0.3932)$, $a_9 = (0.3964, 0.4476)$.



The corresponding (symmetric) distance matrix (multiplied by 10,000) is then:

	1	2	3	4	5	6	7	8	9
1	0	2059	2080	1469	2709	2567	3527	2352	5299
2	2059	0	3600	3330	3609	4606	5274	4411	7215
3	2080	3600	0	2715	4529	1951	4369	2203	5501
4	1469	3330	2715	0	1925	1977	2057	1525	3889
5	2709	3609	4529	1925	0	3842	2390	3346	4460
6	2567	4606	1951	1977	3842	0	2755	517	3564
7	3527	5274	4369	2057	2390	2755	0	2287	2087
8	2352	4411	2203	1525	3346	517	2287	0	3324
9	5299	7215	5501	3889	4460	3564	2087	3324	0

Table: Distance matrix (d_{ij}) with $n = 9$

.