Les Cahiers du GERAD ISSN: 0711–2440

Sequential variable neighborhood descent variants: An empirical study on Travelling salesman problem

A. Mjirda, R. Todosijević, S. Hanafi, P. Hansen, N. Mladenović

G–2015–37

April 2015

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Sequential variable neighborhood descent variants: An empirical study on Travelling salesman problem

Anis Mjirda [a]
Raca Todosijević [a]
Saïd Hanafi [a]
Pierre Hansen [b]
Nenad Mladenović [a,b]

[a] *LAMIH – Université de Valenciennes et du Hainaut-Cambrésis, 59313 Valenciennes Cedex 9, France*

[b] *GERAD, HEC Montréal, Montréal (Québec) Canada, H3T 2A7*

anis.mjirda@univ-valenciennes.fr
racatodosijevic@gmail.com
said.hanafi@univ.valenciennes.fr
pierre.hansen@gerad.ca
nenad.mladenovic@gerad.ca

April 2015

Les Cahiers du GERAD
G–2015–37

Abstract: Usually several neighborhood structures may be explored within a single local search algorithm. The simplest way is to define a single neighborhood as a union of all predefined neighborhood structures. The another possibility is to make an order (or sequence) of those neighborhoods and use them in the first improvement or the best improvement fashion, following that order. In this work, we firstly classify possible variants of sequential use of neighborhoods and then empirically analyze them in solving the classical travelling salesman problem. Most commonly used TSP neighborhood structures of the same size, such as 2-opt and insertion neighborhoods are explored in our empirical study. We in fact tested 76 different such heuristics on 15,200 random test instances. Several interesting observations are derived. In addition, the two best out of 76 heuristics (used as a local searches within variable neighborhood search (VNS)) are tested on 23 TSPLIB test instances. It appears that union of neighborhoods does not perform well.

Key Words: Neighborhood structures, variable neighborhood descent, variable neighborhood search, TSP, empirical study.

1 Introduction

General form of an optimization problem is as follows:

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

where $\mathcal{S}, X, x$ and f respectively denote the solution space and feasible set, a feasible solution and a real-valued objective function.

Variable neighborhood search (VNS)[7] is a metaheuristic, or a framework for building heuristics, able to solve problem (1). VNS systematically changes neighborhood structures during the search for a (near)optimal solution of a considered problem. The change of neighborhood structures occurs in both a descent phase, which generates a local optimum, and a shaking phase which aim is to possibly resolve local optima traps. These two phases together with a neighborhood change step represent main ingredients of a VNS based heuristic. In each VNS based heuristic they are iterated until the predefined stopping condition is reached. Following this basic scheme of a VNS, many VNS variants have been proposed until now (see e.g. [2, 4] for recent surveys on VNS).

The widely used VNS variants are Basic VNS and General VNS (GVNS) which use a simple local search and Variable Neighborhood Descent (VND), respectively. The idea of VND stems from the facts that a local optimum relatively to one neighborhood structure is not necessarily a local optimum for another neighborhood structure; and a global optimum is a local optimum with respect to all neighborhood structures. Thus, VND explores a solution space using several neighborhood structures either in a sequential, a nested (or composite) or mixed nested way [3, 9].

In this paper we are focused on sequential VND variants. The main purpose of this paper is to experimentally compare several sequential VND strategies that exist in the literature and to see are there some significant differences among them. Our empirical study considers VND variants that differ in the following aspects:

1. **Order of neighborhoods** that may play important role in the quality of the final solution;
2. **First or best improvement search strategy.** Once neighborhood structures are placed in the list in order they will be used, there are still a few possibilities how to perform a search within them. The question common to any local search heuristic is whether to use:
   (i) **the first improvement** strategy - as soon as an improving solution is detected it is set to be the new incumbent solution or
   (ii) **the best improvement** strategy - the best among all improving solutions, if any, is set to be the new incumbent solution.
3. **Move or not** decision (i.e., if an improvement has been detected in some neighborhood, how the search after updating the incumbent solution is continued):
   (i) **Basic VND (B-VND)** - returns to the first neighborhood from the list;
   (ii) **Pipe VND (P-VND)** - continues the search in the same neighborhood;
   (iii) **Cyclic VND (C-VND)** - resumes the search in the next neighborhood from the list;
   (iv) **Union VND (U-VND)** (or Multiple neighborhood search, where the single neighborhood is obtained as the union of all predefined neighborhoods) - continues the search in the same large neighborhood. U-VND is recently proposed in [6, 10] and used within Tabu search.
4. **Initial solution** could be obtained at random or by applying some constructive heuristic.

It is clear that more than dozen different VND variants may be obtained by choosing different options in previous four possibilities: order of neighborhoods and/or by changing the search strategy, etc. Although most of them have been already used, precise classification of sequential VND search strategies has not been made before.

Therefore, the main goal of this paper is the classification of sequential VND algorithms and extensive empirical evaluation of their similarities and differences.

In order to shed more light on these issues, we conduct an empirical study on classical Travelling salesman problem (TSP). The empirical study is performed implementing each VND variant (Basic VND, Pipe VND, Cyclic VND, and Union VND) using classical neighborhood structures of the same size, i.e., 2-Opt, Insertion-1 and Insertion-2 [5]. Since it is clear from other studies that neighborhoods should be ordered in the increasing cardinality order, in this study we focus on well-known neighborhoods of the same size $O(n^2)$. These three neighborhood structures are tested in all possible orders. Moreover, VND variants include examination of the search strategies (first or best) and also what initial solution (greedy or random) is most suitable for the certain order of neighborhoods. After detecting the best VND approach among those that re-center search in the inner loop (i.e., B-VND, P-VND and C-VND), we compare performance of a GVNS heuristic that uses it in a descent phase with performance of a GVNS heuristic that uses Union VND in a descent phase.

In our view the need for the empirical study of this kind is evident. Questions we are trying to answer in this paper are usually treated as technical and not important for the behaviour of heuristics. On the contrary, we believe that some common patterns regarding performances of VND variants do exist and could be explored in designing an effective and an efficient VNS based heuristic in the future work. Moreover, if some patterns are evident, they could be even treated and investigated theoretically.

The rest of the paper is organized as follows. In the next section we present main ideas of a VND heuristic and describe existing VND variants which examines neighborhood structures one after another according to the predefined rules. In Section 3, we perform comparison of VND variants presented in Section 2 on randomly generated test instances for TSP. After that, in the same section we compare performance of GVNS heuristic that uses the best VND variant with re-centering (according to the results obtained in the first part of experiments) with performance of VNS heuristic that uses Union VND. Finally, in Section 4, we draw some conclusions and present possible future research lines.

2 Variants of variable neighborhood descent

In this section we provide the detailed description of the Basic VND, the Pipe VND, the Cyclic VND, and the Union VND procedures. All these procedures follow the steps given in Algorithm 1. Each of these procedures starts from a given solution x. In each VND iteration a local search procedure through a given neighborhood structure is applied, followed by a neighborhood change function (Step `Change Neighborhood` (x, x', k)). The neighborhood change function defines the neighborhood structure that will be examined next. Each VND variant stops when there is no improvement with respect to any of the considered neighborhood structures. Thus, the solution obtained by a VND is local optimum with respect to all neighborhood structures.

However, the difference among these variants comes from the way of changing neighborhood structures through the function *Change Neighborhood* in Algorithm 1. Thus, for each variant,its neighborhood change function will be presented.

The first variant is so-called Basic VND which performs search for an improving solution examining neighborhoods following their order in the list $\{1, \ldots, k_{max}\}$. As soon as an improving solution is detected in some neighborhood structure, exploration is continued in the first neighborhood structure in the list. The steps of *Change Neighborhood* function for the Basic VND procedure are given in Algorithm 2.

The second variant is named Pipe VND and it uses the neighborhood change function (see Algorithm 3) based on the following rule: If there is an improvement with respect to some neighborhood $\mathcal{N}_k$, exploration is continued in that neighborhood, otherwise it is continued in the next neighborhood from the list.

The third one is entitled Cyclic VND. It continues the search in the next neighborhood structure in the list regardless of the fact if an improvement with respect to the current neighborhood structure is reached or not. The neighborhood change function that follows that principle is presented in Algorithm 4.

The U-VND explores several neighborhood structures at each iteration. The best solution encountered in the union of considered neighborhood structures is examined to replace the current incumbent solution.

Algorithm 1: Variable Neighborhood Descent

```
   Function VND (x)
1  Define neighborhood structures N_k (k = 1, ..., k_max);
2  x'' ← x;
3  Stop= False;
   while Stop= False do
4  |  x ← x'';
5  |  Stop= True;
6  |  k ← 1;
   |  while k ≤ k_max do
7  |  |  x' ← argmin_{y∈N(x)} f(y);
8  |  |  Change Neighborhood (x, x', k);
9  |  |  if x' better than x'' then
10 |  |  |  x'' ← x';
11 |  |  |  Stop= False;
   |  |  end
   |  end
   end
12 return x'';
```

Algorithm 2: Change neighborhood for B-VND

```
   Function Change Neighborhood (x, x', k)
   if x' better than x then
1  |  x ← x'; k ← 1;
   else
2  |  k ← k + 1;
   end
```

Algorithm 3: Change neighborhood for P-VND

```
   Function Change Neighborhood (x, x', k)
   if x' better than x then
1  |  x ← x';
   else
2  |  k ← k + 1;
   end
```

Algorithm 4: Change neighborhood for C-VND

```
   Function Change Neighborhood (x, x', k)
1  k ← k + 1;
   if x' better than x then
2  |  x ← x';
   end
```

This VND variant is also known in the literature as multiple neighborhood search [6, 10]. The neighborhood change procedure used within U-VND is given in Algorithm 5. It is obvious that U-VND in each iteration explores neighborhoods defined relative to the the same solution x, while that is not necessary true for the other VND variants. Namely, the other VNDs may re-center search in the inner loop moving to a new solution which is better than the current incumbent solution.

Algorithm 5: Change neighborhood for U-VND

Function `Change Neighborhood` (x, x', k)

1 $k \leftarrow k + 1$;

3 Empirical study

As it is usual, in order to asses performance of each VND variant under different settings, we perform testing on the well-known Travelling Salesman Problem. TSP is a well-studied problem in the literature and usually used to present new solution methods. It consists of finding an optimal Hamiltonian tour for a salesman visiting a set of cities. Formally, the problem is defined on a graph $G =< N, A >$ where $N = 1, 2, \ldots, n$ represent the set of nodes (cities) and A is the set of arcs (the arc $(i, j) \in A$ if $i < j$). For each arc (i, j), a cost of travelling is associated denoted by c_{ij}.

For testing VND variants, we used three classical TSP neighborhood structures [5]: 2-opt, Insertion-1 and Insertion-2 neighborhoods. The first neighborhood structure is 2-opt neighborhood which is based on a move that consists of breaking down two arcs from a tour and reconnecting obtained sub-tours to have a new tour. The 2-opt move that involves edges (i, j) and $(i + 1, j + 1)$ is given in Figure 1. The last two neighborhood structures are based on Insertion-1 and Insertion-2 moves, respectively (Figures 2 and 3). Note that Insertion-k move relocates k successive vertices from the current position between any two vertices in a tour.

The empirical study presented hereafter consists of two parts. First part of experiments is devoted to determining best settings for each VND variant and mutual comparison of VND variants. In the second part we evaluate impact of each of two best VND variants, according to the results in the first part, on overall performance of a VNS heuristic that applies it in the intensification phase.

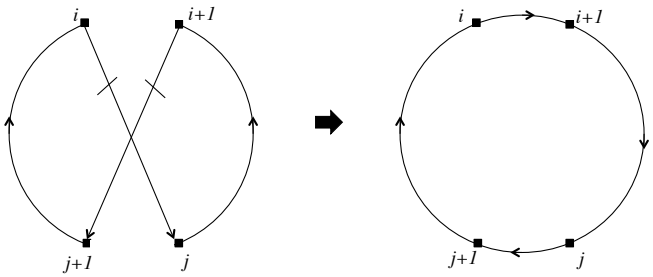

Figure 1: 2-opt move

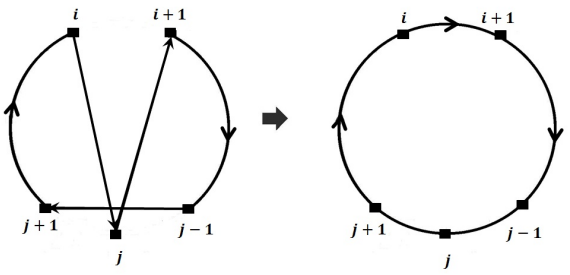

Figure 2: Insertion-1 move

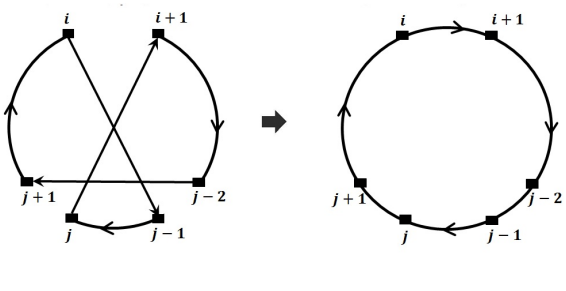

Figure 3: Insertion-2 move

3.1 Comparison of VND variants

After choosing the neighborhood structures to be exploited by a VND variant, several issues to be resolved arise. Namely, it is obvious that the performance of a VND variant is directly correlated with a provided initial solution as well as with search strategy (first or best) used to explore chosen neighborhoods. Additionally, the performances of all VND variants except U-VND depend on the order of neighborhoods in the list. Thus, in order to have more insights on performance of each VND variant as well as to detect more suitable settings for each of them, we perform exhaustive testing on a data set of TSP instances generated randomly from a uniform distribution on the [0,100] $\times$ [0,100] square varying number of nodes, i.e., n, from 20 to 1000 (see [1] for more details). More precisely, the data set is obtained by generating 1000 instances for each $n = \{20, 30, 40, \ldots, 150\}$, and by generating 100 instances for each $n = \{200, 250, 300, 350, 400, 450, 500, 600, 700, 800, 900, 1000\}$. In each test instance, the distance between any two nodes is calculated as Euclidian distance.

On each instance from this data set, each VND variant except U-VND is tested under 24 different settings. Each setting corresponds to choosing one of two common ways for getting an initial solution (i.e., *random* (solution generated as a random permutation of nodes) or *greedy*), choosing one of six possible neighborhood orders, and choosing the best or the first improvement search strategy for neighborhood exploration. On the other hand, U-VND is tested under two different settings as: U-VND that uses the best improvement search strategy and the greedy initial solution and U-VND that uses the best improvement search strategy and the random initial solution. Note that if the first improvement search strategy is used within U-VND, U-VND is equivalent to Basic VND.

The summarized results for each VND variant are presented in Tables 1–4. In the tables, we used the following abbreviations to denote used search strategy and way of generating an initial solution for considered VND variant:

- `BI-GI`: Solution provided by the greedy heuristic, while neighborhoods explored by the best improvement search strategy.
- `FI-GI`: Solution provided by the greedy heuristic, while neighborhoods explored by the first improvement search strategy.
- `BI-RI`: Solution obtained as random permutations of nodes, while the best improvement strategy used.
- `FI-RI`: Solution provided by a random permutations of nodes while using the first improvement strategy.

In order to distinguish six possible orders of the neighborhoods, we use the following notation:

- 1^{st} `order`: 2-opt, Insertion-1, Insertion-2;
- 2^{nd} `order`: 2-opt, Insertion-2, Insertion-1;
- 3^{rd} `order`: Insertion-1, 2-opt, Insertion-2;
- 4^{th} `order`: Insertion-2, 2-opt, Insertion-1;
- 5^{th} `order`: Insertion-1, Insertion-2, 2-opt;
- 6^{th} `order`: Insertion-2, Insertion-1, 2-opt.

In the tables, we report average solution values and average CPU times over all test instances in the data set, attained by VND variants under different settings. The settings of a VND variant are provided in column and row headings as described above. For example, in Table 1, the value in the cell, in the intersection of the row `BI-GI` and column 1^{st} `order` corresponds to the average solution value achieved by B-VND which starts from the greedy initial solution, and explores neighborhoods using the 1^{st} order and the best improvement search strategy. In each table, the best average solution value, found by a certain VND variant, is boldfaced as well as the average CPU time consumed to find that value.

Table 1: B-VND: Average values and CPU Time of all test instances

Orders	1^{st}		2^{nd}		3^{rd}		4^{th}		5^{th}		6^{th}		**Average**	
	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time
BI-GI	1199.17	0.11	1198.98	0.16	1205.40	1.00	1200.94	0.99	1203.91	1.27	1201.85	1.64	1201.71	0.86
FI-GI	1198.72	0.09	**1198.24**	**0.16**	1210.53	0.86	1203.94	0.99	1208.14	1.15	1205.47	1.63	1204.17	0.81
BI-RI	1220.11	1.25	1219.29	1.36	1229.75	10.38	1219.52	7.43	1225.21	12.16	1219.69	10.65	1222.26	7.21
FI-RI	1203.05	0.12	1202.57	0.19	1232.50	40.39	1220.55	8.17	1228.09	42.56	1222.53	11.78	1218.22	17.20
Average	1205.26	0.39	1204.77	0.47	1219.54	13.16	1211.24	4.39	1216.34	14.29	1212.39	6.42	1211.59	6.52

Table 2: P-VND: Average values and CPU Time of all test instances

Orders	1^{st}		2^{nd}		3^{rd}		4^{th}		5^{th}		6^{th}		**Average**	
	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time
BI-GI	1199.28	0.13	1199.11	0.14	1205.37	0.75	1201.56	0.48	1204.78	0.79	1203.34	0.82	1202.24	0.52
FI-GI	1198.92	0.11	**1198.52**	**0.12**	1209.12	0.53	1204.59	0.33	1208.68	0.56	1207.02	0.53	1204.47	0.37
BI-RI	1220.51	1.29	1219.86	1.32	1227.50	7.84	1224.34	4.91	1227.48	8.46	1225.02	6.49	1224.12	5.05
FI-RI	1203.24	0.15	1202.86	0.15	1222.63	36.78	1221.04	4.70	1224.78	33.42	1223.94	6.90	1216.42	13.68
Average	1205.49	0.42	1205.09	0.43	1216.15	11.48	1212.88	2.60	1216.43	10.81	1214.83	3.69	1211.81	4.91

Table 3: C-VND: Average values and CPU Time of all test instances

Orders	1^{st}		2^{nd}		3^{rd}		4^{th}		5^{th}		6^{th}		**Average**	
	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time
BI-GI	1199.04	0.46	1198.97	0.46	**1198.76**	**0.46**	1199.00	0.46	1198.94	0.46	1198.96	0.46	1198.94	0.46
FI-GI	1204.45	0.30	1204.34	0.30	1204.36	0.30	1204.35	0.30	1204.37	0.30	1204.51	0.30	1204.39	0.30
BI-RI	1225.47	3.22	1226.07	3.22	1226.21	3.21	1225.48	3.21	1225.74	3.21	1226.31	3.21	1225.88	3.21
FI-RI	1216.16	1.27	1215.73	1.30	1215.96	1.31	1216.70	1.26	1215.98	1.26	1215.66	1.31	1216.03	1.29
Average	1211.28	1.31	1211.28	1.32	1211.32	1.32	1211.38	1.31	1211.26	1.31	1211.36	1.32	1211.31	1.31

Table 4: Union VND: Average values & CPU Time of all test instances

	Average value	Average CPU time
BI-GI	**1197.65**	**1.06**
BI-RI	1226.41	8.69
Average	1212.03	4.87

From the results presented in Tables 1–4, we may conclude the following:

(i) Use of greedy initial together with the best improvement strategy appears to be more successful than other variants;

(ii) For B-VND, P-VND and C-VND, the first improvement search strategy is the better option if an initial solution is chosen at random. On the other hand if an initial solution is constructed in a greedy manner, the best improvement search strategy turns out to be better choice (Compare the averages for B-VND, P-VND and C-VND over 6 possible orders); Note that that same conclusion was drawn in [1].

(iii) There is no significant difference among variants that use different orders in the descent, but slightly better results have been reported when the 2-opt neighborhood structure is used as the first one. On the other hand, it turns out that use of Insertion-1 neighborhood as the first one in the list provides the worst results.

(iv) C-VND is more resistent to the change in the neighborhood order than any other VND variant.

(v) B-VND and P-VND offer the best results if they use 2^{nd} order of neighborhoods (2-opt, Insertion-2, Insertion-1) together with the first improvement search strategy to improve the greedy initial solution provided to them. Additionally, the average CPU times consumed by B-VND and P-VND upon reporting the best results are very similar (compare 0.16s of B-VND and 0.12s of P-VND)

(vi) C-VND provides the best results when it uses the greedy solution and explores neighborhoods using the 3^{rd} order (Insertion-1, 2-opt, Insertion-2) together with the best improvement search strategy.

(vii) For all VND variants the best initial solution is greedy. Such outcome was relatively expected, confirming again that good initial solution may be crucial for final performance of a heuristic.

(viii) U-VND is slightly better than the others VNDs, regarding the best average values attained, but significantly slower than the others. This may be explained by the fact that U-VND in each iteration performs exploration of large part of the solution space before, deciding to re-center search. Obviously, such principle is suitable for reaching good final solution, but requires huge CPU time investment.

(ix) Comparing VNDs that re-center search in the inner loop (i.e., B-VND, P-VND and C-VND), it follows that B-VND is able to provide the best solution values. Regarding average CPU times consumed by B-VND, P-VND and C-VND to find the best reported average solution value their ranking is as follows: the fastest is P-VND, B-VND is ranked as the second, while C-VND is the slowest one. However, since, B-VND consumed negligible more CPU time to find the best reported average solution value comparing to CPU time that P-VND consumed to do so, we may conclude that B-VND is the best VND approach among those that re-center search in the inner loop.

3.2 GVNS using Basic VND vs. GVNS using union VND

As stated in the previous section, regarding the best average solution quality, the best VND approach is U-VND, while the best VND approach among those that re-center search in the inner loop is B-VND. However, U-VND is significantly slower than B-VND. Thus, the aim of this part of experiments is to reveal whether is more beneficial to use within VNS, a VND that re-center search in the inner loop or not. For that purpose, we implement and test two GVNS heuristics named: `GVNS_B-VND`, which uses B-VND as a local search, and `GVNS_U-VND` which employs U-VND as a local search. U-VND and B-VND used within tested GVNS heuristics explores again neighborhood structures 2-opt, Insertion-1 and Insertion-2. The search strategy and order of neighborhoods used within B-VND are determined as the ones that enable B-VND to achieve the highest performance, according to results from the previous section, i.e., the first improvement search strategy and the 2^{nd} order of neighborhoods (2-opt, Insertion-2, Insertion-1). Both GVNS heuristics use the same shaking procedure to possibly resolve local optima traps. The shaking procedure, for the input requires solution x and parameter k, while at the output it returns a solution obtained executing k random 2-opt moves on the solution x. Following the observation that the best initial solution for all VND approaches is the one built by greedy procedure, we provide the same greedy solution as the initial one for both GVNS heuristics. The framework of both GVNS heuristics is given in Algorithm 6. As the input, both GVNS require parameters t_{max} and k'_{max} which represent the maximum CPU time allowed to GVNS and the maximum

Algorithm 6: Steps of GVNS

```
   Function GVNS(t_max, k'_max);
1  x ← greedy_solution();
2  repeat
3  |  k ← 1;
4  |  while k ≤ k'_max do
5  |  |  x' ← Shake(x, k) ;
6  |  |  x'' ← VND(x') ;
7  |  |  k ← k + 1;
8  |  |  if x'' is better than x then
9  |  |  |  x ← x'';
10 |  |  |  k ← 1;
   |  |  end
   |  end
11 |  t ← CpuTime();
   until t > t_max;
12 Return x;
```

value of shaking parameter k. The performances of `GVNS_B-VND` and `GVNS_U-VND` have been disclosed on 23 TSP instances from TSPLIB [8]: eil51, berlin52, kroC100, kroD100, lin105, ch130, ch150, d198, tsp225, pr299, linhp318, rd400, pr439, pcb442, u574, p654, pr1002, u1060, pcb1173, d1291, rl1323, u1432, pr2392 (i.e., the same ones used in [1]). On each test instance, each GVNS has been executed ten times on each test instance, with different random seeds. In addition, we test both GVNS using 12 different time limits ranging from 10 seconds to 300 seconds. In all testing GVNS parameter k'_{max} is set to 10. The results are summarized in Table 5 and Figure 4. For each time limit and each GVNS, we report the average value of the best solution values found on the entire set of 23 instance in ten runs (Column `Best results GVNS_B-VND value` and Column `Best results GVNS_U-VND value`), as well as the average of average solution values over ten runs on all test instances (Column `Average results GVNS_B-VND value` and Column `Average results GVNS_U-VND value`). Additionally, for each time limit and each GVNS variant, we report the average CPU time needed to find the best solution value (Column `Best results GVNS_B-VND time` and Column `Best results GVNS_U-VND time`) and average CPU time needed to solve an instance from the data set with respect to 10 runs (Column `Average results GVNS_B-VND time` and Column `Average results GVNS_U-VND time`).

Table 5: GVNS using Basic VND vs. GVNS using Union VND (U-VND)

	Best results				Average results			
	`GVNS_U-VND`		`GVNS_B-VND`		`GVNS_U-VND`		`GVNS_B-VND`	
Time limit (s)	Value	Time	Value	Time	Value	Time	Value	Time
10	84232.65	6.40	81401.09	4.81	84652.94	5.85	81827.33	4.85
20	83469.83	11.36	81158.70	8.86	83741.43	9.51	81520.45	9.15
30	83091.09	16.25	80904.30	14.21	83370.63	14.06	81351.99	13.96
40	82732.13	19.64	80683.22	17.90	83026.13	17.34	81141.85	16.94
50	82518.48	26.36	80619.09	23.48	82780.81	20.40	81020.50	22.04
60	82273.13	31.15	80614.30	24.41	82563.04	24.80	80931.38	25.12
70	82106.83	33.36	80411.30	29.63	82412.95	30.11	80798.78	28.68
80	82085.43	39.87	80401.96	38.24	82325.90	35.04	80779.40	32.55
90	82042.30	42.85	80393.09	39.75	82266.19	40.63	80778.00	36.82
100	81957.96	46.30	80301.48	43.57	82167.61	41.92	80701.29	39.43
200	81749.39	87.54	80051.61	86.46	82027.38	81.85	80422.77	74.88
300	81705.17	116.50	79923.61	115.95	81957.40	118.52	80221.06	117.00
Average	82497.03	39.80	80571.98	37.27	82774.37	36.67	80957.90	35.12

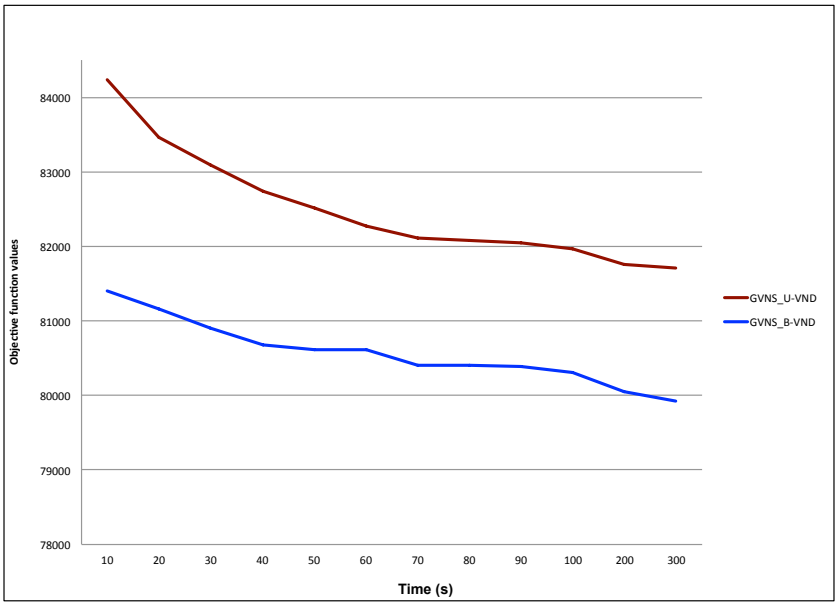

Figure 4: GVNS_U-VND versus GVNS_B-VND

From the results presented in Table 5 and Figure 4, we may draw the following conclusions.

(i) GVNS_B-VND significantly outperforms GVNS_U-VND regarding solution quality for each time limit imposed. It is worth mentioning that even with the time limit of 300 seconds GVNS_U-VND cannot reach neither the best nor the average solution quality obtained by GVNS_B-VND after 10 seconds of the execution (e.g., compare the average of best solution values found by GVNS_U-VND after 300 seconds of computation, i.e., 81705.17 and the average of best solution values found by GVNS_U-VND after 10 seconds of computation, i.e., 81401.09).

(ii) Comparing CPU times consumed by each GVNS variant, it follows that GVNS_B-VND is slightly faster than GVNS_U-VND (e.g., compare average of CPU times when best found solutions are encountered, i.e., 39.80 of GVNS_U-VND and 37.27 of GVNS_B-VND as well as average CPU times to solve an instance, i.e.,36.67 of GVNS_U-VND and 35.12 of GVNS_B-VND).

All in all, these observations, undoubtable confirm the superiority of GVNS_B-VND) over GVNS_U-VND. Such outcome may be explained by the fact that U-VND used within GVNS_U-VND is very slow compared to B-VND as demonstrated in the previous subsection. In addition, there is high probability that U-VND may be got stuck in some local optima valley, since at each iteration it generates local optimum with respect to three neighborhood structures. Thus, we may conclude that in order to achieve high performance of GVNS, it is preferable to use VND variant that in all iterations but the last re-center search around new solution which is not local optimum for the union of neighborhoods.

4 Concluding remarks

The purpose of this paper is to experimentally compare different variants of sequential Variable neighborhood descent (VND) and their influence on the solution qualities when they are used as local searches within General variable neighborhood search (GVNS). The problem chosen to execute such comparison is classical widely studied combinatorial optimization problem, i.e., Travelling salesman problem (TSP). As neighborhood structures we choose also those based on most commonly used moves that in addition have the same size $O(n^2)$: 2-opt, 1-insertion and 2-insertion. Moreover, we wanted to test the solution qualities in the following cases: first vs best improvement strategies; random vs greedy initial solution; does order of neighborhoods play significant role.

Answers to all those questions are mostly expected. Some of them are: There is no significant difference between proposed VND variants, although the most successful on average is GVNS that uses Basic VND; Union VND (that uses union of all neighborhoods as a single one) is too slow and provides the worst behavior when used within GVNS; use of greedy initial together with the best improvement strategy after, appear to be more successful than other variants; there is no significant difference among variants that use different order in a descent, but slightly better results have been reported when the 2-opt neighborhood structure is used first.

Future work may include empirical study on two another variants of VND, i.e., Nested VND and Mixed Nested VND, that explore more complex neighborhoods than the ones considered in this paper.

References

[1] Hansen, P., Mladenović, N., 2006. First vs. best improvement: An empirical study. Discrete Applied Mathematics 154(5), 802–817.

[2] Hansen, P., Mladenović, N., Pérez, J.A.M., 2010. Variable neighbourhood search: methods and applications. Annals of Operations Research 175(1), 367–407.

[3] Ilić, A., Urošević, D., Brimberg, J., Mladenović, N., 2010. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. European Journal of Operational Research 206(2), 289–300.

[4] Lazic, J., Todosijevic, R., Hanafi, S., Mladenovic, N., 2015. Variable and Single Neighbourhood Diving for MIP Feasibility, Yugoslav Journal of Operations Research, doi:10.2298/YJOR140417027L.

[5] Johnson, D.S., McGeoch, L.A., 1997. The traveling salesman problem: A case study in local optimization. Local Search in Combinatorial Optimization 1, 215–310.

[6] Lu, Z., Hao, J.K., Glover, F., 2011. Neighborhood analysis: A case study on curriculum-based course timetabling. Journal of Heuristics 17(2), 97–118.

[7] Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operations Research 24(11), 1097–1100.

[8] Reinelt G., 1991. TSPLIB – A traveling salesman problem library. ORSA Journal on Computing 3, 376–384.

[9] Todosijević, R., Urošević, D., Mladenović, N., Hanafi, S., 2015. A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. Optimization Letters, doi:10.1007/s11590-015-0867-6.

[10] Wu, Q., Hao, J.K., Glover, F., 2012. Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operations Research 196(1), 611–634.