Yugoslav Journal of Operations Research
26 (2016), Number 1, 3–16
DOI: 10.2298/YJOR140928038M

VARIABLE NEIGHBORHOOD SEARCH FOR MINIMUM LINEAR ARRANGEMENT PROBLEM

Nenad MLADENOVIĆ
LAMIH, University of Valenciennes, France
Nenad.Mladenovic@univ-valenciennes.fr

Dragan UROŠEVIĆ
Mathematical Institute SANU, Belgrade
draganu@mi.sanu.ac.rs

Dionisio PÉREZ-BRIT
Departamentos de Estadística, Investigación Operativa y Computación,
Universidad de La Laguna, Spain.
Dperez@ull.es

Received: September 2014 / Accepted: December 2014

Abstract: The minimum linear arrangement problem is widely used and studied in many practical and theoretical applications. It consists of finding an embedding of the nodes of a graph on the line such that the sum of the resulting edge lengths is minimized. This problem is one among the classical NP-hard optimization problems and therefore, there has been extensive research on exact and approximative algorithms. In this paper we present an implementation of a variable neighborhood search (VNS) for solving minimum linear arrangement problem. We use Skewed general VNS scheme witch appeared to be successful in solving some recent optimization problems on graphs. Based on computational experiments, we argue that our approach is comparable with the state-of-the-art heuristic.

Keywords: Graphs, Optimization, Minimum linear arrangement problem, Variable neighborhood search.

MSC: 90B06, 90C05, 90C08.

1. INTRODUCTION

Let $G = (V, E)$ be an undirected graph, where V is a set of vertices, and E be a set of undirected edges. Let *labelling* be any bijective mapping f of V into itself: $V \rightarrow \{1, ..., n\}$ ($n = |V|$). For a given labelling f one can calculate cost of mapping C_f as:

$$C_f = \sum_{(u,v)\in E} |f(u) - f(v)|$$

Minimum arrangement problem (MinAP) consists of finding labelling f whose cost C_f is minimal.

Let us denote with $\mathcal{S}$ a set of all valid labelling. We suppose that vertices of a graph are denoted by numbers $1, 2, ..., n$, then any solution can be presented as array (or vector) f with n elements, where $f[i]$ represents the label of vertex i. We also use inverse operator of f denoted as $g = f^{-1}$. In other words, $g[j]$ is the vertex whose label is j (i.e. $f[g[j]] = j$, and $g[f[j]] = j$).

MinLA is also connected with graph drawing: A bipartite drawing (or 2-layer drawing) is a graph representation where the nodes of a bipartite graph are placed in two parallel lines and the edges are drawn with straight lines between them. The bipartite crossing number of a bipartite graph is the minimal number of edge crossings over all bipartite drawings. Pach et al. [32] show that for a large class of bipartite graphs, reducing the bipartite crossing number is equivalent to reducing the total edge length, that is, to the minimum linear arrangement problem. Moreover, an approximate solution of MinLA can be used to generate an approximate solution to the Bipartite crossing number problem [35].

Given its importance, MinLA has been the object of extensive research that can be divided into two classes: exact and heuristic algorithms. Exact algorithms guarantee always to discover the optimal label. There exist polynomial time exact algorithms for some special cases of MinLA such as trees, rooted trees, hypercubes, meshes, outer-planar graphs, and others (see [10, 33] for a detailed surveys). However, as is the case with many graph layout problems, finding the minimum linear arrangement is known to be NP-hard for general graphs [16]. Therefore, there is a need for heuristics to address this problem in a reasonable time. Heuristic algorithms try to find good solutions as fast as possible, but they do not guarantee the optimality of the solution found. Many heuristics and specialized meta-heuristics have thus been proposed and moderate size problem instances may today be tackled quite efficiently. Table 1 summarizes the most relevant heuristics to this problem.

The VNS and its variants [24] have proved to be very successful in that respect. Large problem instances are still difficult to address, however. The aim of our current development is to expand the VNS methodology for global optimization by implementing different search strategies to apply its underlying framework to the Matrix bandwidth minimization problem [31]. In this paper we combine two variants of VNS, namely General VNS and Skewed VNS to get Skewed general VNS (SGVNS) based heuristic for solving MinLA. Combining those two VNS

Reference	Contribution
Harper (1964) [18]	Optimal assignment method
Juvan and Mohar (1992)[26]	Spectral Sequencing method (SSQ)
McAllister (1999) [29]	Constructive procedure that labels vertices in a sequential order
Petit (2003) [33]	Successive Augmentation (SAG), Simulating Annealing (SA)
Petit (2003) [34]	A more elaborate Simulating Annealing (SA)
Rodriguez-Tello et al. (2006) [36]	Memetic algorithms
Rodriguez-Tello et al. (2007)[37]	Two-Stage Simulating Annealing (TSSA)
J.J.Pantrigo et al. (2008) [32]	GRASP and Path Relinking

Table 1: Relevant heuristics literature related to the MinLA

variants appeared to be efficient in solving some hard combinatorial problems on graphs (see e.g. [7, 30]). As will be shown in the computational experiments, the proposed procedure provides high quality solutions within a short computational time.

The next section recalls the main VNS methodology and provides a description how to solve the MinLA based on the VNS framework. Section 3 presents and analyzes the results of our computational testing over a set of previously reported instances, and section 4 concludes the paper.

2. VARIABLE NEIGHBORHOOD SEARCH FOR THE MINLA

The basic idea of VNS is to implement a systematic change of neighborhood within a local search algorithm (see Hansen & Mladenović [19, 20, 21, 22, 23] and Hansen, Mladenović & Moreno [24]). Exploration of these neighborhoods can be done in two ways. The first one consists of systematically exploring the small neighborhoods, i.e. those closest to the current solution, until a better solution is found. The second consists of partially exploring the large neighborhoods, i.e., those far from the current solution, by drawing a solution at random from them and beginning a (variable neighborhood) local search from there. The algorithm remains in the same solution until a better solution is found and then it jumps there. These algorithms rank the neighborhoods to be explored in such a way that they are increasingly far from the current solution. We may view VNS as a way of escaping from local optima, where movement to a neighborhood further from the current solution corresponds to a harder shake. In contrast to random restart, VNS allows a controlled level of the shake.

2.1. Initial solution

As noted earlier, we save solution in the computer memory by using two arrays with length n: labelling array f, and its inverse $g = f^{-1}$. The initial solution (arrays f and g) is obtained by using some usual greedy add heuristic. Its pseudo-code is given at Algorithm 1. The set of unlabelled vertices is denoted with U.

Initially, all vertices are unlabelled. The number of labelled vertices adjacent to u is denoted as $nlab(u)$. Degree of each vertex u, $deg(u)$, is calculated as the cardinality of the set $N(u)$ (vertices adjacent to u in graph G). In its first iteration, the vertex with the minimum degree u gets label 1 ($f(u) = 1, g(1) = u$). It is then removed from the set of unlabelled vertices U and the remaining graph is updated: the number of labeled vertices $nlab(u)$ of all its adjacent vertices is increased by 1; the degree of unlabelled vertices adjacent to u are decresed by 1. New updated graph is considered in the next iteration. This procedure is iterated until all vertices are labelled.

Algorithm 1: Greedy add initial solution procedure

Function `InitialSolution` (f, g)
$k \leftarrow 1$
$U \leftarrow \{1, 2, ..., n\}$
for $u \in V$ **do**
  $nlab[u] \leftarrow 0$
  $deg[u] \leftarrow |N(u)|$
while $U \neq \emptyset$ **do**
  $u \leftarrow \arg\min\{deg[u] - nlab[u] \mid u \in U\}$
  $f[u] \leftarrow k$, $g[k] \leftarrow u$
  $U \leftarrow U \setminus \{u\}$
  for $v \in U \cap N(u)$ **do**
    $nlab[v] \leftarrow nlab[v] + 1$
    $deg[v] \leftarrow deg[v] - 1$
  $k \leftarrow k + 1$

We have also tested some other methods for getting initial solution which resulted in inferior performance. One among them is to take initial solution labels as $1, \ldots, n$ and another by choosing labels at random.

2.2. Sequential variable neighborhood descent (VND)

We design local search that uses four neighborhood structures in a sequential deterministic way. They are denoted as $N_k, k = 1, 2, 3, 4$. Those four neighborhoods are ordered and always used in that order. In this subsection we also show how to modify/update the objective function when the current solution is moved to a neighboring point.

Neighborhood $N_1(f)$. $N_1(f)$ consists of labelling f' obtained from labelling f by swapping labels of two vertices, but not any two vertices. For a given vertex u we enumerate all vertices connected to u according to a current labelling. In other words, let $v_1, v_2, ..., v_k$ represent all vertices connected to u, i.e., let

$$f(v_1) < f(v_2) < f(v_3) < \cdots < f(v_k).$$

We define median label as:

$$fm_u = \begin{cases} f(v_{l+1}) & \text{if } k = 2l+1 \\ \left\lfloor \frac{f(v_l)+f(v_{l+1})}{2} \right\rfloor & \text{if } k = 2l. \end{cases}$$

Neighborhood $N_1(f)$ consists of labelling obtained by swapping labels of all vertex pairs u and v ($u, v \in V, u \neq v$) such that the following condition is satisfied

$$v \in N(u) = \{w | fm_u - r \leqslant f(w) \leqslant fm_u + r\}.$$

Here r is a parameter whose value is set to 8 after a brief testing. Note that the similar moves are used in TSSA, but for randomly selected u and for $r = 2$. Detailed steps are given in Algorithm 2.

Algorithm 2: First improvement local search in neighborhood N_1

```
Function LocalSearcN1 (f, g, r)
impr ← false
for u ∈ V do
    SortByValueOfF(N(u), f)
    na ← |N(u)|
    if odd(na) then
        fm ← f[(na+1)/2]
    else
        fm ← ⌊(f[na/2]+f[na/2+1])/2⌋
    for k ∈ [max{0, fm − r}, min{n − 1, fm + r}] do
        v ← g[k]
        Δ ← ComputeSwap(f, g, u, v)
        if Δ < 0 then
            MakeSwap(f, g, u, v)
            impr ← true
            break
return impr
```

Enumeration of vertices in the graph (the outer loop in `LocalSearchN1` procedure) is repeated until an improvement is obtained ($impr \leftarrow$ `true`). Therefore, our local search applies the first improvement strategy, i.e., the procedure stops when an improvement in the objective function is found. If there is no $u \in V$ that brings better labelling, logical variable *impr* keeps *false* value, indicating that the local minimum is found.

An efficient calculation of change in the objective function value in point (labelling) that belongs to $N_1(f)$ is given in the procedure `ComputeSwap`, presented in Algorithm 3. If the calculated change value, denoted by dif is negative, then the improvement is reached.

Algorithm 3: Calculation of the change in the objective function value after swapping labels of vertices u and v in a given labelling f

```
Function ComputeSwap (f, g, u, v)
dif ← 0
for w ∈ N[u] do
    if w ≠ v then
        dif ← dif − |f[w] − f[u]|
        dif ← dif + |f[w] − f[v]|
for w ∈ N[v] do
    if w ≠ u then
        dif ← dif − |f[w] − f[v]|
        dif ← dif + |f[w] − f[u]|
return dif
```

Clearly, the complexity of `ComputeSwap` (.) presented in Algorithm 3 is in $O(d_{max})$, where d_{max} denotes the maximum degree of the graph. Note that pseudo code for `MakeSwap` procedure is omitted since it is obvious.

Neighborhood $N_2(f)$. $N_2(f)$ consists of labelling f' obtained by swapping labels of two vertices u and v whose labels in f are relatively close:

$$|f(u) - f(v)| \leqslant r_2$$

Intuitively, if we swap vertices whose labels are far then probability that the new labelling has smaller cost is relatively small. In our experiments r_2 is set to 10.

Algorithm 4: Local search in Neighborhood N_2

```
Function LocalSearcN2 (f, g, r)
impr ← false
for u ∈ V do
    for k ∈ [max{0, f[u] − r}, min{n − 1, f[u] + r}] do
        v ← g[k]
        Δ ← ComputeSwap(f, g, u, v)
        if Δ < 0 then
            MakeSwap(f, g, u, v)
            impr ← true
            break
return impr
```

`LocalSearchN2` also use routines `ComputeSwap` and `MakeSwap`, explained earlier.

Neighborhood $N_3(f)$. We also implement neighborhoods N_3 (rotate left, or insert forward) and N_4 (rotate right, or insert backward). Let u and v be vertices such that $f(u) < f(v)$. We can enumerate vertices

$$w_0 = u, w_1, w_2, ..., w_{k-1}, w_k = v$$

such that $f(w_{i+1}) = f(w_i) + 1$. New labelling f' can be obtained by decreasing label of vertices $w_1, w_2, ..., w_k$ by 1, and label vertex $w_0 = u$ with $f(v)$. Neighborhood $N_3(f)$ consists of all labelling obtained by using described method. We restrict the set of possible moves by introducing a condition

$$f(v) - f(u) \leqslant r_3$$

Algorithm 5: Local search in Neighborhood N_3

```
Function LocalSearcN3 (f, g, r)
impr ← false
for b ∈ [0, n − r] do
    for e ∈ [b + 2, min{n − 1, b + r}] do
        Δ ← ComputeRotL(f, g, b, e)
        if Δ < 0 then
            MakeRotL(f, g, b, e)
            impr ← true
            break
return impr
```

`LocalSearchN3` use two subroutines: `ComputeRotL` and `MakeRotL`. It is clear that the first one updates the difference in objective function value after *left rotation* (or *forward insertion*) move. The details are presented in Algorithm 6.

¿From pseudo-code it is clear that labels of vertices currently labelled with values between $b + 1$ and e decrease their values by 1. Also, vertex currently labelled with b obtains the new label e. Note that values for b and e are known. They are obtained as inner and outer loop counters coming from the procedure `LocalSearch3`().

Neighborhood $N_4(f)$. Neighborhood structure N_4 is similar to N_3: for a sequence of vertices

$$w_0 = u, w_1, w_2, ..., w_{k-1}, w_k = v$$

satisfying conditions $f(w_{i+1}) = f(w_i) + 1$ $(i = 0, 1, ..., k - 1)$, we consider neighbor whose label f' is obtained by increasing labels of vertices $w_0, ..., w_{k-1}$ by one and setting label of w_k to $f(w_0)$. We will not give here pseudo-code for the `LocalSearchN4`() since it is very similar to `LocalSearchN3`().

Sequential Variable neighborhood descent (VND). As mentioned earlier, deterministic variant of VNS is called VND. Its sequential version makes a list of

Algorithm 6: Compute change in the objective function value after left rotation move

```
Function ComputeRotL (f, g, b, e)
dif ← 0
u ← g[b]
for w ∈ N[u] do
    if f[w] < b or f[w] > e then
        dif ← dif − |f[w] − b|
        dif ← dif + |f[w] − e|
for t ∈ [s + 1, e] do
    v ← g[t]
    for w ∈ N[v] do
        if f[w] < b then
            dif ← dif − 1
        if f[w] > e then
            dif ← dif + 1
        if w = u then
            dif ← dif − (f[v] − b)
            dif ← dif + (e − f[v] + 1)
return dif
```

different neighborhood structures and use them one after another. The pseudocode for Variable neighborhood descent (VND) for solving MinLA problem that uses 4 neighborhoods is presented in Algorithm 7.

2.3. Shaking

We define a new set of neighborhoods $\mathcal{N}_k$ ($k = 1, \ldots, k_{max}$) to be used in the perturbation or shaking phase of VNS. $\mathcal{N}_k$ may be defined as k repetitions of one move with respect to one neighborhood structure. We could use for example one among 4 neighborhoods N_j ($j = 1, 2, 3, 4$) already described. After a brief experimentation we decided to use moves obtained from the neighborhood N_1: randomly select vertex u and vertex $v \in N(u)$ (see description of N_1), and swap their labels. Repeat this step k times to get random solution that belongs to $\mathcal{N}_k$. Details are depicted in Algorithm 8.

Indeed, the procedure repeats k times all steps of of `LocaSearchN1()`, but not systematically; it is applied only on a randomly chosen neighboring point.

2.4. Skewed GVNS

General VNS (GVNS) is a variant of VNS that includes Variable neighborhood descent (VND) as a local search within basic VNS. On the other hand, Skewed VNS (SVNS) extends the sharp move acceptance criterion (`Neighborhood change`

Algorithm 7: Sequential Variable neighborhood descent

```
Function VND (f, g)
k ← 1
while k ⩽ 4 do
    switch k do
        case 1 do
            impr ← LocalSearchN1(f, g, r_1)
        case 2 do
            impr ←LocalSearchN2(f, g, r_2)
        case 3 do
            impr ←LocalSearchN3(f, g, r_3)
        case 4 do
            impr ←LocalSearchN4(f, g, r_4)
    if impr then
        k ← 1
    else
        k ← k + 1
```

step of the Basic VNS). Within SVNS deteriorating moves are also accepted as next current solutions, but only if they are relatively far from the incumbent. For that purposes, one needs to introduce distance function or to supply the solution space with some metric function. Combining these two VNS variants we get Skewed general VNS (SGVNS) that has recently showen excellent results in solving Maximum diversity grouping problem [40, 7] and Clique partitioning problem [8].

Here we make a move from the solution f to the solution f'' (obtained after shaking and local search) if

$$C_{f''} < C_f(1 + \alpha \times d(f, f'')).$$

or equivalently if

$$\frac{C_{f''}}{C_f} < 1 + \alpha \times d(f, f'').$$

With $d(f, f'')$ we denote the distance between labelling f and f''. We use the following formula for calculating distance d:

$$d(f, f'') = \frac{1}{|V|} \sum_{u \in U} |f(u) - f''(u)|$$

Parameter α is set to 0.005. It can easily be shown that the function $d(.,.)$ satisfies metric axioms. Pseudo–code is given in the Algorithm 9.

Algorithm 8: Shaking

Procedure `Shaking` (k, f, g)
$r \leftarrow 8$
while $k > 0$ **do**
  $u \leftarrow$ `RandomVertex`()
  `SortByValueOfF`$(N(u), f)$
  $na \leftarrow |N(u)|$
  if *odd(na)* **then**
    $fm \leftarrow f\left[\frac{na+1}{2}\right]$
  else
    $fm \leftarrow \left\lfloor\frac{f[na/2]+f[na/2+1]}{2}\right\rfloor$
  $\ell \leftarrow$ `RandomNumber`$[\max\{0, fm-r\}, \min\{n-1, fm+r\}]$
  $v \leftarrow g(\ell)$
  `MakeSwap`(f, g, u, v)
  $k \leftarrow k-1$

3. COMPUTATIONAL RESULTS

In this section, we compare the results of our Skewed general VNS based algorithm (SGVNS) with the best existing algorithm from the literature, i.e., with TSSA [37]. We wanted to make comparison on the same computer and we asked authors to provide us with either source or executable versions of their code. Since they could not find any version, we decided to code their relatively easy method by ourself. Our implementation of TSSA below. is denoted as TSSA1.

Instances proposed by Petit (available on `www.lsi.upc.es/~jpetit/MinLA/Experiments`) are used. Running time is used as a stopping condition and set to $t_{max} = 2000$ seconds for all instances. After detailed testing the value of $k_{\max}$ of our SGVNS is set to 30. Both methods, i.e., SGVNS and TSSA1, are implemented in C (compiled with GNU C++) and run on Pentium Core Duo based (2.66Gz) computer under Linux operating system.

The first column of Table 3 contains the name of the problem. The best known objective function value is given in the second column (note that this results are also obtained by TSSA as reported in [37]). Columns 3 to 4 contain results obtained by SGVNS and our implementation of TSSA. Columns 5 to 7 contain CPU times until obtaining reported results for corresponding methods. Last two columns contain percentage deviation of SGVNS and our implementation of TSSA over solutions reported in [37]. The results reported in Table 3 are best obtained in 20 executions on each instance, but with the different seed values for the random number generator.

According to computatational results reported in Table 3, the two-stage Simulating annealing based heuristic (TSSA) reaches the best known values for each test instance. It appears:

Algorithm 9: SGVNS

```
Function SGVNS (k_max, t_max)
InitialSolution(f, g)
f_opt ← f, k ← 1, t ← 0
while t ⩽ t_max do
    (f', g') ← (f, g)
    Shake(k, f', g')
    VND(f', g')
    if C(f')/C(f) < 1 + αd(f, f') then
        if C(f') < C(f_opt) then
            f_opt ← f'
        (f, g) ← (f', g')
        k ← 1
    else
        k ← k + 1
        if k > k_max then
            k ← 1
    t ← ElapsedTime
return f_opt
```

(i) the original and our implementation of TSSA are very different. The average solution qualities between them is 18.48% in favor to original implementation. Average running time is also much smaller for TSSA. We have no explanation for that since we could not see the original code. Our implementation of TSSA (i.e., TSSA1) may be found at the following web page: `http://mi.sanu.ac.rs/~nenad/minla`.

(ii) SGVNS significantly outperforms TSSA1 in terms of solution quality; compare 3.48% and 18.48% deviations above the best known obtained by SGVNS and TSSA1 respectively.

(iii) The running times when the best solutions are reached (within 2000 seconds) of both methods are comparable, although average time spent by SGVNS is slightly shorter (compare average times of 1346.51 sec and 1392.09 sec obtained by SGVNS and TSSA1, respectively.

4. CONCLUSIONS

In this paper we present a VNS based heuristic for solving the minimum linear arrangement problem MinLA. We implement Skewed general VNS (SGVNS) scheme, the VNS variant that has recently shown very good results. In its descent or intensification phase, this algorithm explores four neighborhood structures. Efficient updating of the objective function values at the points from each of those

Instance name	Best kn (TSSA)	Obj. values		Time (in seconds)			% dev.	
		SGVNS	TSSA1	SGVNS	TSSA1	TSSA	SGVNS	TSSA1
randomA1	866968	884329	926995	1924.81	1487.72	86.50	2.00	6.92
randomA2	6522206	6650675	6693724	1895.12	1980.60	181.00	1.97	2.63
randomA3	14194583	14369075	14511188	1717.07	1989.76	279.10	1.23	2.23
randomA4	1717176	1760334	1782333	1592.23	1969.45	90.00	2.51	3.79
hc10	523776	523776	523776	0.02	0.01	1.20	0.00	0.00
mesh33x33	31856	32667	39009	1911.94	1983.79	89.90	2.55	22.45
3elt	359151	381738	776438	2000.03	1994.90	1030.80	6.29	116.19
airfoil1	276381	299549	563674	1998.61	1999.85	982.10	8.38	103.95
whitaker3	1143645	1232190	1297786	1999.79	1978.48	3330.10	7.74	13.48
c1y	62230	62987	67071	1177.96	352.86	32.80	1.22	7.78
c2y	78757	79573	81592	1748.44	1899.11	46.70	1.04	3.60
c3y	123145	132053	148669	1904.81	1487.70	93.30	7.23	20.73
c4y	114936	117629	129124	1886.27	1783.73	88.10	2.34	12.34
c5y	96850	100214	110950	1974.62	1843.95	69.20	3.47	14.56
gd95c	506	509	506	0.03	113.61	2.10	0.59	0.00
gd96a	95263	105461	108859	1744.57	1959.36	61.00	10.71	14.27
gd96b	1416	1491	1478	3.75	255.25	2.90	5.30	4.38
gd96c	519	519	519	0.04	81.46	0.30	0.00	0.00
gd96d	2391	2428	2434	103.61	1288.14	6.70	1.55	1.80
Average	1379566	1407221	1461375	1346.51	1392.09	340.73	3.48	18.48

neighborhoods is proposed, including the worst case analysis of such calculations. Since the source or the executable versions of the current state-of-the-art heuristic TSSA are not available, we made our own implementation of TSSA, which we called TSSA1. It appeared that these two algorithms, i.e. TSSA and TSSA1, show very different characteristics: TSSA was always better despite the fact that TSSA is relatively easy to code. That is why, in Computational experiments (on Petit instances available on `www.lsi.upc.es/~jpetit/MinLA/Experiments`) we compare our SGVNS with both TSSA and TSSA1. It appeared that all the best known TSSA results remained best, although it was not clear how they are obtained. On the other hand, our SGVNS showed much better performances than TSSA1 in both average solution quality and the running times.

We plan to extend this approach in the future in three directions: (i) we plan to change parameter r, that is now fixed in advance within all four neighborhoods, during the execution of the code, generating new neighborhood structures of the problem; (ii) we implemented sequential VND algorithm as a local search. However, it is clear that the nested or mixed nested could be used to increase intensification [25, 38]; (iii) recent extended variants of VNS could be tried out as well [27, 28, 17, 39]

Acknowledgements: Work of Nenad Mladenovic is conducted at National Research University Higher School of Economics, Russia and supported by RSF

grant 14-41-00039. The third author has been partially supported by Gobierno de Canarias grant SolSubC200801000048.

REFERENCES

[1] Aarts E. H. and Korst J. H., *Simulated Annealing and Boltzmann Machines:A Stochastic Approach to Combinatorial Optimization and Neural computing*, John Wiley and Sons, 1989.

[2] Aarts E. H. and Laarhoven P. J. V., "Statistical cooling: A generalapproach to combinatorial optimization problem", *Philips Journal of Research*, 40 (1985) 193–226.

[3] Adolphson D. and Hu T. C., "Optimal linear ordering", *SIAM Journal on Applied Mathematics*, 25(3) (1973) 403–423.

[4] Bar-Yehuda R., Even G., Feldman J., and Naor J., "Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems", *Journal of Graph Algorithms and Applications*, 5(4) (2001) 1–27.

[5] Beasley J., "OR-library: distributing test problems by electronic mail", *Journal of the Operational Research Society*, 41 (1990) 1069-1072. Available at: `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files`.

[6] Boese K. D. and Kahng A. B., "Best-so-far vs. where-you-are: Implications for optimal finite-time annealing", *Systems and Control Letters*, 22(1) (1994) 71–78.

[7] Brimberg J., Mladenović N., Urošević D., "Solving the maximally diverse grouping problem by skewed general variable neighborhood search", *Information Sciences*, 295 (2015) 650–675, `http://dx.doi.org/10.1016/j.ins.2014.10.043`.

[8] Brimberg J., Janićijević S., Mladenović N., Urošević D., "Solving the Clique Partitioning Problem as a Maximally Diverse Grouping Problem" (submitted for publication).

[9] Charon I. and Hudry O., "The noising method: A new method for combinatorial optimization", *Operations Research Letters*, 14(3) (1993) 133–137.

[10] Diaz J., Petit J., and Serna M., "A survey of graph layout problems", *ACM Computing Surveys*, 34(3) (2002) 313–356.

[11] Daskin M., *Network and Discrete Location: Models, Algorithms, and Applications*, Wiley, 1995.

[12] Domínguez-Marín P., The Discrete Ordered Median Problem: Models and Solution Methods, PhD thesis, University of Kaiserslautern, 2003.

[13] Domínguez-Marín P., Nickel S., Hansen P., and Mladenović N., "Heuristic procedures for solving the discrete ordered median problem", *Annals of Operations Research*, 136 (2005) 145–173.

[14] Drezner Z. and Hamacher H., Facility Location: Applications and Theory, Springer Berlin Heidelberg New York, 2002.

[15] Falkenauer E., "A hybrid grouping genetic algorithm for bin packing", *Journal of Heuristics*, 2 (1996) 5–30.

[16] Garey M. R. and Johnson D.S., *Computers and Intractability: A guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[17] Hanafi S., Lazic J., Mladenovic N., Wilbaut C., Crevits I., "New VNS based 0-1 MIP Heuristics", *Yugoslav Journal of Operations Research*, DOI: 10.2298/YJOR140219014H.

[18] Harper L. H., "Optimal assignment of numbers to vertices", *SIAM Journal on Applied Mathe matics*, 12(1) (1964) 131–135.

[19] Hansen P. and Mladenović N., "Variable neighborhood search for the *p*-median", *Location Science*, 5(4) (1997) 207–226.

[20] Hansen P. and Mladenović N., *Developments of Variable Neighborhood Search*, In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439, Kluwer Academic Publishers, 2001.

[21] Hansen P. and Mladenović N., "Variable neighborhood search: Principles and applications", *European Journal of Operational Research*, 130(3) (2001) 449–467.

[22] Hansen P. and Mladenović N., *Variable Neighborhood Search*, In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, 57, Boston, MA: Kluwer Academic Publishers, 2003.

[23] Hansen P., Mladenovic N. and Pérez-Brito D., "Variable Neighborhood Decomposition Search", *Journal of Heuristics*, 7 (2001) 335–350.

[24] Hansen P., Mladenovic N. and Moreno Pérez J.A., "Variable neighborhood search: methods and applications", *Annals of Operations Research*, 175 (2010) 367–407.

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

[26] Juvan M. and Mohar B., "Optimal linear labelings and eigenvalues of graphs", *Discrete Applied Mathematics*, 36(2) (1992) 153–168.

[27] Kritzinger S., Doerner K. F., Tricoire F., Hartl R. F., "Adaptive search techniques for problems in vehicle routing, Part I: A survey", *Yugoslav Journal of Operations Research*, DOI: 10.2298/YJOR140219014H.

[28] Kritzinger S., Doerner K. F., Tricoire F., Hartl R. F., "Adaptive search techniques for problems in vehicle routing, Part II: A numerical comparison", *Yugoslav Journal of Operations Research*, DOI: 10.2298/YJOR140217011K.

[29] McAllister A.J., "A new heuristic algorithm for the linear arrangement problem", *Technical Report TR-99-126a*, Faculty of Computer Science, University of New Brunswick.

[30] Mladenović N., Todosijević R. and Urošević D., "An efficient General variable neighborhood search for large TSP problem with time windows", *Yugoslav Journal of Operations Research*, 22 (2012) 141–151.

[31] Mladenović N., Urošević! D., Perez-Brito D. and Garcia-Gonzalez!C. G., "Variable neighborhood search for bandwidth reduction", *European J of Operational Research* 200 (2010) 14-27.

[32] Pantrigo J. J., Duarte A., Campos V. and Martí R., "Heuristic for Minimum Linear Arrangement Problem", *Working Paper*

[33] Petit J., "Experiments on the minimum linear arrangement problem", *ACM Journal of Experimental Algorithmics*, 8 (2003).

[34] Petit J., "Combining spectral sequencing and parallel simulated annealing for the MinLA problem", *Parallel Processing Letters*, 13(1) (2003) 77–91.

[35] Reinelt G., Seitz H., "On a binary distance model for the minimum linear arrangement problem", *TOP*, DOI10.1007/s11750-012-0263-7.

[36] Rodriguez-Tello E., Hao J.-K., and Torres-Jimenez J., "Memetic algorithms for the MinLA problem", *Lecture Notes in Computer Science*, 3871 (2006) 73–84.

[37] Rodriguez-Tello E., Hao J., Torres-Jimenez J., "An Effective Two-Stage Simulated Annealing Algorithm for the Minimum Linear Arrangement Problem", *Computers and Operations Research*, 35(10) (2008) 3331–3346.

[38] Todosijević R., Mladenović N., Urošević D., Hanafi S., "A general variable neighborhood search for solving the uncapacitated r-allocation *p*-hub median problem" (to appear).

[39] Urošević!D., Brimberg!J., Mladenović N., "Variable neighborhood decomposition search for the edge weighted *k*–cardinality tree problem", *Computers & Operations Research*, 31(8) (2004) 1205–1213.

[40] Urošević D., "Variable neighborhood search for Maximum diverse grouping problem", *Yugoslav Journal of Operations Research*, 24 (1) (2014) 21–33.