Yugoslav Journal of Operations Research
26 (2016), Number 2, 173–188
DOI: 10.2298/YJOR150124006D

GAUSSIAN VARIABLE NEIGHBORHOOD SEARCH FOR THE FILE TRANSFER SCHEDULING PROBLEM

Zorica DRAŽIĆ
Faculty of Mathematics, University of Belgrade, Serbia
zdrazic@matf.bg.ac.rs

Received: January 2015 / Accepted: March 2015

Abstract: This paper presents new modifications of Variable Neighborhood Search approach for solving the file transfer scheduling problem. To obtain better solutions in a small neighborhood of a current solution, we implement two new local search procedures. As Gaussian Variable Neighborhood Search showed promising results when solving continuous optimization problems, its implementation in solving the discrete file transfer scheduling problem is also presented. In order to apply this continuous optimization method to solve the discrete problem, mapping of uncountable set of feasible solutions into a finite set is performed.

Both local search modifications gave better results for the large size instances, as well as better average performance for medium and large size instances. One local search modification achieved significant acceleration of the algorithm. The numerical experiments showed that the results obtained by Gaussian modifications are comparable with the results obtained by standard VNS based algorithms, developed for combinatorial optimization. In some cases Gaussian modifications gave even better results.

Keywords: Combinatorial Optimization, Variable Neighborhood Search, Gaussian Variable Neighborhood Search, File Transfer Scheduling Problem.

MSC: 90C59, 68T20, 05C90.

1. INTRODUCTION

The standard form of the optimization problem is given with:

$$\min\{f(x) \mid x \in X, X \subseteq S\} \tag{1}$$

where S is a solution space, X is a set of feasible solutions, x is a feasible solution, and f is an objective function. In the case when S is a finite or countable infinite set, this is a problem of combinatorial (discrete) optimization. On the other hand, when $S \subseteq \mathbb{R}^n$, it is a problem of continuous optimization.

Variable Neighborhood Search (VNS) [11, 8] has shown to be a powerful tool for solving both discrete and continuous optimization problems. The basic idea of this method is to explore a set of predefined neighborhoods in search for a better solution. Systematically changing these neighborhoods, the algorithm tries to escape from the current local minimum. Shaking and local search steps perform this exploration. In [3] authors presented a new VNS-based heuristic for solving continuous optimization problems, so called Gauss-VNS. The main idea of this approach is to replace the class of neighborhoods $\{N_k(x)\}_{1\leq k\leq k_{\max}}$ centered at point x by a class of probability distributions $\{P_k(x)\}_{1\leq k\leq k_{\max}}$. For generating a random point in the neighborhood of the incumbent solution in shaking step, the Gaussian distribution was used. To the best of the authors knowledge, this approach for solving continuous optimization problems has never been used for any discrete problem. Since the results obtained by Gauss-VNS for continuous optimization were comparable with other heuristics and some advantages of this approach were shown, in this paper we considered the utilization of Gauss-VNS for discrete File transfer scheduling problem.

File transfer scheduling problem (FTSP) was introduced and proved to be NP-complete by E.G. Coffman et al. in [4]. In this problem, a number of large files should be transferred between various nodes of a computer network. For each file, the amount of time needed to perform its transfer is given. Each computer in the network has limited number of communication ports, i.e. the upper limit of number of file transfers it can handle at the same time. Forwarding is not allowed, meaning that each file has to be transferred directly from its starting point to its destination, without involving any other intermediary computer. Also, it is assumed that when the file transfer starts, it continues without any interruption until its completion. The objective of FTSP consists of finding a schedule of file transfers so the total time for transfer process is minimal.

The instance of FTSP consists of an undirected multigraph $G = (V, E)$, which is called the file transfer graph. The set of vertices V of the graph G correspond to the set of nodes of a computer network. Each vertex $v \in V$ is labeled by a positive integer $p(v)$, representing its port constraint, i.e. the sum of uploads and downloads which that particular node (computer) can handle simultaneously. The set of edges E correspond to the set of files that should be transferred. Each edge $e \in E$ is labeled by an integer $L(e)$, representing the amount of time (expressed in any of time units) needed for transferring that file, represented by the edge e.

For a given graph G, a schedule of file transfers can be represented as a function $s : E \to [0, \infty)$ which assigns a starting time $s(e)$ to each edge $e \in E$, such that for each vertex $v \in V$, and time $t \geq 0$

$$\left|\{e : v \text{ is an end point of } e \text{ and } s(e) \leq t < s(e) + L(e)\}\right| \leq p(v)\,. \tag{2}$$

The makespan length of a schedule s is defined as the largest finishing time moment (the first time moment when all file transfers are completed), i.e. the maximum of $s(e) + L(e)$ over all edges $e \in E$. Given a file transfer graph G, the goal is to find a schedule s with the minimum possible makespan.

The optimal schedule length $OPT(G)$ for any graph G must satisfy the constraint

$$OPT(G) \geq \max_u \lceil \sum\nolimits_u / p_u \rceil\,, \tag{3}$$

where $\sum_u = \sum_{e \in E_u} L(e)$ is the sum of times required for transferring all files from E_u, where E_u is the set of files which should be sent or received by the vertex u, and p_u denotes the port constraint for the vertex u. The right hand side of inequality (3) is called the elementary lower bound for FTSP.

File transfer scheduling problem and its modifications have been studied extensively [10, 15, 1, 9, 16]. A Variable Neighborhood Search algorithm for solving FTSP was designed in [5]. The experimental results were carried out on instances of up to 100 vertices and 500 files that should be transferred. Optimality of VNS solutions on smaller size instances has been verified by total enumeration, and for several larger instances optimality followed from reaching the elementary lower bound of the problem.

In [6], an integer linear programming (ILP) formulation for solving FTSP was introduced. Using CPLEX and Gurobi solvers, based on proposed ILP problem formulation, authors obtained results on extended set of instances. Numerical results showed that both solvers were able to produce optimal solutions on instances up to 40 vertices and 50 edges in reasonable time. This ILP formulation was improved in [7], so the CPLEX and/or Gurobi solvers were able to produce optimal solutions for some larger instances.

1.1. VNS for the file transfer scheduling problem

Variable Neighborhood Search has already been successfully used for solving various combinatorial optimization problems [12, 13, 14]. VNS proposed in [5], designed for solving the file transfer scheduling problem, was implemented as follows. The solution of FTSP was denoted as a n-dimensional vector $x = (x_1, ..., x_n)$ with integer values $0 \leq x_k \leq n-1$ where n is the number of edges in FTSP. Elements x_k of this vector represent the priorities assigned to each edge e_k in the graph.

The objective function value. For a given solution x, the objective function value was calculated in the following way. In the first step, the edges were sorted according to their priority. Using this sorted array, the unique schedule s was formed by starting from the time $t = 0$ and trying to find the first feasible edge

to which the starting time t will be assigned. The same procedure was repeated for the remaining unscheduled edges. If no edge could be started at time t, t was increased by 1, and the whole procedure was repeated until all edges e_k got their starting times t_k. For the obtained schedule, the value of the objective function $f(x)$ was defined as $f(x) = \max_{1 \leq k \leq n}(t_k + L(e_k))$.

Building an initial solution. An initial feasible solution x was taken as a random permutation of integer values $0, 1, ..., n-1$.

Neighborhood structures and shaking procedure. For given k, the neighborhood $N_k(x)$ contains all priority vectors which may differ from the current solution x in most k index positions. In the shaking step, a new solution $x' \in N_k(x)$ is generated by forming a vector $(i_1, ..., i_k)$ of k distinct random integers from the set $\{0, 1, ..., n-1\}$, which represent the indices of elements in the solution vector that will be changed. The random permutation of elements from the vector x at those positions is created by forming the vector $(p_1, ..., p_k)$. Finally, the vector x' is created from x by modifying its k elements as $x'(i_j) = p_j$, $j = 1, 2, ..., k$.

Local search procedure. The neighborhood of the solution x', which was explored for potentially better solution x'', consists of all vectors obtained from x' by swapping two of its elements. For this exploration, the first improvement search strategy was used. The local search terminates when the entire neighborhood of the current solution is searched, and no further improvement is possible.

After the local search, if the obtained solution x'' is better than x ($f(x'') < f(x)$), the algorithm moves to this new solution $x := x''$ and the search continues with the same neighborhood size $N_k(x)$. If $f(x'') > f(x)$, the search is repeated with the same x and the next neighborhood $N_{k+1}(x)$. In the case when $f(x'') = f(x)$, with some predefined probability p, the algorithm moves to this new solution and continues the search from it, and with probability $1 - p$ it repeats the search with the same x and the next neighborhood.

2. IMPROVED VNS FOR FILE TRANSFER SCHEDULING PROBLEM

In order to improve the local search procedure for VNS proposed in [5], two new local search implementations are presented in this paper.

2.1. Variable Neighborhood Descent

The idea for this local search modification is to explore first the solutions which are "close" to the solution x'. This strategy is based on the assumption that, in VNS representation of the solution of FTSP, "close" elements of the priority vector have greater correlation than the elements which are on a "larger" distance.

In order to implement this kind of exploration within the local search procedure, the set of smaller neighborhoods will be used instead of only one neighborhood. In this way, a General Variable Neighborhood Search (GVNS) is designed for FTSP. GVNS is a variant of VNS, where the deterministic Variable Neighborhood Descent (VND) is used as the local search.

The neighborhood which is explored in local search for a better solution starting from x' is divided into several smaller neighborhoods as follows:

- The neighborhood $\mathcal{N}_1(x')$ consists of all vectors obtained from the solution x' by swapping two of its consecutive elements.
- The neighborhood $\mathcal{N}_2(x')$ consists of all vectors obtained from the solution x' by swapping two of its elements, which have exactly one element between them, i.e. by swapping each $x'(i)$ and $x'(i+2)$, $i = 0, ..., n-3$.
- The neighborhood $\mathcal{N}_l(x')$, $l = 1, ..., n-1$, consists of all vectors obtained from the solution x' by swapping two of its elements with exactly $l-1$ elements between them.

The exploration of each neighborhood $\mathcal{N}_l(x')$, $l = 1, ..., n-1$ is faster than the exploration of the neighborhood for local search described in [5]. On the other hand, the union of all the neighborhoods $\mathcal{N}_1, ..., \mathcal{N}_{n-1}$ is equal to the neighborhood where each two of the elements are swapped, so no local minimum will be missed by this modification. Note that the neighborhood structures used in local search are different from the neighborhoods in the shaking procedure.

In this VND implementation, the local search procedure explores $n-1$ neighborhoods systematically as follows:

- Procedure starts searching for better solution by exploring the neighborhood $\mathcal{N}_1(x')$.
- If better solution is not found in the neighborhood $\mathcal{N}_k(x')$, $k \leq n-2$, the search continues in the next neighborhood $\mathcal{N}_{k+1}(x')$.
- If better solution is found in any neighborhood, it becomes new incumbent solution and the local search procedure starts over from the neighborhood $\mathcal{N}_1$ of this new solution.
- If better solution is not found in the last neighborhood $\mathcal{N}_{n-1}(x')$, the local search stops, and the best found solution is denoted as x''.

Performing the local search procedure in this way, $\mathcal{N}_1$ is the the most explored neighborhood since every time when a better solution is found in $\mathcal{N}_l$, $l = 2, ..., n-1$, the search continues from the neighborhood $\mathcal{N}_1$. In this way, "closer" solutions will be found more quickly. On the other hand, the search in the neighborhood $\mathcal{N}_l$ will be performed only when no improvement was found in all neighborhoods $\mathcal{N}_1, ..., \mathcal{N}_{l-1}$, which may not happen so often.

The details of the GVNS for the FTSP are given in a pseudo-code in Figure 1.

Example 2.1. *Consider the simple example of the local search procedure execution. Let the vector x', obtained from the shaking procedure, contain 5 elements: $x' = (1, 2, 3, 4, 5)$. Let the vectors $p1 = (1, 2, 4, 3, 5)$ and $p2 = (1, 4, 2, 3, 5)$ be the only solutions with smaller objective function value, such that $f(x') > f(p1) > f(p2)$. The local search procedure, defined in [5], consists of the next swap moves:* $\mathbf{x'} = (\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5})$, $(2, 1, 3, 4, 5)$, $(3, 2, 1, 4, 5)$, $(4, 2, 3, 1, 5)$, $(5, 2, 3, 4, 1)$, $(1, 3, 2, 4, 5)$, $(1, 4, 3, 2, 5)$, $(1, 5, 3, 4, 2)$, $\mathbf{p1} = (\mathbf{1}, \mathbf{2}, \mathbf{4}, \mathbf{3}, \mathbf{5})$. *After obtaining the first better solution $p1$, the search continues in the neighborhood of the*

/* *Initialization* */
Select a set of neighborhood structures N_k, $k = k_{\min}, \ldots, k_{\max}$
that will be used in the search;
Select a set of neighborhood structures $\mathcal{N}_l$, $l = 1, \ldots, n-1$
that will be used in the local search procedure;
Randomly choose an arbitrary initial point $x \in X$ and set
$x^* \leftarrow x$, $f^* \leftarrow f(x)$;
repeat the following steps **until** the stopping criterion is met
Set $k \leftarrow k_{\min}$;
1: **repeat** the following steps **until** $k > k_{\max}$
/* *Shaking* */
Generate at random a point $x' \in N_k(x^*)$;
/* *Local search (VND)* */
Set $l \leftarrow 1$;
repeat the following steps **until** $l > n-1$
/* *Explore the current neighborhood* */
Find the local minimum x'' in the neighborhood $\mathcal{N}_l(x')$;
/* *Move or not?* */
if $f(x'') < f(x')$ **then**
Set $x' \leftarrow x''$ and $l \leftarrow 1$;
else
Set $l \leftarrow l + 1$;
end
/* *Move or not?* */
if $f(x'') < f^*$ **then**
Set $x^* \leftarrow x''$, $f^* \leftarrow f(x'')$ and **goto** 1;
elseif $f(x'') > f^*$ **then**
Set $k \leftarrow k + 1$;
else
With probability p set $x^* \leftarrow x''$, $f^* \leftarrow f(x'')$ and **goto** 1,
and with probability $p - 1$ set $k \leftarrow k + 1$;
end
end
Stop. Point x^* is an approximative solution of the problem.

Figure 1: The GVND method for FTSP.

```
/* Local search */
Set l ← 1;
repeat the following steps until l > 1
    /* Explore the neighborhood N_1 */
    Find the local minimum x'' in the neighborhood N_1(x');
    /* Move or not? */
    if f(x'') < f(x') then
        Set x' ← x'' and l ← 1;
    else
        Set l ← l + 1;
end
```

Figure 2: The local search procedure for VNS-LS1 method for FTSP.

*solution p*1*:* $(2,1,4,3,5)$, $(4,2,1,3,5)$, $(3,2,4,1,5)$, $(5,2,4,3,1)$, $\mathbf{p2} = (\mathbf{1},\mathbf{4},\mathbf{2},\mathbf{3},\mathbf{5})$. *The total number of iterations is 13.*

On the other hand, the VND described above consists of the following swap moves: $\mathbf{x}' = (\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5})$, $(2,1,3,4,5)$, $(1,3,2,4,5)$, $\mathbf{p1} = (\mathbf{1},\mathbf{2},\mathbf{4},\mathbf{3},\mathbf{5})$, $(2,1,4,3,5)$, $\mathbf{p2} = (\mathbf{1},\mathbf{4},\mathbf{2},\mathbf{3},\mathbf{5})$. *The number of iterations is 5. As it can be seen, in this case VND algorithm reaches the local minimum much faster. In both cases, after obtaining the solution p*2 $= (1,4,2,3,5)$*, the same number of iterations will be performed until the local search procedure in (unsuccessful) search for possible better solution ends.*

2.2. Acceleration of the local search

The local search procedures, described in [5] and in Subsection 2.1, are profound since both explore the neighborhood of the solution x', which consists of swapping each of its two elements, just in a different order. The main drawback of these approaches is the long execution time. One possible way to speed up significantly the local search procedure is to reduce the neighborhood of x' that will be explored. In this way, less time will be spent in swapping the elements of vector x' and calculating the objective function value for each newly obtained vector.

Let the only neighborhood to be used in the local search procedure be the $\mathcal{N}_1$ neighborhood, described in Subsection 2.1. In this modification, the only neighborhood $\mathcal{N}_1(x')$ that will be explored in the local search procedure consists of all solutions that could be reached from x' by swapping two of its consecutive elements. If the better solution x'' is found, it becomes the new incumbent solution $x' := x''$, and the whole local search procedure starts again from the neighborhood $\mathcal{N}_1(x')$ of the new solution. If no improvement can be made, the local search is over.

The modification of VNS with local search procedure described above is denoted as VNS-LS1, and the details of corresponding local search are given in pseudo-code in Figure 2.

In this way, the neighborhood which will be explored in the local search is reduced, and VNS algorithm is significantly accelerated. On the other hand, relying only on the fact that the local minima are "close" to each other, it is possible to omit some better solutions at "larger" distance. Despite this potential flaw, it is possible to reach a better solution which, despite the thorough exploration, was not reachable by the local search procedures from [5]. The following simple example illustrates this observation.

Example 2.2. *Let the vector x', obtained from the shaking procedure, contain 5 elements: $x' = (1,2,3,4,5)$. Let the vectors $p1 = (1,2,4,3,5)$, $p2 = (4,2,1,3,5)$ and $p3 = (1,4,2,3,5)$ be the only solutions with smaller objective function values, such that $f(x') > f(p1) > f(p2) > f(p3)$. The local search procedure, defined in [5], consists of the following swap moves:* $\mathbf{x'} = (\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5})$, $(2,1,3,4,5)$, $(3,2,1,4,5)$, $(4,2,3,1,5)$, $(5,2,3,4,1)$, $(1,3,2,4,5)$, $(1,4,3,2,5)$, $(1,5,3,4,2)$, $\mathbf{p1} = (\mathbf{1},\mathbf{2},\mathbf{4},\mathbf{3},\mathbf{5})$. *After obtaining the solution $p1$, the search continues in the first neighborhood of the solution $p1$:* $(2,1,4,3,5)$, $\mathbf{p2} = (\mathbf{4},\mathbf{2},\mathbf{1},\mathbf{3},\mathbf{5})$. *Vectors $p2 = (4,2,1,3,5)$ and $p3 = (1,4,2,3,5)$ differ at tree positions, and it is clear that by swapping either two of them in $p2$, it is not possible to reach the solution $p3$ with smaller objective function value.*

On the other hand, the local search from VNS-LS1 algorithm consists of the following swap moves: $\mathbf{x'} = (\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5})$, $(2,1,3,4,5)$, $(1,3,2,4,5)$, $\mathbf{p1} = (\mathbf{1},\mathbf{2},\mathbf{4},\mathbf{3},\mathbf{5})$, $(2,1,4,3,5)$, $\mathbf{p3} = (\mathbf{1},\mathbf{4},\mathbf{2},\mathbf{3},\mathbf{5})$. *In this way, the local search procedure from VNS-LS1 procedure reached the solution $p3$, which was not reachable with the local search from GVNS.*

Comparing the local search procedures from GVNS and VNS-LS1, it is clear that VND can reach some better solutions by thorough exploration trough the neighborhoods $\mathcal{N}_2$,...,$\mathcal{N}_{n-1}$, which are not reachable within the local search from VNS-LS1. But, on the other hand, local search from VNS-LS1 is much faster.

3. GAUSS-VNS FOR FILE TRANSFER SCHEDULING PROBLEM

In the classic VNS-based heuristics the number of different neighborhoods of some incumbent solution is finite. As a consequence, one can reach only the points from these bounded neighborhoods, but not an arbitrary point from the solution space. In this way, one may not reach the region of attraction of the true global optimum. In order to be able to reach arbitrary point from the solution space, one may chose a very large number $k_{\max}$ and the largest neighborhood, but this approach is not efficient enough. In [3] authors presented a new variant of VNS, Gauss-VNS, which avoids this limitation. In Gauss-VNS, instead of defining a sequence of neighborhoods $N_1(x),\ldots,N_{k_{\max}}(x)$, a sequence of random point distributions for shaking step $\mathcal{P}_1(x),\ldots,\mathcal{P}_{k_{\max}}(x)$, are defined. For simplicity, they assumed that each $\mathcal{P}_k(x)$ is an n-variate Gaussian distribution centered at x. With such an approach, one can jump from an incumbent x to any trial point in the solution space.

In Gauss-VNS approach, since $\mathcal{P}_k(x)$ is a multivariate Gaussian distribution with mean x and covariance matrix Σ_k, a random values following this distribution can be easily obtained from n independent values $z_1, \ldots, z_n$ of a univariate Gaussian distribution with mean 0 and variance 1. Let $\Sigma_k = L_k L_k^\top$ be the Cholesky decomposition of the symmetric positive definite matrix Σ_k. The random vector $x + L_k z$, with $z = (z_1, \ldots, z_n)$ is distributed as $\mathcal{P}_k(x)$. If the covariance matrices Σ_k are chosen to be multiples of the identity matrix I, $\Sigma_k = \sigma_k^2 I, k = 1, 2, \ldots, k_{\max}$, the Cholesky decomposition is simply $\Sigma_k = (\sigma_k I)(\sigma_k I)^\top$.

In order to adapt Gauss-VNS for solving the discrete FTSP, the neighborhoods $N_k(x)$ will no longer exist. The only neighborhood used in the search process will be equal to the entire solution space. In the shaking step, a solution x', representing the priorities assigned to edges e_k, should be chosen using the Gaussian distribution (instead of previously used uniform distribution).

In continuous global optimization, the set of feasible solutions is uncountable, while in the case of the discrete FTSP this set is finite, so it is necessary to make a mapping from the uncountable to this finite set. For solving FTSP, the integer numbers were used in solution representation, which would be modified to accept real numbers in order to apply continuous Gauss-VNS. For creating the unique schedule of the file transfers, the priorities of the edges were compared, thus it is necessary to adapt the new solution representation. The vector with integer edge priorities will be replaced by the vector with continuous edge weights. Using this weight representation, the principle for creating the unique file transfer schedule remains the same: the edge with the larger weight have the priority over the edges with smaller weight. After obtaining the transfer schedule, all other calculations over this schedule do not use the weights, so no other adaptation is necessary.

Using this weight representation of the solution, it is clear that an uncountable number of solutions can be assigned to the same schedule. For example, weight vectors $x_1 = (2.8, 3.4, 1.1)$ and $x_2 = (2.8, 3.3, 1.1)$ have the same transfer schedule: the file represented by edge e_2 will start its transfer first, followed by files represented by edges e_1 and e_3, respectively.

The realization of mapping uncountable set of feasible solutions into a finite set makes this application of Gauss-VNS interesting. Although Gauss-VNS showed promising results for solving the problems of continuous optimization, it is natural to ask if after realization of this kind of mapping, its efficiency will be preserved. In other words, the possible outcome may be that, through iterations, weight vectors will change, but none of these changes would result in changing the transfer schedule.

Taking previous considerations and changes of VNS into account, Gauss-VNS algorithm for solving FTSP can be described as follows. The solution of FTSP is represented by a n-dimensional vector $x = (x_1, ..., x_n)$ with real values, where n is the number of edges in FTSP. Vector elements represent the weights assigned to each edge e_k in the graph.

The objective function value. For a given solution x, the objective function value is calculated in the similar way as in previous VNS implementation. All edges are sorted according to their weights. In the case of equal weight values,

the edge with larger length has priority, and in the case of equal length, the edge with smaller index has priority. Using this sorted array, the unique schedule s and the objective function value are obtained in the same way as previously.

The shaking procedure. In the shaking procedure, the new vector of edge weights x' will be chosen using the Gaussian distribution with mean x (where x is the incumbent solution), and a variance σ_k. It is necessary to define in advance these variances that will be used in this step. Parameter k, $k_{\min} \leq k \leq k_{\max}$ in this modification represents the index of the element from the set of predefined variances $\sigma_{k_{\min}}, ..., \sigma_{k_{\max}}$ to be used in current iteration. Since the parameter k is representing the index, $k_{\min} = 1$ is an obvious choice.

For the given k, in the shaking procedure Gauss-VNS generates a new solution x' according to the following steps:

- Form n independent values $z_1, \ldots, z_n$ using the Gaussian distribution with mean 0 and variance 1.
- Form the vector x' as $x' = x + z \cdot \sigma_k$.

For generating univariate Gaussian variables z_i, the well known Ahrens-Dieter algorithm [2] was used.

Local search procedure. After obtaining the solution x' in the shaking step, any of three previously defined local search algorithms can be applied. In that way, three new modifications of VNS can be defined:

- Gauss-VNS - The neighborhood to be explored for potentially better solution x'' consists of all vectors obtained from x' by swapping two of its elements.
- Gauss-GVNS - Using VND as the local search trough the neighborhoods $\mathcal{N}_l(x')$, $l = 2, ..., n-1$ described in the Subsection 2.1.
- Gauss-VNS-LS1 - Using the only one neighborhood $\mathcal{N}_1(x')$ for exploration in the local search procedure, as described in Subsection 2.2.

In all three modifications, the first improvement strategy was used. Making the decision whether the algorithm should move to a solution obtained by local search x'' or not is the same as before. The steps of VNS modifications using the Gaussian distribution are given on the pseudo-code on Figure 3.

In general, it is not easy to perform the discretization of the Gauss-VNS approach. The solution representation of FTSP from [5] was a good base since it was possible to change the values from the solution vector from integer to real numbers without changing other aspects of the algorithm. For the other problems of discrete optimization, for which a discrete VNS approach is already developed, it is not obvious if it can be adapted to the Gauss-VNS approach.

4. EXPERIMENTAL RESULTS

This section contains the experimental results obtained by all new VNS modifications for solving FTSP, presented in this paper. All computations were carried

```
/* Initialization */
Select the set of covariance matrices Σ_k, k = 1,...,k_max;
Randomly choose an arbitrary initial point x ∈ X and set
  x* ← x, f* ← f(x);
repeat the following steps until the stopping criterion is met
    Set k ← 1;
1:  repeat the following steps until k > k_max
        /* Shaking */
        Generate x' from a Gaussian distribution with
         mean x* and covariance matrix Σ_k;
        /* Local search */
        Apply some local search method with x' as initial solution
          to obtain a local minimum x'' of the problem;
        /* Move or not? */
        if f(x'') < f* then
            Set x* ← x'', f* ← f(x'') and goto 1;
        elseif f(x'') > f* then
            Set k ← k + 1;
        else
            With probability p set x* ← x'', f* ← f(x'') and goto 1,
              and with probability p − 1 set k ← k + 1;
    end
end
Stop. Point x* is an approximative solution of the problem.
```

Figure 3: The Gauss-VNS method for FTSP.

out on a single core of the Intel Core 2 Duo 2.67 GHz PC with 4 GB RAM under Windows XP operating system. The algorithm was coded in C programming language.

For experimental testing, the instances from [5] were used. The stopping criterion for all the algorithms was the maximal number of iterations $itermax = 100$ for small size (number of vertices $v_{\max} = 5, 10$ and number of edges $e_{\max} = 10$) and medium size ($v_{\max} = 10, 30$ and $e_{\max} = 50, 100$) instances. For large size instances ($v_{\max} = 30, 50, 100$ and $e_{\max} = 200, 500$), 100 iterations resulted in large CPU times, so for these instances the stopping criterion was set to CPU time limited to $t_{tot} = 1000s$. For GVNS and VNS_LS1 algorithms, the other parameter values were: $k_{\min} = 2, k_{\max} = 20, p = 0.4$. These are the same values used for testing VNS in [5]. The results are summarised in Table 1. On the other hand, for Gauss-VNS, Gauss-GVNS and Gauss-VNS-LS1 algorithms, the following parameter values were used: $k_{\min} = 1$ (since the parameter k represents the index of the element from the set of predefined variances), $k_{\max} = 10$ and the set of variances $\sigma =$ 0.01, 0.05, 0.10, 0.20, 0.50, 1.00, 2.00, 5.00, 10.00, 50.00. The results of these three modifications are presented in Table 2. Because of the stochastic nature of the VNS algorithm, each VNS modification was run 20 times for each instance.

Table 1 is organized as follows. In the first column, labeled as "Inst." , the test instance name is given. It contains the information about the number of vertices, the number of edges and the ordinal number of the instance of that type, respectively. In the second column, labeled as "LB", the elementary lower bound for that instance is given, calculated as described by (3). The third column, labeled

as "opt", contains the optimal solution value found by CPLEX and/or Gurobi on proposed ILP problem formulations from [6] and [7]. If CPLEX or Gurobi did not produce an optimal result, the symbol "-" is written. The next five columns contain the results and data obtained by VNS algorithm presented in [5]: the column labeled as "VNS_{best}" contains the best objective function value found by VNS algorithm in 20 runs; next two columns, t and t_{tot}, contain average execution time used to reach the final VNS solution and average total execution time given in seconds; column labeled as "err" contains an average relative percentage error of objective function solution values, defined as $err = \frac{1}{20}\sum_{i=1}^{20} err_i$ where $err_i = 100 * \frac{VNS_i - VNS_{best}}{VNS_{best}}$, and VNS_i represents the VNS result obtained in the i-th run, $i = 1, ..., 20$; column "sd" is standard deviation of err_i, $i = 1, ..., 20$ obtained by formula $sd = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(err_i - err)^2}$. The next five columns contain the data obtained by the GVNS algorithm presented in the same way as for the VNS algorithm. In the last five columns the results obtained by VNS-LS1 algorithm are presented.

Table 2 is organized in the similar way as Table 1, containing the results obtained by Gauss-VNS, Gauss-GVNS and Gauss-VNS-LS1 algorithms.

In the columns containing the best objective function value, if the obtained result is equal to the proved optimal result from the third column, it is marked as "opt". If the the optimal result of the problem is not known and the obtained result is equal to LB, it has to be the optimal result of FTSP. In this case, the obtained result is marked as "LB opt".

As it can be seen from Tables 1 and 2, for small size instances, all six modifications reached all optimal solutions in each of 20 runs ($err = 0$ and $sd = 0$) very fast.

For medium size instances, all six modifications also reached all optimal solutions, but the best average performance had GVNS and Gauss-GVNS. GVNS and Gauss-GVNS gave the best average results since both algorithms reached all optimal solutions in all 20 runs for all instances. In total, VNS performed as the worst algorithm, obtaining the same result in 20 runs for 17 out of 20 instances. Gauss-VNS, VNS-LS1 and Gauss-VNS-LS1 had better average results than VNS since they reached all optimal solutions in all runs for 19 out of 20 instances. Comparing the total execution times, GVNS and Gauss-GVNS are insignificantly slower than VNS and Gauss-VNS, and VNS-LS1 and Gauss-VNS-LS1 are the fastest.

For large size instances, all five new modifications performed better than VNS. VNS reached 7 optimal solutions, from which the optimality for three instances follows from its equality to the elementary lower bound. On the other hand, all five other modifications reached 9 optimal solutions: four of them are equal to the proved optimal solution from the third column, and other five have values equal to LB. For other six instances, for which the optimal solutions are not known, all five modifications reached better solutions than VNS in two cases ($ftsp\_30\_500\_3$ and $ftsp\_100\_500\_2$). For the instance $ftsp\_30\_500\_3$, Gauss-VNS obtained better solution ($Gauss - VNS_{best} = 128$) than VNS ($VNS_{best} = 135$), but the smallest value was obtained by other four modifications ($GVNS_{best} = VNS - LS1_{best} =$

Table 1: VNS, GVNS and VNS-LS1 results

Inst	LB	opt	VNS_{best}	t	t_{tot}	err	sd	$GVNS_{best}$	t	t_{tot}	err	sd	$VNS\text{-}LS1_{best}$	t	t_{tot}	err	*sd*
Small size instances																	
ftsp_5_10_0	14	14	opt	0.018	0.021	0.000	0.000	opt	0.025	0.032	0.000	0.000	opt	0.015	0.019	0.000	0.000
ftsp_5_10_1	10	15	opt	0.020	0.021	0.000	0.000	opt	0.022	0.025	0.000	0.000	opt	0.009	0.011	0.000	0.000
ftsp_5_10_2	12	12	opt	0.015	0.017	0.000	0.000	opt	0.015	0.017	0.000	0.000	opt	0.007	0.009	0.000	0.000
ftsp_5_10_3	14	14	opt	0.018	0.020	0.000	0.000	opt	0.018	0.019	0.000	0.000	opt	0.008	0.010	0.000	0.000
ftsp_5_10_4	10	10	opt	0.012	0.014	0.000	0.000	opt	0.014	0.015	0.000	0.000	opt	0.009	0.011	0.000	0.000
ftsp_5_10_5	6	13	opt	0.014	0.018	0.000	0.000	opt	0.018	0.020	0.000	0.000	opt	0.014	0.015	0.000	0.000
ftsp_5_10_6	12	13	opt	0.017	0.020	0.000	0.000	opt	0.021	0.022	0.000	0.000	opt	0.009	0.012	0.000	0.000
ftsp_5_10_7	8	8	opt	0.018	0.019	0.000	0.000	opt	0.013	0.014	0.000	0.000	opt	0.011	0.012	0.000	0.000
ftsp_5_10_8	11	11	opt	0.013	0.014	0.000	0.000	opt	0.015	0.020	0.000	0.000	opt	0.007	0.009	0.000	0.000
ftsp_5_10_9	26	26	opt	0.019	0.020	0.000	0.000	opt	0.025	0.028	0.000	0.000	opt	0.009	0.011	0.000	0.000
ftsp_10_10_0	12	15	opt	0.013	0.019	0.000	0.000	opt	0.011	0.012	0.000	0.000	opt	0.007	0.008	0.000	0.000
ftsp_10_10_1	12	12	opt	0.007	0.009	0.000	0.000	opt	0.014	0.014	0.000	0.000	opt	0.005	0.007	0.000	0.000
ftsp_10_10_2	5	14	opt	0.010	0.012	0.000	0.000	opt	0.012	0.012	0.000	0.000	opt	0.007	0.008	0.000	0.000
ftsp_10_10_3	19	19	opt	0.019	0.020	0.000	0.000	opt	0.020	0.021	0.000	0.000	opt	0.011	0.012	0.000	0.000
ftsp_10_10_4	17	17	opt	0.013	0.015	0.000	0.000	opt	0.014	0.016	0.000	0.000	opt	0.007	0.009	0.000	0.000
ftsp_10_10_5	10	15	opt	0.013	0.014	0.000	0.000	opt	0.014	0.017	0.000	0.000	opt	0.008	0.008	0.000	0.000
ftsp_10_10_6	3	15	opt	0.011	0.013	0.000	0.000	opt	0.012	0.013	0.000	0.000	opt	0.007	0.009	0.000	0.000
ftsp_10_10_7	11	14	opt	0.011	0.012	0.000	0.000	opt	0.012	0.014	0.000	0.000	opt	0.009	0.009	0.000	0.000
ftsp_10_10_8	11	14	opt	0.012	0.013	0.000	0.000	opt	0.011	0.014	0.000	0.000	opt	0.007	0.007	0.000	0.000
ftsp_10_10_9	2	14	opt	0.010	0.011	0.000	0.000	opt	0.010	0.011	0.000	0.000	opt	0.005	0.007	0.000	0.000
Medium size instances																	
ftsp_10_50_0	26	92	opt	5.681	5.775	0.000	0.000	opt	5.900	5.999	0.000	0.000	opt	0.223	0.226	0.000	0.000
ftsp_10_50_1	17	81	opt	4.733	4.811	0.000	0.000	opt	4.884	4.964	0.000	0.000	opt	0.186	0.189	0.000	0.000
ftsp_10_50_2	40	42	opt	3.371	3.532	0.000	0.000	opt	3.695	3.842	0.000	0.000	opt	0.178	0.185	0.000	0.000
ftsp_10_50_3	31	57	opt	4.147	4.212	0.000	0.000	opt	4.343	4.413	0.000	0.000	opt	0.163	0.167	0.000	0.000
ftsp_10_50_4	17	20	opt	1.903	2.025	0.000	0.000	opt	2.259	2.314	0.000	0.000	opt	0.107	0.111	0.000	0.000
ftsp_10_50_5	32	32	opt	2.493	2.725	1.406	1.817	opt	3.125	3.216	0.000	0.000	opt	0.141	0.158	0.469	1.111
ftsp_10_50_6	36	36	opt	2.962	3.328	0.000	0.000	opt	3.583	3.702	0.000	0.000	opt	0.168	0.179	0.000	0.000
ftsp_10_50_7	85	85	opt	6.015	6.114	0.000	0.000	opt	6.406	6.511	0.000	0.000	opt	0.238	0.244	0.000	0.000
ftsp_10_50_8	85	85	opt	5.141	5.225	0.000	0.000	opt	5.274	5.361	0.000	0.000	opt	0.203	0.208	0.000	0.000
ftsp_10_50_9	44	44	opt	3.206	3.525	0.000	0.000	opt	3.791	3.878	0.000	0.000	opt	0.181	0.189	0.000	0.000
ftsp_30_100_0	199	199	opt	72.225	73.568	0.000	0.000	opt	86.978	88.528	0.000	0.000	opt	1.497	1.527	0.000	0.000
ftsp_30_100_1	45	68	opt	26.481	27.229	0.000	0.000	opt	29.147	29.497	0.000	0.000	opt	0.569	0.582	0.000	0.000
ftsp_30_100_2	29	108	opt	42.427	44.350	0.093	0.278	opt	46.443	47.143	0.000	0.000	opt	1.142	1.168	0.000	0.000
ftsp_30_100_3	108	108	opt	50.615	52.523	0.000	0.000	opt	54.489	56.208	0.000	0.000	opt	1.364	1.393	0.000	0.000
ftsp_30_100_4	31	67	opt	26.903	27.724	0.970	4.188	opt	33.880	34.293	0.000	0.000	opt	0.648	0.658	0.000	0.000
ftsp_30_100_5	163	163	opt	59.277	60.400	0.000	0.000	opt	78.232	79.685	0.000	0.000	opt	1.215	1.239	0.000	0.000
ftsp_30_100_6	30	50	opt	7.918	8.070	0.000	0.000	opt	11.180	11.401	0.000	0.000	opt	0.176	0.178	0.000	0.000
ftsp_30_100_7	177	177	opt	71.007	72.350	0.000	0.000	opt	87.588	89.286	0.000	0.000	opt	1.471	1.499	0.000	0.000
ftsp_30_100_8	45	96	opt	42.533	43.724	0.000	0.000	opt	45.654	46.357	0.000	0.000	opt	0.901	0.919	0.000	0.000
ftsp_30_100_9	270	270	opt	98.240	100.092	0.000	0.000	opt	99.061	100.980	0.000	0.000	opt	1.945	1.982	0.000	0.000
Large size instances																	
ftsp_30_500_0	167	-	171	887.605	1000	5.393	2.716	*LBopt*	825.678	1000	0.180	0.274	*LBopt*	997.955	1000	0.000	0.000
ftsp_30_500_1	718	-	*LBopt*	991.395	1000	0.000	0.000	*LBopt*	713.109	1000	0.000	0.000	*LBopt*	996.824	1000	0.000	0.000
ftsp_30_500_2	269	-	*LBopt*	795.315	1000	2.034	1.732	*LBopt*	833.329	1000	0.000	0.000	*LBopt*	999.611	1000	0.000	0.000
ftsp_30_500_3	117	-	135	886.661	1000	4.327	3.738	127	879.243	1000	0.630	0.878	127	997.990	1000	0.278	0.377
ftsp_30_500_4	528	-	*LBopt*	955.888	1000	0.000	0.000	*LBopt*	870.958	1000	0.000	0.000	*LBopt*	999.077	1000	0.000	0.000
ftsp_50_200_0	73	-	567	981.955	1000	0.000	0.000	567	979.895	1000	0.000	0.000	567	999.901	1000	0.000	0.000
ftsp_50_200_1	89	116	opt	986.646	1000	0.000	0.000	opt	994.576	1000	0.000	0.000	opt	999.952	1000	0.000	0.000
ftsp_50_200_2	423	423	opt	985.672	1000	0.000	0.000	opt	990.852	1000	0.000	0.000	opt	999.875	1000	0.000	0.000
ftsp_50_200_3	102	109	opt	988.453	1000	3.165	2.150	opt	996.405	1000	0.000	0.000	opt	999.916	1000	0.000	0.000
ftsp_50_200_4	59	344	opt	986.380	1000	0.000	0.000	opt	987.642	1000	0.000	0.000	opt	999.951	1000	0.000	0.000
ftsp_100_500_0	208	-	578	941.905	1000	0.000	0.000	578	807.497	1000	0.000	0.000	578	999.816	1000	0.000	0.000
ftsp_100_500_1	197	-	237	940.841	1000	9.008	4.292	224	943.347	1000	3.482	3.080	224	999.669	1000	0.000	0.000
ftsp_100_500_2	300	-	306	943.365	1000	10.327	5.433	*LBopt*	920.666	1000	0.217	0.190	*LBopt*	998.593	1000	0.000	0.000
ftsp_100_500_3	794	-	830	990.698	1000	0.000	0.000	830	719.325	1000	0.000	0.000	830	998.847	1000	0.000	0.000
ftsp_100_500_4	154	-	224	999.698	1000	11.964	9.504	224	871.875	1000	0.000	0.000	224	999.241	1000	0.000	0.000

Table 2: Gauss-VNS, Gauss-GVNS and Gauss-VNS-LS1 results

Inst	LB	opt	Gauss-VNS$_{best}$	t	t$_{tot}$	err	sd	Gauss-GVNS$_{best}$	t	t$_{tot}$	err	sd	Gauss-VNS-LS1$_{best}$	t	t$_{tot}$	err	*sd*
Small size instances																	
ftsp_5_10_0	14	14	opt	0.015	0.017	0.000	0.000	opt	0.016	0.017	0.000	0.000	opt	0.006	0.007	0.000	0.000
ftsp_5_10_1	10	15	opt	0.015	0.015	0.000	0.000	opt	0.016	0.017	0.000	0.000	opt	0.007	0.007	0.000	0.000
ftsp_5_10_2	12	12	opt	0.011	0.014	0.000	0.000	opt	0.014	0.015	0.000	0.000	opt	0.002	0.006	0.000	0.000
ftsp_5_10_3	14	14	opt	0.015	0.016	0.000	0.000	opt	0.015	0.017	0.000	0.000	opt	0.007	0.007	0.000	0.000
ftsp_5_10_4	10	10	opt	0.011	0.012	0.000	0.000	opt	0.012	0.012	0.000	0.000	opt	0.007	0.009	0.000	0.000
ftsp_5_10_5	6	13	opt	0.014	0.015	0.000	0.000	opt	0.014	0.014	0.000	0.000	opt	0.002	0.007	0.000	0.000
ftsp_5_10_6	12	13	opt	0.014	0.015	0.000	0.000	opt	0.013	0.017	0.000	0.000	opt	0.005	0.007	0.000	0.000
ftsp_5_10_7	8	8	opt	0.010	0.012	0.000	0.000	opt	0.009	0.012	0.000	0.000	opt	0.002	0.004	0.000	0.000
ftsp_5_10_8	11	11	opt	0.013	0.014	0.000	0.000	opt	0.012	0.014	0.000	0.000	opt	0.004	0.006	0.000	0.000
ftsp_5_10_9	26	26	opt	0.021	0.021	0.000	0.000	opt	0.017	0.021	0.000	0.000	opt	0.006	0.007	0.000	0.000
ftsp_10_10_0	12	15	opt	0.007	0.008	0.000	0.000	opt	0.008	0.009	0.000	0.000	opt	0.004	0.004	0.000	0.000
ftsp_10_10_1	12	12	opt	0.009	0.009	0.000	0.000	opt	0.007	0.008	0.000	0.000	opt	0.003	0.004	0.000	0.000
ftsp_10_10_2	5	14	opt	0.006	0.009	0.000	0.000	opt	0.007	0.007	0.000	0.000	opt	0.004	0.004	0.000	0.000
ftsp_10_10_3	19	19	opt	0.009	0.015	0.000	0.000	opt	0.014	0.015	0.000	0.000	opt	0.005	0.006	0.000	0.000
ftsp_10_10_4	17	17	opt	0.012	0.013	0.000	0.000	opt	0.012	0.013	0.000	0.000	opt	0.004	0.007	0.000	0.000
ftsp_10_10_5	10	15	opt	0.009	0.011	0.000	0.000	opt	0.010	0.012	0.000	0.000	opt	0.003	0.005	0.000	0.000
ftsp_10_10_6	3	15	opt	0.007	0.008	0.000	0.000	opt	0.008	0.008	0.000	0.000	opt	0.004	0.005	0.000	0.000
ftsp_10_10_7	11	14	opt	0.012	0.013	0.000	0.000	opt	0.010	0.012	0.000	0.000	opt	0.004	0.007	0.000	0.000
ftsp_10_10_8	11	14	opt	0.011	0.013	0.000	0.000	opt	0.009	0.011	0.000	0.000	opt	0.005	0.005	0.000	0.000
ftsp_10_10_9	2	14	opt	0.008	0.009	0.000	0.000	opt	0.008	0.009	0.000	0.000	opt	0.004	0.004	0.000	0.000
Medium size instances																	
ftsp_10_50_0	26	92	opt	5.896	5.975	0.000	0.000	opt	5.619	5.697	0.000	0.000	opt	0.227	0.230	0.000	0.000
ftsp_10_50_1	17	81	opt	4.831	4.899	0.000	0.000	opt	4.626	4.687	0.000	0.000	opt	0.201	0.205	0.000	0.000
ftsp_10_50_2	40	42	opt	3.428	3.760	0.000	0.000	opt	3.342	3.550	0.000	0.000	opt	0.152	0.185	0.000	0.000
ftsp_10_50_3	31	57	opt	4.257	4.314	0.000	0.000	opt	4.078	4.136	0.000	0.000	opt	0.181	0.184	0.000	0.000
ftsp_10_50_4	17	20	opt	2.107	2.182	0.000	0.000	opt	2.024	2.072	0.000	0.000	opt	0.098	0.106	0.000	0.000
ftsp_10_50_5	32	32	opt	2.213	3.113	0.313	1.358	opt	2.773	2.842	0.000	0.000	opt	0.096	0.194	1.875	1.503
ftsp_10_50_6	36	36	opt	3.486	3.582	0.000	0.000	opt	3.229	3.404	0.000	0.000	opt	0.155	0.182	0.000	0.000
ftsp_10_50_7	85	85	opt	6.236	6.321	0.000	0.000	opt	6.021	6.103	0.000	0.000	opt	0.264	0.268	0.000	0.000
ftsp_10_50_8	85	85	opt	5.060	5.132	0.000	0.000	opt	4.896	4.965	0.000	0.000	opt	0.203	0.207	0.000	0.000
ftsp_10_50_9	44	44	opt	3.600	3.684	0.000	0.000	opt	3.361	3.447	0.000	0.000	opt	0.150	0.181	0.000	0.000
ftsp_30_100_0	199	199	opt	78.051	79.682	0.000	0.000	opt	73.495	75.041	0.000	0.000	opt	1.525	1.562	0.000	0.000
ftsp_30_100_1	45	68	opt	29.585	30.211	0.000	0.000	opt	27.237	27.820	0.000	0.000	opt	0.583	0.603	0.000	0.000
ftsp_30_100_2	29	108	opt	55.522	56.661	0.000	0.000	opt	43.768	44.649	0.000	0.000	opt	1.208	1.250	0.000	0.000
ftsp_30_100_3	108	108	opt	50.053	54.346	0.000	0.000	opt	46.230	52.853	0.000	0.000	opt	1.278	1.348	0.000	0.000
ftsp_30_100_4	31	67	opt	29.157	29.809	0.000	0.000	opt	28.703	29.344	0.000	0.000	opt	0.656	0.695	0.000	0.000
ftsp_30_100_5	163	163	opt	64.701	66.052	0.000	0.000	opt	62.379	63.645	0.000	0.000	opt	1.301	1.329	0.000	0.000
ftsp_30_100_6	30	50	opt	8.739	8.919	0.000	0.000	opt	8.331	8.501	0.000	0.000	opt	0.172	0.173	0.000	0.000
ftsp_30_100_7	177	177	opt	76.274	77.841	0.000	0.000	opt	70.811	72.246	0.000	0.000	opt	1.488	1.522	0.000	0.000
ftsp_30_100_8	45	96	opt	47.032	48.008	0.000	0.000	opt	40.986	41.850	0.000	0.000	opt	0.836	0.914	0.000	0.000
ftsp_30_100_9	270	270	opt	98.661	100.680	0.000	0.000	opt	92.165	94.019	0.000	0.000	opt	1.939	1.978	0.000	0.000
Large size instances																	
ftsp_30_500_0	167	-	*LBopt*	942.840	1000	1.198	1.183	*LBopt*	999.650	1000	1.078	1.409	*LBopt*	646.215	1000	0.000	0.000
ftsp_30_500_1	718	-	*LBopt*	821.341	1000	0.000	0.000	*LBopt*	678.870	1000	0.000	0.000	*LBopt*	999.022	1000	0.000	0.000
ftsp_30_500_2	269	-	*LBopt*	912.350	1000	0.372	0.483	*LBopt*	937.986	1000	0.446	0.904	*LBopt*	995.326	1000	0.000	0.000
ftsp_30_500_3	117	-	128	864.230	1000	2.500	2.723	127	999.649	1000	1.969	1.629	127	398.197	1000	0.236	0.360
ftsp_30_500_4	528	-	*LBopt*	767.504	1000	0.000	0.000	*LBopt*	999.205	1000	0.000	0.000	*LBopt*	999.607	1000	0.000	0.000
ftsp_50_200_0	73	-	567	999.601	1000	0.000	0.000	567	998.030	1000	0.000	0.000	567	999.866	1000	0.000	0.000
ftsp_50_200_1	89	116	opt	995.756	1000	0.000	0.000	opt	997.844	1000	0.000	0.000	opt	999.978	1000	0.000	0.000
ftsp_50_200_2	423	423	opt	990.523	1000	0.000	0.000	opt	995.582	1000	0.000	0.000	opt	999.873	1000	0.000	0.000
ftsp_50_200_3	102	109	opt	999.270	1000	0.000	0.000	opt	996.374	1000	0.000	0.000	opt	820.537	1000	0.000	0.000
ftsp_50_200_4	59	344	opt	994.847	1000	0.000	0.000	opt	988.919	1000	0.000	0.000	opt	999.872	1000	0.000	0.000
ftsp_100_500_0	208	-	578	982.725	1000	0.000	0.000	578	942.355	1000	0.000	0.000	578	999.449	1000	0.000	0.000
ftsp_100_500_1	197	-	224	999.619	1000	2.946	2.424	224	970.648	1000	4.107	2.954	224	908.768	1000	0.000	0.000
ftsp_100_500_2	300	-	*LBopt*	890.219	1000	0.350	0.267	*LBopt*	920.023	1000	0.283	0.217	*LBopt*	605.165	1000	0.000	0.000
ftsp_100_500_3	794	-	830	871.945	1000	0.000	0.000	830	734.119	1000	0.000	0.000	830	998.601	1000	0.000	0.000
ftsp_100_500_4	154	-	224	962.870	1000	0.000	0.000	224	927.883	1000	0.000	0.000	224	999.343	1000	0.000	0.000

$Gauss - GVNS_{best} = Gauss - VNS - LS1_{best} = 127$). Comparing the quality of obtained results, Gauss-VNS was better than VNS with values *err* and *sd* equal to zero for 10 out of 15 instances, while for VNS this was obtained for 8 out of 15 instances. GVNS reached the same solution in 20 runs for 11 out of 15 instances, and Gauss-GVNS for 10 out of 15 instances. VNS-LS1 and Gauss-VNS-LS1 were the best, obtaining the same solution in 20 runs for 14 out of 15 instances.

Since the algorithm pairs VNS and Gauss-VNS, GVNS and Gauss-GVNS, VNS-LS1 and Gauss-VNS-LS1 have the same local search procedure, it is interesting to compare the results obtained by each pair. As it can be seen from discussion above, Gauss-VNS performed better than VNS, obtaining better results for the large size instances, and better average performance for medium and large size instances. On the other hand, GVNS, Gauss-GVNS, VNS-LS1 and Gauss-VNS-LS1 obtained the same or similar results, all better than VNS and Gauss-VNS.

5. CONCLUSION

In this paper new modifications of VNS approach for solving the discrete File transfer scheduling problem were presented. In order to obtain better solutions in a small neighborhood of a current solution, we implemented two new local search procedures: Variable Neighborhood Descent, and swapping two adjacent elements. In order to apply the continuous optimization method (Gaussian Variable Neighborhood Search) to solve FTSP, mapping of continuous set of feasible solutions into a finite set is performed, and the weight solution representation is developed. Such representation enables the continuous optimization methods to be used, which do not require the differentiability of objective function. Since Gauss-VNS method is proved to be successful in continuous optimization problems, it was applied to discrete FTSP. Previously described local search procedures could also be used with weight solution representation. Numerical experiments showed that the results obtained by the Gaussian modifications are comparable with the results obtained by standard VNS based heuristics developed for combinatorial optimization. In some cases the Gaussian modifications gave even better results.

Future research may contain automatic estimation of range of parameter σ for Gauss-VNS, as well as the usage of Gaussian distribution for some other discrete optimization problems.

Acknowledgement: This research was supported by Serbian Ministry of Education, Science and Technological Development under the grants No. 174010.

REFERENCES

[1] Akbari M.K., Nezhad M.H, and Kalantari M., "Neural Network Realization of File Transfer Scheduling", *The CSI Journal of Computer Science and Engineering*, 2(2&4) (2004) 19-29.

[2] Ahrens, J. H., and Dieter, U., "Efficient table-free sampling methods for the exponential, Cauchy, and normal distributions", *Communications of the ACM*, 31(11) (1988) 1330-1337.

[3] Carrizosa, E., Dražić, M., Dražić, Z., and Mladenović, N., "Gaussian variable neighborhood search for continuous optimization", *Computers & Operations Research*, 39(9) (2012) 2206-2213.

[4] Coffman, Jr, E. G., Garey, M. R., Johnson, D. S., and LaPaugh, A. S., "Scheduling file transfers", *SIAM Journal on Computing*, 14(3) (1985) 744-780.
[5] Dražić, Z., "Variable Neighborhood Search for the File Transfer Scheduling Problem", *Serdica Journal of Computing*, 6(3) (2012) 333-348.
[6] Dražić, Z., Savić, A., and Filipović, V., "An integer linear formulation for the file transfer scheduling problem" *TOP*, 22(3) (2014) 1062-1073.
[7] Dražić, Z., "Modifications of the Variable neighborhood search method and their applications to solving the File transfer scheduling problem" (in Serbian), *Ph.D. thesis, University of Belgrade, Faculty of Mathematics*, (2014).
[8] Hansen, P., and Mladenović, N., "Variable neighborhood search: Principles and applications", *European journal of operational research*, 130(3) (2001) 449-467.
[9] Higuero, D., Tirado, J.M., Isaila, F., and Carretero, J., "Enhancing file transfer scheduling and server utilization in data distribution infrastructures" *In Proceedings of MASCOTS 2012 : The 20th IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, (2012) 431-438
[10] Mao, W., and Simha, R., "Routing and scheduling file transfers in packetswitched networks", *Journal of Computing and Information* 1(1) (1994) 559-574.
[11] Mladenović, N., and Hansen, P., "Variable neighborhood search", *Computers & Operations Research*, 24(11) (1997) 1097-1100.
[12] Mladenović, N., Todosijević, R., and Urošević, D., "An efficient general variable neighborhood search for large travelling salesman problem with time windows", *Yugoslav Journal of Operations Research*, 22 (2012) 141-151.
[13] Mladenović, N., Kratica, J., Kovačević-Vujčić, V., and Čangalović, M., "Variable neighborhood search for metric dimension and minimal doubly resolving set problems", *European Journal of Operational Research*, 220(2) (2012) 328-337.
[14] Urošević, D., "Variable neighborhood search for maximum diverse grouping problem", *Yugoslav Journal of Operations Research*, 24 (2014) 21-33.
[15] Veeraraghavan, M., Zheng, X., Feng, W. C., Lee, H., Chong, E. K., and Li, H., "Scheduling and transport for file transfers on high-speed optical circuits", *Journal of Grid Computing*, 1(4) (2003) 395-405.
[16] Wang, J., and Qiaoling M., "Some Results on Edge Cover Coloring of Double Graphs", *Applied Mathematics*, 3 (2012) 264-266.