Serdica J. Computing **6** (2012), 333–348

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

VARIABLE NEIGHBORHOOD SEARCH FOR THE FILE TRANSFER SCHEDULING PROBLEM

Zorica Dražić

Abstract. In this paper a file transfer scheduling problem is considered. This problem is known to be NP-hard, and thus provides a challenging area for metaheuristics. A variable neighborhood search algorithm is designed for the transfer scheduling of files between various nodes of a network, by which the overall transfer times are to be minimized. Optimality of VNS solutions on smaller size instances has been verified by total enumeration. For several larger instances optimality follows from reaching the elementary lower bound of a problem.

1. Introduction. The problem discussed in this paper is introduced by E.G. Coffman et al. in [3]. File transfer scheduling is concerned with transferring a number of large files between various nodes of a computer network. At initial time, files are located at some nodes and they should be transferred to some other nodes of a network. For each file the amount of time needed for its transfer

ACM Computing Classification System (1998): I.2.8, G.1.6.

Key words: Metaheuristics, scheduling, file transfers, variable neighborhood search, optimization.

is known. The files are transferred directly between their starting and ending nodes, without any forwarding involving other nodes. When the file transfer starts, it continues without any interruption until its completion. Every node in the network has its port constraint, i. e., the upper limit of simultaneous number of file transfers (sum of uploads and downloads) it can handle. The problem of minimizing the makespan of the schedule (the time interval between the beginning of the first transfer and the completion of the last transfer) is investigated in this paper. Node constraints make this problem hard to solve.

The file transfer scheduling problem (FTSP) can be modeled by an undirected multigraph $G = (V, E)$, which is called the file transfer graph. The vertices $v \in V$ represent the nodes of a network labeled with integers $p(v)$ representing its port constraint. The edges $e \in E$ represent the files to be transferred labeled with integers $L(e)$ representing the transferring time expressed in some time units. The set of starting file transfer time moments forms the schedule for the problem.

Minimizing the file transfer time is a common and very important problem in many areas such as Wide Area computer Networks (WAN), Local Area Networks (LAN), telecommunications, as well as in multiprocessor scheduling in a MIMD (multiple instruction multiple data) machine, task assignment in a company, multi-tasking operating systems, etc.

In [3] the authors proved that the FTSP is NP-complete. They have suggested several algorithms in which they have tried to compute the minimum makespan in polynomial time, but the solutions were restricted by some conditions: the file transfer graph G is bipartite and all the files have equal lengths; the graph G possesses no simple cycle except in the trivial case of 2-vertex cycle included by multiple edges and same value lengths; the graph G is a multiple-edged graph with even cycle and equal file lengths; the graph G is a multiple-edged graph with odd cycle, equal lengths of files, and all port capacities equal to one.

In [1] the authors presented the 2-dimensional neural network architecture for solving the FTSP. The experiments were carried out on instances with small dimensions—up to 39 vertices and 100 files that should be transferred. For all instances where the solution was found, the theoretical lower bound of objective function was reached which indicates that all test instances were too easy. The simulation results showed that for bigger instances the algorithm didn't converge to the optimum solution.

In [9] the authors extended this model to directed file transfer graphs and presented a polynomial-time algorithm for the problem when all file transfers require an equal amount of time. Also, they studied the case when the file

transfer time is arbitrary and proved that this problem is NP-complete.

In [5] the authors studied the performance of a greedy on-line algorithm for a similar problem in which file transfer requests become known to the algorithm one at a time and the routing decision must be made for each request before any future requests are known. They discussed the performance of several greedy on-line algorithms and showed that they are not always the best available.

In [6] the authors reconsidered time-index formulation of the FTSP in multi-server and multi-user environments. Thy reduced the complexity of the optimization by transforming it into an approximation problem.

2. Definition of the problem. This section has been written with extensive reference to the work of E. G. Coffman et al. [3]. Given a file transfer graph $G = (V, E)$, a schedule can be formally represented as a function $s : E \to [0, \infty)$ that assigns a starting time $s(e)$ to each edge $e \in E$, such that for each vertex $v \in V$ and time $t \geq 0$

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

The makespan length of a schedule s is defined as the largest finishing time, i.e., the maximum of $s(e) + L(e)$ over all edges $e \in E$. Figure 1 illustrates the file transfer graph and one of feasible timing diagrams which represent the schedule. Given a file transfer graph G, the goal is to find a schedule s with the minimum possible makespan.

One lower bound for the makespan, which shall be called the elementary lower bound, is obtained as follows. Let E_u denote the set of files that are to be sent or received by vertex u (i. e., the set of all edges that have one endpoint at the vertex u). The degree of the vertex u is given by $d_u = |E_u|$. Let $E_{u,v}$ denote $E_u \bigcap E_v$, i. e., the set of files to be transferred between u and v. Further, let $\sum_u = \sum_{e \in E_u} L(e)$ and $\sum_{u,v} = \sum_{e \in E_{u,v}} L(e)$. For consistency with this notation, p_u will be used to denote the port constraint $p(u)$ for each vertex u. The time to transmit all files sent or received by vertex u is at least $\lceil \sum_u / p_u \rceil$ ($\lceil x \rceil$ denoting the smallest integer greater than or equal to x). Thus, the following holds true: The optimal schedule length OPT(G) for any graph G must satisfy $OPT(G) \geq max_u \lceil \sum_u / p_u \rceil$. The right-hand side of this inequality is called the elementary lower bound for the considering FTSP.

Figure 2 illustrates the fact that OPT(G) can be substantially larger than $\max_{u \in V} \lceil \sum_u / p_u \rceil$. Here the value of a lower bound is $\max\{2, 2, 2, \frac{6}{2}\} = 3$, but the following simple analysis shows that a makespan of 3 is not achievable. In order

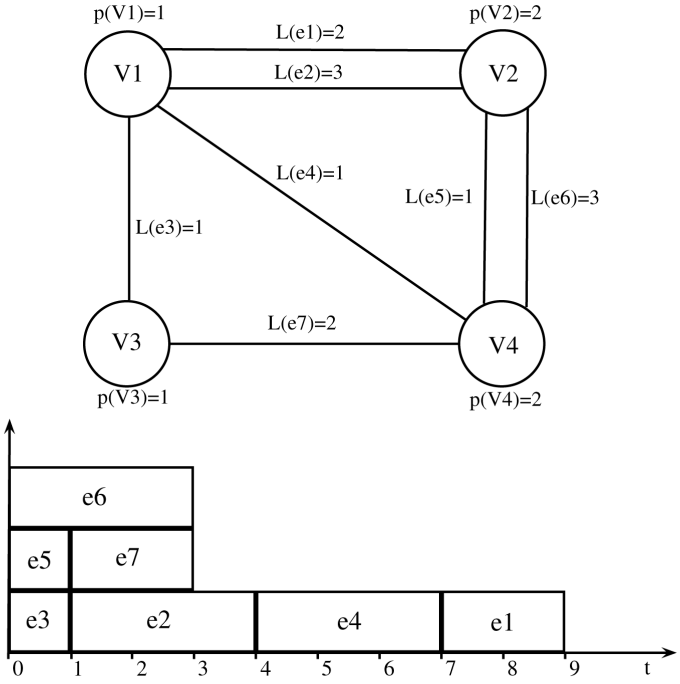

Fig. 1. A file transfer graph and a schedule

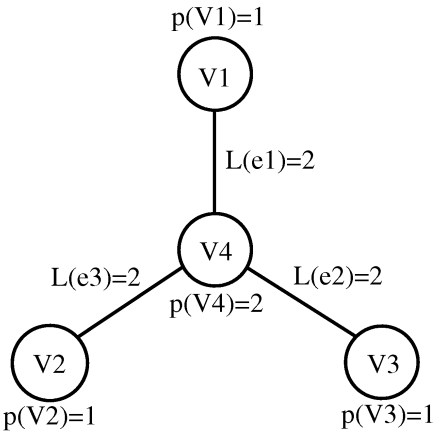

Fig. 2. An example where $OPT(G) \geq \max_u \lceil \sum_u / p_u \rceil$

to finish the transfer of three files of vertex v_4 in 3 time units, this two-port vertex has to transmit two of its three files simultaneously. During this time the third transfer cannot be in transmission. Since their transfer requires 2 time units, the third transfer can start only when the other two finish. Thus, it is easy to see that $OPT(G) = 4 > 3$.

Also note that this lower bound is not achieved by the schedule in Figure 1, where the length of the makespan is equal to 9, but $\lceil \sum_{v_1} / p_{v_1} \rceil = 7/1 = 7$.

It turns out that finding OPT(G) is not an easy problem to solve. The general decision problem "Given G and a bound B, is there a schedule s for G with makespan B or less" is NP-complete and hence unlikely to be solved efficiently by methods known so far. For a discussion of NP-completeness see [3].

3. Variable neighborhood search for FTSP. Variable neighborhood search (VNS) is an effective metaheuristic introduced by Mladenović and Hansen [10] in 1997. The basic idea of this method is to proceed to a systematic change of neighborhoods in order to avoid the algorithm traps in local optima. The algorithm does not follow a trajectory but explores an increasingly distant neighborhood of the current solution and jumps from the current solution to another only if the new solution is better. Local search and shaking functions perform this exploration, systematically.

At the starting point, a suitable neighborhood structure must be defined. Let $N^t(t = k_{\min}, \ldots, k_{\max})$ be a finite set of neighborhoods, where $N^k(x)$ is the set of solutions in the kth neighborhood of the solution x. The simplest and most common choice is a neighborhood structure in which the neighborhoods have increasing cardinality: $|N^{k_{\min}}(x)| < |N^{k_{\min}+1}(x)| < \ldots < |N^{k_{\max}}(x)|$.

At each step, VNS starts from the incumbent solution x and an integer $k \in \{k_{\min}, \ldots, k_{\max}\}$ associated to a current neighborhood. In the shaking step a random solution x' is generated from the $N^k(x)$. Starting from x', a local search is then performed to produce a possibly better solution x''. After exploring the local area, the best solution of the local search x'' is compared with x. If x'' is better than x, it replaces x and the next search continues with the first neighborhood $N^{k_{\min}}$ of x''. Otherwise, x is not replaced and algorithm continues from the shaking phase with the next neighborhood structure.

Whenever $k_{\max}$ is attained, the search continues with the first neighborhood $N^{k_{\min}}$. This is repeated until some stopping criterion is met. Possible stopping conditions can be a maximum CPU time allowed, maximum number of iterations or maximum number of iterations between two improvements. In its most popular version, the VNS algorithm takes the following form:

/* *Initialization* */
Select a set of neighborhood structures N_k, $k = k_{\min}, \ldots, k_{\max}$
that will be used in the search;
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
1: Set $k \leftarrow k_{\min}$;
repeat the following steps **until** $k > k_{\max}$
/* *Shaking* */
Generate at random a point $x' \in N_k(x^*)$;
/* *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 $x^* \leftarrow x''$, $f^* \leftarrow f(x'')$ and **goto** 1;
endif
Set $k \leftarrow k + 1$;
end
end
Stop. Point x^* is an approximative solution of the problem.

A detailed description of different VNS variants is out of this paper's scope and can be found in [4] by Hansen and Mladenovic, 2001. Some recent successful VNS applications can be found in:

- Xiao et al. in [12] presented a reduced VNS algorithm and several implementation techniques for solving uncapacitated multilevel lot-sizing problems. The algorithm is competitive against other methods, enjoying good effectiveness as well as high computational efficiency.

- Kratica et al. in [7] considered a VNS approach for the task assignment problem. An appropriate neighborhood scheme along with a shaking operator and local search procedure are constructed specifically for this problem. The proposed VNS approach reached all optimal solutions quickly.

- Mladenović et al. in [11] presented a VNS approach to solving the one-commodity pickup-and-delivery traveling salesman problem. Using a binary indexed tree, they efficiently update the data structures mainly used for solving the classical TSP for feasibility checking in the neighborhoods. The proposed VNS-based heuristics outperformed the best-known algorithms in

terms of both solution quality and computational efforts and improved the best-known solution of all benchmark instances from the literature (with 100 to 500 customers). They were also able to solve instances with up to 1000 customers.

- Labadie et al. in [8] investigated the Team Orienteering Problem (TOP), an NP-hard problem that arises in vehicle routing and production scheduling contexts. They proposed a VNS procedure based on the idea of exploring granular instead of complete neighborhoods in order to improve the algorithm's efficiency without losing effectiveness. The method comes out to be, on average, quite effective allowing to improve the best known values for 25 test instances.
- Carrizosa et al. in [2] applied VNS metaheuristics to continuous optimization problems. Instead of perturbing the incumbent solution by randomly generating a trial point in a ball of a given metric, they proposed perturbing the incumbent solution by Gaussian distribution. Computational results showed some advantages of this new approach.

For solving the FTSP the following optimization problem is considered:

$$\min_{x \in X} \ f(x).$$

In order to apply VNS metaheuristics for FTSP, first of all the objective function $f(x)$ and the set X must be defined. The set $X = \{1, ...n\}^n$ is a set of n-dimensional vectors with integer elements, $1 \leq x_k \leq n$, where n is the number of edges in the FTS problem. For an arbitrary vector $x \in X$ the value of the function $f(x)$ is obtained as follows. The elements x_k of x are considered to be priorities assigned to each edge e_k in the graph. With given edge priorities, the unique schedule is formed following this algorithm:

1) All edges are uniquely sorted according to their priority. In the case of identical priority values, the edge with larger length has priority, and if the lengths are the same, the one with the smaller index.

2) All time lengths l_i are integers, so in the optimal schedule all edges are started at integer time moments. Starting with time $t = 0$ the algorithm tries to find the first feasible edge from the sorted array, i. e., that does not violate vertex constraints. To this edge a starting time t will be assigned. The same procedure is repeated for the remaining nonscheduled edges. If no edge could be started at time t, t is increased by 1 and the whole procedure repeated until all edges e_k get their starting times t_k.

For the obtained schedule, the value of the function $f(x)$ is defined as the maximal ending time for all edges, so $f(x) = max_{1 \leq k \leq n}(t_k + l_k)$.

Input parameters for VNS are: minimal and maximal number of neighborhoods which should be searched, $k_{\min}$ and $k_{\max}$, maximal number of iterations and p which represents probability of moving from one solution to another with same value of the objective function.

Initialization is carried randomly. At the beginning of the VNS procedure for each graph an initial solution x is formed by a random permutation of the integer values $0, 1, \ldots, n-1$.

The neighborhood structures are the key elements of VNS and the performance depends on the choice of the neighborhood structure. The main difficulty is to find a balance between effectiveness and the chance to get out of a local minimum. For a given k, the neighborhood $N_k(x)$ contains all priority vectors which may differ from the current solution x in at most k index positions. The shaking step in VNS generates a new vector $x' \in N_k(x)$ following the next algorithm:

1) First, a vector of k distinct random integers from $0, 1, \ldots, n-1$ is formed. This vector $(i_1, \ldots, i_k)$ represents the indices of the solution vector which will be modified.

2) Using the values of $i_1, \ldots, i_k$ obtained at the previous step, another vector $(p_1, \ldots, p_k)$ is formed as a random permutation of these values. This vector contains priority values to be used for obtaining x'.

3) The vector x' is created as a copy of the current vector x, after which only k of its elements are modified as $x'(i_j) = p_j, j = 1, 2, \ldots, k$.

The solution x', obtained from the shaking procedure, is usually not even a locally optimal solution, so from x' a local search is performed. The local search explores the neighborhood of a solution x' in order to identify a new solution with a smaller objective function value. The main component of a local search is the definition of an appropriate neighborhood. The neighborhood of a given solution consists of all solutions which can be obtained from the given solution by modifying the current solution in some way. The small neighborhood of x', which is here explored for better solution, consists of all vectors obtained from x' by swapping two arbitrary elements. The main advantage of these neighborhoods is that they can be explored very fast. The first improvement strategy was used. This strategy iteratively scans the neighborhood of x' and as soon as it finds the first improving solution (i. e., with the smaller objective function value), the local search is restarted and the improved solution is passed as initial solution

to the next iteration. If there is no improvement, the solution x' is not changed. The local search stops when the solution can no longer be improved and the best solution is denoted by x''.

After the local search procedure it should be decided whether the search should move to the new solution x'' or not. The basic VNS algorithm is moving from solution x to solution x'' only if the objective function value for x'' is smaller than the value of the objective function for x. In the cases when the problem has many local minima with the same objective function value, moving from one to another can expand the search and increase the chance of finding a better solution. On the other hand, if this move is made every time when a solution with the same objective function value is found, there is a large probability of getting into cycles. To prevent both of these problems, in cases when $f(x) = f(x'')$ the proposed algorithm uses a parameter p which presents the probability of moving from one solution to another. Thus, after the local search, there are three possibilities:

- In the case when solution x'' is better than x, i. e., $f(x'') < f(x)$, set $x := x''$ and continue the search with the same neighborhood N^k.
- If $f(x'') > f(x)$, then repeat the search with the same x and the next neighborhood.
- If $f(x'') = f(x)$, then with probability p set $x := x''$ and continue the search with the same neighborhood N^k and with probability $1 - p$ repeat the search with the same x and the next neighborhood.

The stopping criterion of the proposed VNS was set to a limit on the maximum number of iterations. One iteration corresponds to constructing one new solution (shaking) and optimizing it locally in the local search step.

4. Experimental results. This section presents the experimental results which show the effectiveness of the VNS method applied to FTSP. All computations were carried out on a single core of the Intel Core 2 Duo 2.67 GHz PC with 4 GB RAM under the Windows XP operating system. The algorithm was coded in the C programming language.

For experimental testing randomly generated instances were used. Those instances include different numbers of vertices of the graph ($|V| = 5, 10, 30, 50, 100$) and different numbers of edges ($|E| = 10, 50, 100, 200, 500$). At the beginning of

the generation process, for each vertex the randomly generated integer port constraint is selected from the interval $[1, v_{\max}]$, where $v_{\max} = 8, 15, 50, 80, 150, 300$, depending on the number of vertices in the graph. In a similar way, the lengths for each edge were generated, choosing the random integer from the interval $[1, e_{\max}]$, where $e_{\max} = 8, 15, 50, 80, 150, 300$. After this, the adjacency matrix of the graph is randomly generated avoiding self-loops. After creating the instance in this way, if the graph is not connected, it is ignored and the new instance is constructed. Avoiding self-loops and disconnected graphs is done to make the instances more realistic. To diversify the testing process, for every pair of $|V|$ and $|E|$ ten instances were generated by using different random seeds.

The following values of the VNS parameters were used: $k_{\min} = 2$, $k_{\max} = 20$, $p = 0.4$, $iter_{\max} = 100$. The VNS was run 20 times for each problem instance and the results are summarized in Tables 1–3. Table 1 presents results for small size instances ($v_{\max}$ up to 10 and $e_{\max} = 10$) whose optimal solutions can be obtained by total enumeration. Tables 2 and 3 contain results obtained on medium size ($v_{\max}$=10, 30 and $e_{\max} = 50, 100$) and large scale instances ($v_{\max}$=30, 50, 100 and $e_{\max} = 200, 500$), respectively. The large CPU-times for instances in Table 3 are the reason for taking only 5 instead of 10 random instances as in Tables 1 and 2.

In order to verify the optimality of VNS solutions for small size instances, a total enumeration technique was applied. For doing this, all permutations of the set $\{1, \ldots, |E|\}$ are generated and the best (representing the schedule with the best makespan) is chosen. The number of permutations grew rapidly with the increase of the number of edges. Already for $|E| = 10$ the number of permutations was $10! = 3628800$, so, accordingly, the total enumeration obtained the solutions only for instances where the number of edges of graph G was smaller or equal to 10.

Table 1 is organized as follows:

- In the first column the test instance name is given. The instance name contains information about the number of vertices and the number of edges, respectively. For example, instance ftsp_50_200_0 has 50 vertices and 200 edges. The number 0 at the end is the ordinal number of the instance of that type.
- The second column contains the elementary lower bound calculated as described in Section 2.
- The third column contains optimal solutions which were obtained by total enumeration in case when the method finished its work.

- The fourth column, named VNS_{best}, contains the best objective function value found by VNS in 20 runs. The solutions equal to the proved optimal solutions from column 3 are marked as 'opt'.
- The next two columns contain the average execution time (t) used to reach the final VNS solution and the average total execution time (t_{tot}).
- The last two columns ($avnsp$ and σ) contain information on the average solution quality: $avnsp$ is an average relative percentage error of found optimums defined as $avnsp = \frac{1}{20}\sum_{i=1}^{20} vnsp_i$, where $vnsp_i = 100 \times \frac{VNS_i - VNS_{best}}{VNS_{best}}$ and VNS_i represents the VNS solution obtained in the ith run, whereas σ is the standard deviation of $vnsp_i, i = 1, 2, \ldots, 20$ obtained by the formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(vnsp_i - avnsp)^2}$.

Table 1. Experimental results on small size instances

Inst.	LB	opt	VNS_{best}	t (sec)	t_{tot} (sec)	avnsp (%)	σ (%)
ftsp_5_10_0	14	14	*opt*	0.018	0.021	0.000	0.000
ftsp_5_10_1	10	15	*opt*	0.020	0.021	0.000	0.000
ftsp_5_10_2	12	12	*opt*	0.015	0.017	0.000	0.000
ftsp_5_10_3	14	14	*opt*	0.018	0.020	0.000	0.000
ftsp_5_10_4	10	10	*opt*	0.012	0.014	0.000	0.000
ftsp_5_10_5	6	13	*opt*	0.014	0.018	0.000	0.000
ftsp_5_10_6	12	13	*opt*	0.017	0.020	0.000	0.000
ftsp_5_10_7	8	8	*opt*	0.018	0.019	0.000	0.000
ftsp_5_10_8	11	11	*opt*	0.013	0.014	0.000	0.000
ftsp_5_10_9	26	26	*opt*	0.019	0.020	0.000	0.000
ftsp_10_10_0	12	15	*opt*	0.013	0.019	0.000	0.000
ftsp_10_10_1	12	12	*opt*	0.007	0.009	0.000	0.000
ftsp_10_10_2	5	14	*opt*	0.010	0.012	0.000	0.000
ftsp_10_10_3	19	19	*opt*	0.019	0.020	0.000	0.000
ftsp_10_10_4	17	17	*opt*	0.013	0.015	0.000	0.000
ftsp_10_10_5	10	15	*opt*	0.013	0.014	0.000	0.000
ftsp_10_10_6	3	15	*opt*	0.011	0.013	0.000	0.000
ftsp_10_10_7	11	14	*opt*	0.011	0.012	0.000	0.000
ftsp_10_10_8	11	14	*opt*	0.012	0.013	0.000	0.000
ftsp_10_10_9	2	14	*opt*	0.010	0.011	0.000	0.000

As can be seen from Table 1, total enumeration finished its work and produced optimal solutions for all instances up to $|E| = 10$. VNS reached all optimal solutions and running time on all instances in less than 0.021 seconds.

Table 2. Experimental results on medium size instances

Inst.	LB	VNS_{best}	t (sec)	t_{tot} (sec)	avnsp (%)	σ (%)
ftsp_10_50_0	26	92	5.681	5.775	0.000	0.000
ftsp_10_50_1	17	81	4.733	4.811	0.000	0.000
ftsp_10_50_2	40	42	3.371	3.532	0.000	0.000
ftsp_10_50_3	31	57	4.147	4.212	0.000	0.000
ftsp_10_50_4	17	20	1.903	2.025	0.000	0.000
ftsp_10_50_5	32	*LBopt*	2.493	2.725	1.406	1.817
ftsp_10_50_6	36	*LBopt*	2.962	3.328	0.000	0.000
ftsp_10_50_7	85	*LBopt*	6.015	6.114	0.000	0.000
ftsp_10_50_8	85	*LBopt*	5.141	5.225	0.000	0.000
ftsp_10_50_9	44	*LBopt*	3.206	3.525	0.000	0.000
ftsp_30_100_0	199	*LBopt*	72.225	73.568	0.000	0.000
ftsp_30_100_1	45	68	26.481	27.229	0.000	0.000
ftsp_30_100_2	29	108	42.427	44.350	0.093	0.278
ftsp_30_100_3	108	*LBopt*	50.615	52.523	0.000	0.000
ftsp_30_100_4	31	67	26.903	27.724	0.970	4.188
ftsp_30_100_5	163	*LBopt*	59.277	60.400	0.000	0.000
ftsp_30_100_6	30	50	7.918	8.070	0.000	0.000
ftsp_30_100_7	177	*LBopt*	71.007	72.350	0.000	0.000
ftsp_30_100_8	45	96	42.533	43.724	0.000	0.000
ftsp_30_100_9	270	*LBopt*	98.240	100.092	0.000	0.000

Also, the optimal solution is reached in each of the 20 runs, which indicates high reliability of the VNS algorithm.

The data in Tables 2 and 3 are presented in a way similar to Table 1. Note that the third column from Table 1 is missing because the total enumeration could not finish its work on larger instances for $|E| > 10$. On the other hand, VNS can handle instances with larger number of elements. Although the optimal solution of the problem is not found, the elementary lower bound (LB) is known and it is easily calculated. If the solution obtained by VNS equals to LB, it is the optimal solution of FTSP. If the VNS solution is greater than LB, there is no indication whether the VNS solution is equal or near to optimal FTSP solution. In column 3 the best VNS solutions are presented. In cases when they are equal to LB the optimal solution is found and this is marked with 'LB opt'. In other cases the optimal solution might be found as well, but in the absence of an efficient exact method this is not known.

One of the significant characteristics of the VNS is its local search procedure, which effectively contributes to the intensification of the search process.

Table 3. Experimental results on large scale instances

Inst.	LB	VNS_{best}	t (sec)	t_{tot} (sec)	avnsp (%)	σ (%)
ftsp_30_500_0	167	168	9348.251	9705.179	3.075	1.746
ftsp_30_500_1	718	*LBopt*	34507.249	34860.305	0.000	0.000
ftsp_30_500_2	269	*LBopt*	11505.293	12513.212	0.074	0.189
ftsp_30_500_3	117	129	6187.198	6739.084	5.271	2.671
ftsp_30_500_4	528	*LBopt*	20450.232	20823.454	0.000	0.000
ftsp_50_200_0	73	567	1409.227	1430.328	0.000	0.000
ftsp_50_200_1	89	116	358.705	368.036	1.164	2.560
ftsp_50_200_2	423	*LBopt*	1033.391	1048.931	0.000	0.000
ftsp_50_200_3	102	109	322.114	326.639	5.872	4.132
ftsp_50_200_4	59	344	851.106	863.951	0.000	0.000
ftsp_100_500_0	208	578	22430.227	22846.668	0.000	0.000
ftsp_100_500_1	197	227	9998.633	10234.793	5.132	2.608
ftsp_100_500_2	300	301	13189.866	13547.214	3.090	1.929
ftsp_100_500_3	794	830	31584.480	32155.459	0.000	0.000
ftsp_100_500_4	154	224	10114.071	10383.340	1.674	4.057

The local search is very fast for small size instances. As can be seen from Table 3, for larger instances the performance is slower since it uses sorting for calculating objective function values, but it still contributes to the overall efficiency. It can also be seen that the instances of the same size have significantly different execution times. This is due to different efficiency of the local search step, i. e., in some instances the local minimum procedure performed in each iteration is much faster (reaches the local minimum more quickly) than in other instances.

For many instances the same solution was found in all 20 runs and *avnsp* and σ are both zero. For other instances they are positive. It is interesting to notice that sometimes *avnsp* is greater than σ and sometimes it is smaller. In situations where in one or few runs the $vnsp_i$ is significantly greater than the others, the σ value is greater than *avnsp*. Contrary, in absence of such outliers, the σ value is smaller than *avnsp*. Since *avnsp* and σ are average relative percentage error and standard deviation of relative error, the previous note agrees with the well-known theoretical results from mathematical statistics.

As mentioned before, for the large size instances the same number of iterations were executed in significantly different times. In order to avoid such large CPU times, the other experiment was performed. For each run, the stopping criterion was set to CPU time limited to 1000s, instead of the 100 iterations criterion used before. The results are summarized in Table 4.

In the cases where in all 20 runs for one instance the same solution was

Table 4. Experimental results on large size instances with CPU times limited to 1000s

Inst.	LB	VNS_{best}	avnsp (%)	σ (%)
ftsp_30_500_0	167	171	5.393	2.716
ftsp_30_500_1	718	*LBopt*	0.000	0.000
ftsp_30_500_2	269	*LBopt*	2.034	1.732
ftsp_30_500_3	117	135	4.327	3.738
ftsp_30_500_4	528	*LBopt*	0.000	0.000
ftsp_50_200_0	73	567	0.000	0.000
ftsp_50_200_1	89	116	0.000	0.000
ftsp_50_200_2	423	*LBopt*	0.000	0.000
ftsp_50_200_3	102	109	3.165	2.150
ftsp_50_200_4	59	344	0.000	0.000
ftsp_100_500_0	208	578	0.000	0.000
ftsp_100_500_1	197	237	9.008	4.292
ftsp_100_500_2	300	306	10.327	5.433
ftsp_100_500_3	794	830	0.000	0.000
ftsp_100_500_4	154	224	11.964	9.504

reached after 100 iterations, the same result was achieved after 1000s. As can be seen, in most cases 1000s was sufficient time to find the same minima as obtained in Table 3, but the *avnsp* and σ values are significantly greater, as expected. This indicates that for some runs VNS needed much more time to obtain values equal or close to VNS_{best}. The exceptions are instances ftsp_20_200_1 and ftsp_20_200_3, where 100 iterations were completed in less than 1000s, so after an extended execution time in 20 runs solutions closer to the obtained minimum were found. For four instances the minimal result is greater than in Table 3.

As can be seen from the tables, VNS can handle much larger instances than the neural network proposed in [1]. From the experimental results for the proposed VNS it can be seen that the elementary lower bounds for problems were not always equal to their optimal solutions as they were in [1]. This indicates that the presented test instances are harder to solve.

5. Conclusions. In this paper, the variable neighborhood search approach for solving the file transfer scheduling problem is presented. The suitable choice of neighborhood structures, efficient implementation of shaking and fast local search procedures resulted in promising experimental results. For all instances where the exact solution could be found by exact methods, VNS reached those solutions very fast. For larger instances in some cases the optimality of

the VNS solution is proved by matching with the elementary lower bound. From the experimental results it can be seen that VNS is able to solve even large size FTS problems. However, only static FTS test problems were considered, so the usefulness of this metaheuristic is yet to be tested on more realistic dynamic FTS problems (when the files arrive continuously).

This research can be extended in several ways. The dynamic variant of FTSP should be investigated and its efficiency compared with other metaheuristics used in practice. One possible direction of the future work can be parallelization of the presented VNS approach. Another direction can be improving the performance of VNS through hybridization with exact methods or other metaheuristics, particularly for large instances of the problem.

REFERENCES

[1] Akbari M. K., M. H. Nezhad, M. Kalantari. Neural Network Realization of File Transfer Scheduling. *The CSI Journal of Computer Science and Engineering*, **2** (2004), No 2&4, 19–29.

[2] Carrizosaa E., M. Dražić, Z. Dražić, N. Mladenović. Gaussian variable neighborhood search for continuous optimization. *Computers & Operations Research*, **39** (2012), No 9, 2206–2213.

[3] Coffman E. G., M. R. Garey, D. S. Johnson, A. S. Lapaugh. Scheduling File Transfers. *SIAM Journal of Computing*, **14** (1985), No 3, 744–765.

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

[5] Havill J. T., W. Mao. Greedy on-line file transfer routing. In: Proc. IASTED International Conf. on Parallel and Distributed Systems, 1997, 225–230.

[6] Higuero D., J. M. Tirado, F. Isaila, J. Carretero. 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.

[7] Kratica J., A. Savić, V. Filipović, M. Milanović. Solving the task assignment problem with a variable neighborhood search. *Serdica Journal of Computing*, **4** (2010), No 4, 435–446.

[8] Labadie N., R. Mansini, J. Melechovsky, R. Calvo. The team orienteering problem with time windows: an LP-based granular variable neighborhood search. *European Journal of Operational Research*, **220** (2012), No 1, 15–27.

[9] Mao W. Directed File Transfer Scheduling. In: Proc. of the ACM 31st Annual Southeast Conference, 1993, 199–203.

[10] Mladenović N., Hansen P. Variable neighbourhood search. *Computers and Operations Research*, **24** (1997), 1097–1100.

[11] Mladenović N., D. Urošević, S. Hanafi, A. Ilić. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, **220** (2012), No 1, 270–285.

[12] Xiao Y., I. Kaku, Q. Zhao, R. Zhang. A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems. *European Journal of Operational Research*, **214** (2011), No 2, 223–231.

Zorica Dražić
Faculty of Mathematics
University of Belgrade,
Studentski trg 16/IV, 11 000 Belgrade
Serbia
e-mail: zdrazic@matf.bg.ac.rs

Received May 1, 2012
Final Accepted July 19, 2012