Les Cahiers du GERAD ISSN: 0711–2440

Permutation-Based Genetic, Tabu and Variable Neighborhood Search Heuristics for Multiprocessor Scheduling with Communication Delays

T. Davidović, P. Hansen
N. Mladenović

G–2004–19

March 2004

Permutation-Based Genetic, Tabu and Variable Neighborhood Search Heuristics for Multiprocessor Scheduling with Communication Delays

Tatjana Davidović

Mathematical Institute, Serbian Academy of Science and Arts
Kneza Mihaila 35, 11000 Belgrade, Yugoslavia
tanjad@mi.sanu.ac.yu

Pierre Hansen

GERAD and HEC Montréal
3000 chemin de la Côte-Sainte-Catherine
Montréal H3T 2A7, Canada
pierre.hansen@gerad.ca

Nenad Mladenović

Mathematical Institute, Serbian Academy of Science and Arts
Kneza Mihaila 35, 11000 Belgrade, Yugoslavia
and GERAD
nenad@mi.sanu.ac.yu, nenad.mladenovic@gerad.ca

March, 2004

Les Cahiers du GERAD

G–2004–19

Abstract

The multiprocessor scheduling problem with communication delays that we consider in this paper consists of finding a static schedule of an arbitrary task graph onto a homogeneous multiprocessor system, such that the total execution time (i.e. the time when all tasks are completed) is minimum. The task graph contains precedence relations as well as communication delays (or data transferring time) between tasks if they are executed on different processors. The multiprocessor architecture is assumed to contain identical processors connected in an arbitrary way which is defined by a symmetric matrix containing minimum distances between each two processors. Solution is represented by a feasible permutation of tasks that is scheduled by the use of some constructive scheduling heuristic in order to obtain the objective function value, i.e. makespan. For solving this NP-hard problem, we develop basic Tabu Search and Variable Neighborhood Search heuristics, where various types of reduced Or-opt-like neighborhood structures are used for local search. A Genetic Search approach based on the same solution space is also developed. Comparative computational results on random graphs with up to 500 tasks and 8 processors are reported. It appears that Variable Neighborhood Search outperforms the other metaheuristics in average. In addition, a detailed performance analysis of both the proposed solution representation and heuristic methods is presented.

Keywords: Task Scheduling, Communication Delays, Metaheuristics, Variable Neighborhood Search, Tabu Search, Genetic Algorithms.

Résumé

Le problème d'ordonnancement à processeurs multiples avec délais de communication que nous considérons dans cet article consiste à déterminer un ordonnancement statique d'un graphe de tâches arbitraire sur un système multiprocesseurs homogène de sorte que le temps total d'exécution (c'est-à-dire le temps auquel toutes les tâches sont exécutées) soit minimum. Le graphe de tâches contient des relations de précédence ainsi que des délais de communication (ou temps de transfert de données) entre les tâches si elles sont exécutées sur des processeurs différents. L'architecture multiprocesseurs est supposée contenir des processeurs identiques connectés d'une manière arbitraire qui est définie par une matrice symétrique contenant les distances minimum entre chaque paire de processeurs. La solution est représentée par une permutation admissible de tâches ordonnancée par le recours à une heuristique d'ordonnancement constructive donnant la valeur de la fonction économique. Pour résoudre ce problème NP-difficile, nous développons des heuristiques de base de types Recherche avec Tabous et Recherche à Voisinage Variable, où divers types de structures de voisinage Or — Opt réduites sont utilisées dans la recherche locale. Une Recherche Génétique basée sur le même espace de solution est également développée. On présente des résultats comparatifs de calcul sur des graphes aléatoires avec jusqu'à 500 tâches et 8 processeurs. Les résultats de la Recherche à Voisinage Variable sont, en moyenne, meilleurs que ceux des autres métaheuristiques. De plus une analyse détaillée des performances de la représentation de la solution proposée ainsi que des méthodes heuristiques est présentée.

Mots Clés : Ordonnancement de tâches, Délais de communication, Métaheuristiques, Recherche à Voisinage Variable, Algorithmes Génétiques.

1 Introduction

The Multiprocessor Scheduling Problem with Communication Delays (MSPCD) can be stated as follows: tasks (modules or jobs) have to be executed on a multiprocessor system; we have to find where and when each task will be executed, in order that the total completion time be minimum. The duration of each task is known as well as precedence relations among tasks, i.e. which tasks should be completed before some other ones can begin. In addition, if dependent tasks are executed on different processors, data transferring times (or communication delays), that are given in advance, are also considered.

Scheduling parallel tasks among processors with and without communication delays is a NP-hard problem as it was shown in J. D. Ullman (1975). However, some special cases can be solved in polynomial time as it is described in V. Krishnamoorthy and K. Efe (1996), T. A. Varvarigou, V. P. Roychowdhury, T. Kailath, and E. Lawler (1996). There are many extended and restricted versions of the multiprocessor scheduling problem suggested in the literature (see for example J. Blazewicz, M. Drozdowski, and K. Ecker (2000), P. Brucker (1998), and M. Drozdowski (1996) for surveys), but the most studied is the case where there are no precedence relations among tasks and/or no communication delays.

Among classical constructive heuristics, we emphasize CP (Critical Path) G. C. Sih and E. A. Lee (1993) and LPT (Largest-Processing-Time-first) B. Chen (1993). Many papers proposing constructive heuristic solutions can be found in the literature (M. Drozdowski (1996)). Recently metaheuristic approaches have been applied as well.

Genetic Algorithms (GA) have been proposed in I. Ahmad and M. K. Dhodhi (1996), E. S. H. Hou, N. Ansari, and H. Ren (1994), Y.-K. Kwok and I. Ahmad (1997). The heuristics developed in I. Ahmad and M. K. Dhodhi (1996), E. S. H. Hou, N. Ansari, and H. Ren (1994) assumed communication time to be negligible and performed scheduling onto a complete crossbar interconnection network of processors (the so-called parallel processors). In E. S. H. Hou, N. Ansari, and H. Ren (1994) the members of the population were represented by lists of tasks each list containing tasks associated with one of the processors. The crossover and mutation operations were defined in such a way as to take into account the precedence relations between tasks.

A different approach was used in I. Ahmad and M. K. Dhodhi (1996): genetic heuristic was combined with a list-scheduling heuristic and the so-called Problem Space Genetic Algorithm (PSGA) was developed. The population members (chromosomes) were represented by arrays of task priorities. The first member in the initial population was determined by using the CP method in calculating task priorities. The rest of the chromosomes in this initial population were generated by random perturbations in the priorities (genes) of this first chromosome. In each generation, a list-scheduling heuristic was applied to all of the chromosomes to obtain corresponding cost function values (schedule lengths).

In the parallel GA proposed in Y.-K. Kwok and I. Ahmad (1997) members of the population were represented by feasible permutations of tasks. The initial population was generated randomly with the CP permutation included. An ordered crossover operation was used to assure that precedence constraints between tasks are not violated. Mutation was realized by exchanging positions of two neighbor independent tasks. The whole population was divided into q parts, each processed by a single processor. The best solution was exchanged between processors every $T = N_g/2^n$ iterations $n = 1, 2, \ldots$. Mutation and crossover probabilities were adaptive, i.e. their values were different on different processors.

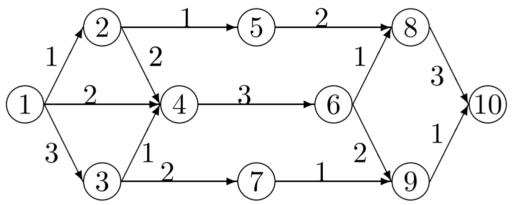
In A. Thesen (1998) the Tabu Search (TS) metaheuristics was used to schedule a set of n independent tasks on a network of m processors. Sequential and parallel TS approaches were used in S.C. Porto and C.C. Ribeiro (1995), S.C. Porto and C.C. Ribeiro (1996) for solving MSPCD on heterogeneous processors. Solutions were represented by ordered lists of tasks associated with processors. Tasks were executed on associated processor in the order defined by these lists as soon as they became executable. The Swap-1 neighborhood was defined by moving a task from one processor to all others in turn and placing it in all possible positions (not violating the precedence constraints). That neighborhood was reduced by considering only "promising" moves. Recent moves are kept in a tabu list as forbidden for next *ntabu* iterations in order to avoid cycling. For calculation of the objective function value, i.e. scheduling length computation a greedy heuristic DES+MFT proposed in D. A. Menascé and S. C. S. Porto (1992) is used.

In this paper we exploit several metaheuristic, or frameworks to build heuristics, using the same solution representation for solving MSPCD. A solution is represented by a feasible permutation of tasks, obeying precedence constraints (dependencies) between tasks. To obtain the criterion function value, the Earliest Start (ES) scheduling heuristic is used. We develop basic VNS, TS and GA approaches, and compare them with each other and with Multistart Local Search (MLS) and PSGA which we modified for the MSPCD case. All heuristics are compared, within the same CPU time limit on two sets of random task graphs. The first type of task graphs is composed of arbitrary graphs, with given graph's edges densities, while the second set consists of random task graphs with known optimal solutions for given multiprocessor architectures. In addition, a sensitivity analysis is performed to examine the influence of the parameter values on the solution quality.

This paper is organized as follows. The next section contains a combinatorial formulation of MSPCD, while in section 3 it is explained how we apply selected metaheuristics for solving MSPCD. In Sections 4 and 5 comparative computational results and sensitivity analysis are presented. Section 6 concludes the paper.

2 The multiprocessor scheduling problem

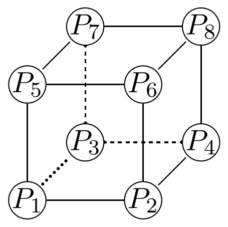
The tasks to be scheduled are represented by a directed acyclic graph (DAG) defined by a tuple $\mathcal{G} = (T,\ E,\ C,\ L)$ where the node set $T = \{t_1, \ldots, t_n\}$ denotes the set of tasks; the edge set $E = \{e_{ij} \mid t_i,\ t_j \in T\}$ represents the set of communication edges; $C = \{c_{ij} \mid e_{ij} \in E\}$ denotes the set of edge communication costs; and $L = \{l_1, \ldots, l_n\}$ represents the set of task computation times (execution times, lengths). The communication cost $c_{ij} \in C$ denotes the amount of data transferred between tasks t_i and t_j if they are executed on different processors. If both tasks are scheduled to the same processor the communication cost equals zero. The set E defines precedence relation between tasks. A task cannot be executed unless all of its predecessors are completed and all relevant data are available. Task preemption and redundant execution are not allowed. An example of task graph is given on Figure 1. Node labels represent task numeration, task lengths are given in a corresponding table, while edge labels denote communication costs.



t_i	1	2	3	4	5	6	7	8	9	10
l_i	6	8	7	10	8	12	9	6	8	7

Figure 1: An example of task graph

The multiprocessor architecture $\mathcal{M}$ is assumed to contain p identical processors with their own local memories which communicate by exchanging messages through bidirectional links of the same capacity. This architecture can be modelled by an undirected graph (G. C. Sih and E. A. Lee (1993)) or by a *distance matrix* (T. Davidović (2000), G. Djordjević and M. Tošić (1996)). The nodes of processor graph represent processors while the edges define the connection between processors. The element $(i,\ j)$ of the distance matrix $D = [d_{ij}]_{p\times p}$ is equal to the minimum distance between nodes i and j. Here, the minimum distance is calculated as the number of links along the shortest path between two nodes. It is obvious that the distance matrix is symmetric with zero diagonal elements. Figure 2 contains the picture of a 3-dimensional hypercube of identical processors and the corresponding distance matrix ($p = 8$).



$$D = \begin{bmatrix} 0 & 1 & 1 & 2 & 1 & 2 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 1 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 1 & 2 & 2 & 3 & 0 & 1 & 1 & 2 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 2 & 3 & 1 & 2 & 1 & 2 & 0 & 1 \\ 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2: 3-dimensional hypercube multiprocessor architecture

The scheduling of DAG $\mathcal{G}$ onto $\mathcal{M}$ consists of determining the index of the associated processor and starting time for each of the tasks from the task graph in such a way as to minimize some objective function. The usual objective function (that we shall use in this paper as well) is completion time of the scheduled task graph (also referred to as makespan, response time or schedule length). The starting time of a task t_i depends on the completion times of its predecessors and the amount of time needed for transferring the data from the processors executing these predecessors to the processor that has to execute the task t_i. The time that is spent for communication between tasks t_i and t_j can be calculated in the following way

$$\gamma_{ij}^{lk} = c_{ij} \cdot d_{lk} \cdot ccr,$$

where it is assumed that task t_i will be executed at processor p_l, task t_j at processor p_k. ccr represents the Communication-to-Computation-Ratio which is defined as the ratio between time for transferring a unit of data and the time needed to perform a single computing operation. This definition is different from the CCR parameter of Kwok and Ahmad Y.-K. Kwok and I. Ahmad (1997) introduced to characterize task graph structure. If both the tasks are scheduled to the same processor, i.e. $k = l$, the amount of communication is equal to zero since $d_{kk} = 0$. Here, we gave the graph based formulation that is used below for the implementation of metaheuristic methods. A mathematical programming model of MSPCD is given in T. Davidović, N. Maculan, and N. Mladenović (2003).

According to classification used in P. Brucker (1998), B. Veltman, B. J. Lageweg, and J. K. Lenstra (1990) the MSPCD can be denoted by $P^*|prec|C_{max}$ where P^* is used to describe multiprocessor system with identical processors connected in an arbitrary way (not necessarily complete crossover interconnection as it is the case in parallel architectures). In other words, we have to schedule tasks with arbitrary precedence constraints and arbitrary execution times on p identical processors connected in an arbitrary way such that the makespan is minimized.

3 Applying metaheuristic

In order to apply metaheuristics for solving MSPCD we first need to define a solution space S and a set of so-called *feasible* solutions $X \subseteq S$. Let S be the set of all permutations of n tasks, and let x, $(x \in X)$ be a feasible solution (feasible permutation means that the order of tasks in that permutation obeys precedence constraints defined by the task graph: a task cannot appear before any of its predecessors or after any of its successors in a feasible permutation). Having a feasible permutation x, we are able to evaluate the objective function value in a unique way, if we follow always the same rule of assigning tasks to processors (for example, ES rule) in the order given by that permutation (see I. Ahmad and M. K. Dhodhi (1996) and T. Davidović (2000)). Therefore, the solution set of MSPCD can be represented by S.

3.1 Neighborhoods

By presenting a solution of MSPCD as permutation of tasks, we can use several neighborhood structures analogous to the very well known ones used in solving the Travelling salesman problem, such as 2-opt, 3-opt, Or-opt etc. A solution x' (or a tour that consists of n cities) belongs to a k-opt neighborhood of x if it has exactly k different and $n - k$

same edges as x. Special cases of the 3-opt neighborhood are known as Or-opt (I. Or (1976)) neighborhoods, where 1,2 or 3 consecutive cities in the tour are inserted between all other cities in the tour. Notice that 1-Or-opt neighborhood is usually called *insertion* neighborhood. Observe also that an interchange neighborhood (where cities exchange their places in the tour) is a special case of 4-opt neighborhood, since four edges in the tour are reconnected. It is necessary to point out here that there is no complete analogy in the neighborhood definitions and we will use term *Swap* to denote Or-opt-like neighborhoods. Now we give more details regarding neighborhoods used for solving the MSPCD, and later we explain which among them we include in the list for Variable Neighborhood Descent (VND) local search.

Swap-1. A Swap-1 neighbor of a feasible solution x is defined by moving a task from one position to another. For example, swap-1 neighbors of the feasible permutation 1-3-2-5-7-4-6-9-8-10 for the task graph given on Figure 1 are 1-2-3-5-7-4-6-9-8-10, 1-2-5-3-7-4-6-9-8-10, and so on (see Figure 3). The total number of feasible permutations in this example is 74, while edge density ρ equals 0.31. These two parameters are closely connected and (as we will explain in Section 5) they strongly influence the quality of the heuristic solutions obtained.

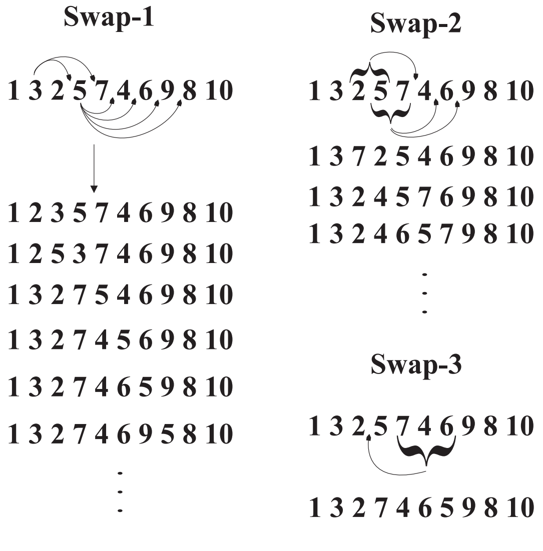


Figure 3: Examples of Swap neighborhoods

Swap-2. A Swap-2 neighborhood is defined by changing the positions of two succeeding arbitrary tasks.

Swap-3. In the Swap-3 neighborhood we move three succeeding tasks between all possible two tasks (Figure 3);

Interchange (IntCh). The IntCh neighborhood is defined by all possible exchanges of positions of two arbitrary tasks. Neighborhoods are reduced because only feasible permutations are considered, i.e., permutation 1-7-3-2-5-4-6-9-8-10 is not in the Swap-3 neighborhood of permutation 1-3-2-5-7-4-6-9-8-10 from the previous example (3-2-5 is moved after 7) since it is not feasible (see Figure 1).

In this paper we experimented with all these neighborhoods as well as with the combinations of some of them (or even all of them). The most natural combination is Swap-3; Swap-2; Swap-1 (in that order), which we called Swap-321 for short. We used this combination when trying to improve the scheduling results of basic variants of our heuristics. The basic idea of the Swap-321 is "try to improve the result with a small number of big moves first and then perform fine moves since the number of fine moves is larger". According to task dependencies defined by precedence relations between tasks, it is obvious that the Swap-3 neighborhood is the smallest one, since the probability to find dependent tasks between 4 chosen ones is bigger then between two tasks only.

The next step in applying metaheuristics is to decide how to represent the solution, i.e., what data structure should be used to make our implementation more efficient? Here we use a "double-link" data structure which allows us to generate a neighbor in $O(1)$ steps.

For each neighbor in a selected neighborhood we have to calculate the objective function value f (i.e. schedule length, makespan). Updating f is not easy because of the data dependencies and the communication delays between tasks (whose values may be different for different schedules since they depend on tasks allocation). Therefore, we have to calculate the value for f each time we generate a new neighbor. The calculation of f involves the application of a selected constructive scheduling rule, with the usual complexity of $O(n^2p)$. To reduce the execution time spent for scheduling we performed reallocation only for changed part of feasible permutation. For example, starting from 1-3-2-5-7-4-6-9-8-10, we determine Swap-1 neighbor 1-3-2-5-7-4-6-8-9-10 and perform scheduling only for tasks 8, 9 and 10. To avoid neighbor duplication we reduce neighborhoods by considering one-side moves only. More precisely, neighborhood search is performed by moving tasks only to the right (FORWARD search) or only to the left (BACKWARD search) as illustrated in Figure 4.

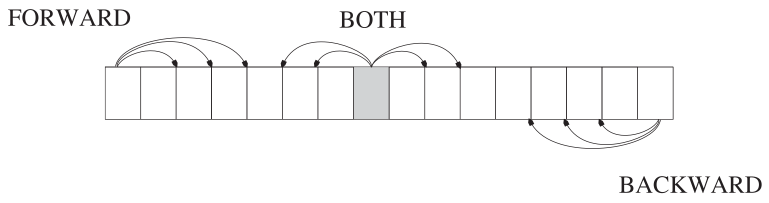


Figure 4: Neighborhood search directions

Notice that the reduced neighborhoods obtained by forward and backward search can differ but jointly they cover the whole neighborhood. This obvious property is illustrated bellow on the example of Figure 1. Starting from the feasible permutation 1-2-3-4-5-6-7-8-9-10 by performing only forward (backward) search we obtain following neighbors:

forward search	backward search
1-3-2-4-5-6-7-8-9-10	1-2-3-4-5-6-7-9-8-10
1-2-3-5-4-6-7-8-9-10	1-2-3-4-5-6-8-7-9-10
1-2-3-4-6-5-7-8-9-10	1-2-3-4-5-7-6-8-9-10
1-2-3-4-6-7-5-8-9-10*	1-2-3-4-7-5-6-8-9-10**
1-2-3-4-5-7-6-8-9-10	1-2-3-7-4-5-6-8-9-10**
1-2-3-4-5-6-8-7-9-10	1-2-3-4-6-5-7-8-9-10
1-2-3-4-5-6-7-9-8-10	1-2-3-5-4-6-7-8-9-10
	1-2-5-3-4-6-7-8-9-10**
	1-3-2-4-5-6-7-8-9-10

We marked by star (*) the neighbor which is only in the forward part and by (**) the neighbors that are only in the backward part of Swap-1 neighborhood.

In all heuristics we start with (at least one) initial solution x which can be determined as the first feasible permutation in topological order, or permutation obtained by the use of some constructive heuristic method (like LPT or nonincreasing CP priority of the tasks) or any randomly chosen feasible permutation.

Local search (LS) in such a neighborhood $\mathcal{N}(x)$ of the given initial solution x is defined as performing scheduling for all the neighbors and calculating the obtained schedule lengths. If a better solution x' is found we move there and look for improvement in the neighborhood $\mathcal{N}(x')$ of this new solution. This process is repeated until there is no better solution in $\mathcal{N}(x)$. Rules for LS are as follows:

1. *Initialization.* Select an initial solution x (at random or by the use of some constructive heuristic method).
2. *Repeat*
   (a) Search for $x' \in \mathcal{N}(x)$ such that the schedule length for x' is less than the schedule length for x, i.e. $f(x') < f(x)$.
   (b) Set $x = x'$

   until there is no such x'.

We try to improve LS results by applying metaheuristic rules to the local minimum solution obtained.

3.2 Multistart Local Search (MLS)

MLS is realized by restarting LS from random initial feasible permutations until a stopping criterion (number of restarts, CPU time limit) is satisfied and save the best obtained local minimum:

Repeat

1. *Initialization.* Select an initial solution x (at random). If this is the first iteration set $x_{opt} = x$.
2. *LS.* Apply some Local Search procedure with x as an initial solution; denote with x' the obtained local optimum.
3. If the schedule length of x' is less that the schedule length of x_{opt}, set $x_{opt} = x'$.

until a stoping criterion is satisfied.

3.3 Variable Neighborhood Search (VNS)

To describe the VNS approach (see P. Hansen and N. Mladenović (1999), P. Hansen and N. Mladenović (2001)b, P. Hansen and N. Mladenović (2001)a, P. Hansen and N. Mladenović (2002), N. Mladenović and P. Hansen (1997)), let $x \in X$ be an arbitrary solution and $\mathcal{N}_k$, $(k = 1, \ldots, k_{max})$, a finite set of pre-selected neighborhood structures. Then $\mathcal{N}_k(x)$ is the set of solutions in the k^{th} neighborhood of x. Steps in the basic VNS are:

Initialization. Find an initial solution $x \in X$; choose a stopping condition.

Repeat until the stopping condition is met:

1. Set $k = 1$;
2. Until $k = k_{max}$ repeat the following steps:
   (a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x, ($x' \in N_k(x)$);
   (b) *Local search.* Apply some local search method with x' as initial solution; denote with x'' the obtained local optimum;
   (c) *Move or not.* If this local optimum is better than the incumbent, move there ($x = x''$), and continue the search with $\mathcal{N}_1$ ($k = 1$); otherwise, set $k = k + 1$.

This is the basic version of the VNS procedure; several modifications and refinements are described e. g. in P. Hansen and N. Mladenović (2001)a, P. Hansen and N. Mladenović (2002).

The role of the shaking operation is to assure moves which bring the search far from the currently best local minimum in a progressive way. This assures intensification of search around good solutions found as well as diversification as the size of neighborhoods is usually increasing.

In our implementation, shaking in the k-Swap neighborhood (i.e. the selection of random neighbor in k-th neighborhood) is realized by performing k times the following steps:

1. select a task to be moved at random;
2. calculate the moving scope for that task, i.e., find indices of its last predecessor and its first successor (with respect to the incumbent solution);

3. find the position where the selected task will be inserted (a random number between the two indices found in the previous step);
4. make a move, i.e., insert the selected task in the new position obtained in step 3.

Please note that we use different notation for neighborhoods used in LS and shake procedures. For LS Swap-k means that we move k succeeding tasks, while shake in k-Swap is the substitution for performing k times a random Swap-1 move.

For k-interchange neighborhood, the procedure for shaking is as follows:

```
1. for $i = 1, k$
   (a) DONE = FALSE;
   (b) while not DONE
       i. select randomly two tasks;
       ii. if they can exchange their positions, i.e., if those tasks are independent and
           all the tasks between these two are neither successors of the first task nor
           predecessors of the second one;
           A. DONE = TRUE;
           B. exchange the positions of these two tasks;
```

The main parameter for the VNS is k_{max}, i.e., the maximum number of neighborhoods used (P. Hansen and N. Mladenović (2001)a, N. Mladenović and P. Hansen (1997)). In some modifications of the basic VNS additional parameters can be used such as n_{step}–step for changing the neighborhoods, *plateaux*–probability to take a solution with scheduling length equal to current minimum solution as a new minimum and so on (see P. Hansen and N. Mladenović (2001)b, P. Hansen and N. Mladenović (2001)a, P. Hansen and N. Mladenović (2002)). Moreover, the neighborhood explored in step 2b is not necessarily a single one; rather it may consists of a combination of neighborhoods, Swap-321 for example. In another words, we may use the Variable Neighborhood Descent (VND) as a LS procedure (P. Hansen and N. Mladenović (2001)a, P. Hansen and N. Mladenović (2003)).

Variable Neighborhood Descent (VND). It is deterministic variant of VNS; the idea is to get local minimum with respect to all pre-defined neighborhood structures $N_1, N_2, \ldots, N_{\ell_{max}}$. Its steps are:

Initialization. Select the set of neighborhood structures N_ℓ, for $\ell = 1, \ldots, \ell_{max}$, that will be used in the descent; find an initial solution x;

Repeat the following sequence until no improvement is obtained:

```
(1) Set $\ell \leftarrow 1$;
(2) Repeat the following steps until $\ell = \ell_{max}$:
    (a) Exploration of neighborhood. Find the best neighbor $x'$ of $x$ ($x' \in N_\ell(x)$);
    (b) Move or not. If the solution $x'$ thus obtained is better than $x$, set $x \leftarrow x'$ and
        $\ell \leftarrow 1$; otherwise, set $\ell \leftarrow \ell + 1$.
```

As mentioned before, in the Parametric analysis section we investigate different choices of $\ell_{max} = 7$ neighborhoods: Swap-1, Swap-2, Swap-3 (forward and backward) and IntCh.

3.4 Tabu Search (TS)

TS metaheuristic (F. Glover and M. Laguna (1997)) is implemented by the use of variable sized tabu list (TL) to store tasks that should not be moved in succeeding iterations. The maximum length of the tabu list ($NTABU$) is a parameter, and the actual length is determined randomly in each iteration. This variant happens to perform better than the basic one with fixed length of TL.

Initialization. Find an initial solution $x \in X$; set $TL = \emptyset$; choose a stopping condition.

Repeat until the stopping condition is met:

1. *Update* $|TL|$ *(length of* TL*).* Generate randomly an integer in range $(1, NTABU)$ to represent the actual length of TL in this iteration.
2. *LS.* Find $x' \in \mathcal{N}(x) \setminus TL$ with minimum value for schedule length.
3. *Update* x*.* If the scheduling length of x' is less than the scheduling length of x_{opt} set $x_{opt} = x'$
4. *Move.* Set $x = x'$.
5. *Update TL.* $TL = TL \cup \{x'\}$; if $|TL| > NTABU$ then $TL = TL \setminus \{x^t\}$ where x^t is the oldest task in TL (FIFO update rule).

3.5 Genetic Algorithms (GA)

Feasible permutations are members of the population in GA, ordered crossover and mutation are realized as in Y.-K. Kwok and I. Ahmad (1997). These operations are applied to all members of the population (with given probabilities).

Initialization. Generate Np different feasible solutions (by the use of constructive heuristic methods and/or randomly).

Repeat until the stopping condition is met:

1. *Evaluation.* Calculate values for schedule length of each solution.
2. *Update.* Save solution x' with minimum schedule length.
3. *Create new generation*
   (a) *Selection.* Choose solutions to be moved to the next generation.
   (b) *Crossover.* With a given probability apply the crossover operator to all pairs of solutions.
   (c) *Mutation.* With a given probability apply the mutation operator to all tasks in each solution.

We also implemented PSGA developed in I. Ahmad and M. K. Dhodhi (1996) and adopted it to MSPCD by taking into account required communications and multiprocessor architecture. Parameters of GA and PSGA are p_{xover}–crossover probability, p_m–mutation probability, Np–population size (number of solutions in population), and Ng–number of generations (if this is stopping criterion), see D. E. Goldberg (1989).

All metaheuristics are compared on random task graphs within the same CPU time limit and results are reported in the next section. A performance analysis is presented in Section 5.

4 Experimental results

We implemented the proposed heuristics in the C programming language on an Intel Pentium III processor (800 MHz) with Linux operating system.

To illustrate the efficiency of our approach we tested it on randomly generated task graphs containing up to $n = 500$ tasks. Due to the lack of adequate benchmarks for this variant of scheduling problem, we generated our own test examples. These examples are described in T. Davidović and T. G. Crainic (2003) and contain two type of random task graphs. The first group of task graphs is generated with preselected edge density ρ ranging from 0.2 to 0.8. These task graphs are scheduled onto different multiprocessor architectures containing up to $p = 8$ processors. Sparse task graphs (with $\rho = 0.2$) are scheduled onto a 3-dimensional hypercube (see Figure 2). Task graphs with $\rho = 0.4, 0.5$ are scheduled onto a 2-dimensional hypercube (ring of four processors), while dense task graphs ($\rho = 0.6, 0.8$) are allocated to $p = 2$ processors. The selection of multiprocessor architecture was guided by experimental analysis from T. Davidović and T. G. Crainic (2003). It appeared that dense graphs can not benefit from adding new processors to the multiprocessor system since too much time is spent on data transfer between tasks scheduled to different processors. For each n we generated 30 task graphs with different ρ and other relevant characteristics trying to generate different types of examples (dense-sparse, computation-communication intensive, with uniform-nonuniform task-communication lengths, etc.).

We compared schedule length obtained by the use of VNS, TS, MLS, PSGA, and GA metaheuristics. Comparative results are presented in Table 1. The first column of that table contains the number of tasks in the task graphs, the best (in average) scheduling result (among all heuristics) is given in the second column, while in the remaining seven columns average values of heuristic schedule lengths are presented. Third column contains average over 30 examples value of constructive heuristic (CP based permutation scheduled by ES rule) solutions which serve as the initial value for all iterative (meta) heuristic methods. The average value of local minimum schedules starting from the initial solution is given in column four. The remaining 5 columns contain average values of results obtained by basic variants of heuristic methods.

Now, let us summarize the parameters we used during the different heuristics searches. For VNS we used Swap-1 neighborhood in forward direction. The value of k_{max} parameter is set to 20, and first improvement (FI) search is explored according to the result from P. Hansen and N. Mladenović (1999)a and our experiments showing that FI search performs faster and better than best improvement (BI) one starting from random initial solution as it is the case each time shaking rule is applied. The values of other parameters are $k_{step} = 1$ and $plateaux = 0.0$. The values of these parameters are determined according to previous experience and will be analyzed later (see the next section). In TS the Swap-1 neighborhood is explored in forward direction. The value for $NTABU$ is set to $n/2$, and

Table 1: The Multiprocessor scheduling: average results for each n over 30 random tests.

	Best	*% Deviation*						
n	(av.)	CP	LS	VNS	TS	MLS	GA	PSGA
50	437.5	37.4	31.6	0.02	0.15	0.09	0.08	5.73
100	983.5	26.7	19.9	0.01	0.07	0.02	0.02	3.92
200	2137.6	17.1	12.2	0.01	0.08	0.02	0.03	2.56
300	3251.74	13.96	8.8	0.01	0.11	0.06	0.04	4.12
400	4454.46	12.4	7.7	0.01	0.12	0.08	0.02	5.17
500	5599.5	11.5	7.2	0.01	0.08	0.03	0.03	4.66

Table 2: The CPU time for heuristic scheduling: average results for each n over 30 random tests.

	CPU time	*CPU time (seconds)*				
n	(max.)	VNS	TS	MLS	GA	PSGA
50	35.6	16.89	33.16	18.33	25.02	28.96
100	140.0	87.20	116.30	63.85	112.22	129.13
200	616.0	488.30	608.11	532.43	551.41	566.87
300	1400.0	924.19	1288.12	886.73	1227.33	1335.28
400	2440.0	1694.63	2360.70	1866.20	2262.52	1992.70
500	3860.0	2693.31	3032.18	2270.62	3622.98	3620.83

the actual value for the length of the tabu list is determined randomly in each iteration. The values of GA (PSGA) parameters are $N_p = 120$, $p_{xover} = 60$ and $p_m = 0.01$.

For the stopping criterion we chose t_{max}, defined as the time needed by GA to perform 600 generations. Average CPU time spent by each method to find the best heuristic solution obtained are presented in Table 2 with number of tasks and maximum allowed CPU time in first two columns. We tried with smaller number of GA generations, but it turned out that the number of iterations for other heuristics was very small (10 at most for MLS, and k_{max} in VNS was reached 1-2 times only).

To examine the search process, we followed the improvement of global minimal solution in time. The results are presented on Figure 5. We set $n = 200$, choose one example (task graph) from the 30 of the same size, and presented the changes in the value of global minimal solution during the $[0,\ t_{max}]$ time interval for all heuristic methods.

As can be seen from Table 1 and Figure 5, VNS performs best in average for this type of randomly generated task graphs. However, we can not discuss the quality of the heuristic solutions obtained since we have no information about the best possible solution and the most suitable multiprocessor architecture for these task graphs.

Therefore, we generated a second set of random task graphs with known optimal schedule lengths on given multiprocessor architecture. The generation procedure was described in T. Davidović and T. G. Crainic (2003),Y.-K. Kwok and I. Ahmad (1997). Here, we selected task graphs generated for 2-dimensional hypercube multiprocessor architecture, i.e. we set $p = 4$ (processor ring).

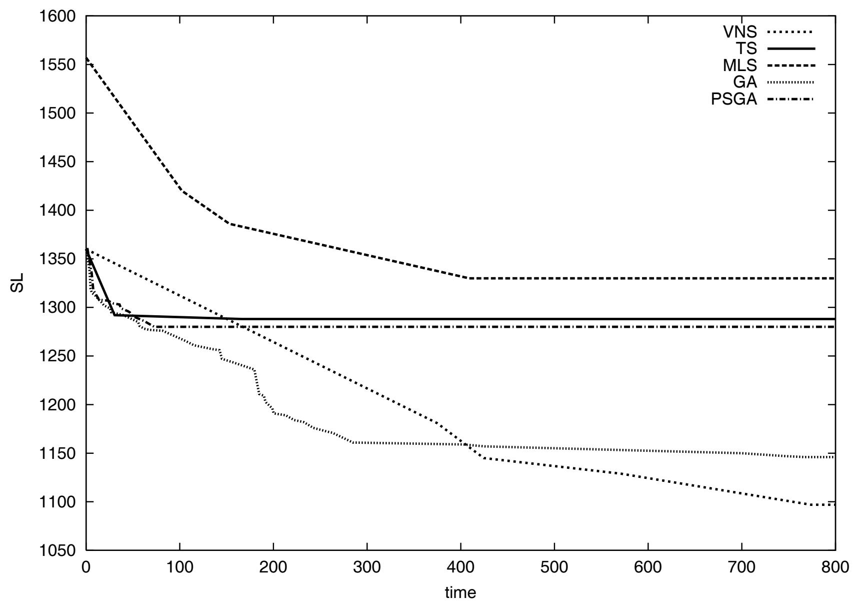


Figure 5: Comparison of heuristic methods for first type task graph with $n = 200$

For each n we generated ten task graphs with different edge densities ρ (see T. Davidović and T. G. Crainic (2003)). Scheduling results are given in Table 3. Each row of this table shows the average results for ten test instances of the same size. The corresponding CPU times spent by each heuristic for finding the best solution are given in Table 4, as well as the maximum allowed CPU time (equal to the time spent for 600 generations of GA) given in the second column.

As in the previous case, we selected $n = 200$, $\rho = 50\%$, and presented the improvements in the time of the global minimal solution for all heuristic methods on Figure 6. We can see that the curves are not smooth as in the previous case. The convergence is rapid at the beginning, and then it is almost impossible to improve the solution anymore. Moreover, we can notice that the methods behave differently for the two types of task graphs. MLS is the worst method for the first type of task graphs, while it performs quite well for the task graphs with known optimal solutions. On the other hand, GA's performance is better for the first type task graphs.

Tables 1 and 3 show that PSGA has the worst performance. This observation coincides with previously published results (T. Davidović and N. Mladenović (2001), Y.-K. Kwok and I. Ahmad (1997)). The explanation lies in the solution representation. The priorities are not sufficient to uniquely define a scheduling order of tasks, implying that additional search is needed in each scheduling step. This means that PSGA spends more time for evaluation of the objective function value than the other heuristics, so less time remains for genetic operations. We proved that fact by letting PSGA run for 600 generations and

Table 3: The Multiprocessor scheduling on random test instances with known optimal solution.

		% Deviation						
n	f_{opt}	CP	LS	VNS	TS	MLS	GA	PSGA
50	600	48.2	22.3	2.42	8.3	5.32	15.7	12.43
100	800	57.54	36.22	10.14	20.79	14.35	25.46	35.37
150	1000	74.7	38.96	11.47	24.91	19.5	44.33	42.5
200	1200	86.97	59.78	24.5	38.37	21.18	49.47	35.95
250	1400	78.48	61.93	31.05	55.1	37.03	65.69	46.53
300	1600	82.82	66.17	53.43	64.23	51.33	72.92	78.99
350	1800	82.24	61.15	35.51	55.59	36.04	63.52	72.88
400	2000	83.66	80.96	47.21	69.88	57.03	26.37	77.99
450	2200	85.36	58.85	31.76	55.6	45.59	30.14	56.69
500	2400	84.87	38.59	37.45	37.41	52.98	76.96	69.76
Av.	1500.00	76.49	52.49	28.5	43.02	34.03	47.06	52.91

Table 4: The CPU Time for scheduling random test instances with known optimal solution.

	CPU	*Time (seconds)*				
n	(max.)	VNS	TS	MLS	GA	PSGA
50	41.0	9.98	6.72	11.13	6.81	12.42
100	180.0	88.40	43.58	76.62	55.9	89.15
150	430.0	231.9	315.1	193.27	162.28	443.67
200	750.0	461.58	539.15	297.02	353.1	777.99
250	1200.0	864.12	945.21	702.68	556.44	1059.78
300	1600.0	1106.97	1042.15	852.52	837.58	744.00
350	2300.0	1709.71	1549.62	1425.41	865.38	1180.29
400	3000.0	2352.4	722.51	1833.1	1500.27	2069.75
450	3500.0	2956.76	1260.39	1527.29	1871.9	1597.36
500	4500.0	1992.32	3334.33	3404.45	2649.23	2861.92
Av.	1750.1	1177.41	975.88	1032.35	885.89	1083.63

it required a several times longer time than GA. Moreover, results were still worse than those obtained by the GA and other heuristics. This argument and the different solution representation (arrays of task priorities vs. permutations of tasks) are sufficient reasons to exclude PSGA from further consideration.

As can be seen from the Table 3, the average schedule lengths were more than 28% (even more than 50% for n=300) above the optimal ones even for the best, VNS obtained, schedules.

To investigate the reason for such high deviations we performed a parametric analysis for the VNS heuristic. The results are reported in the next section.

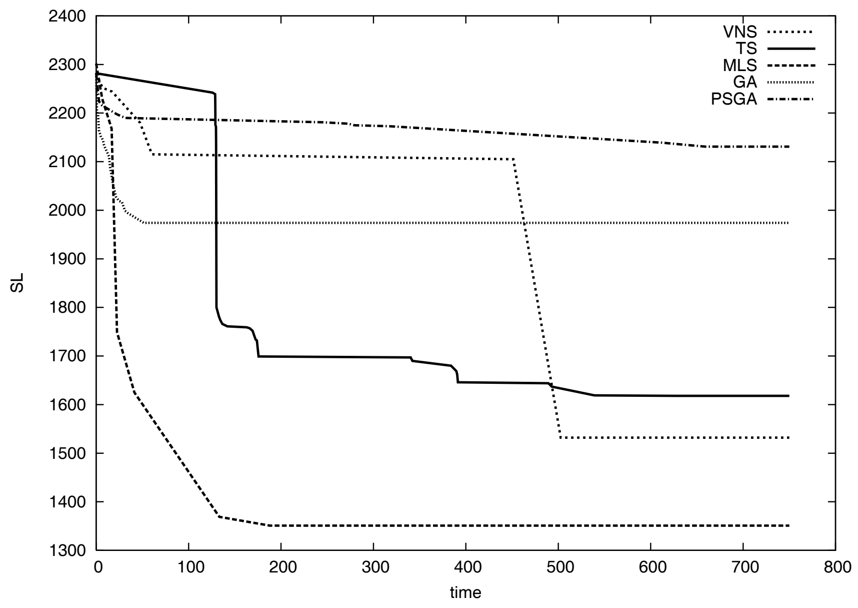


Figure 6: Comparison of heuristic methods for the second type of task graph with n=200 and $\rho = 50\%$

5 Parametric analysis

In this section we describe experiments related to the analysis of the influence of different parameters on the results obtained with permutation based heuristics to MSPCD. We restrict the explanations to the heuristic that performed best in our previous experiments,i.e., VNS. It is straightforward to extend the obtained conclusions to the other heuristics which use a systematic search over preselected neighborhoods. On the contrary, GA exploits a lot of randomness in searching for the best solution, and therefore a different reasoning should be applied.

In Tables 1 and 3 we presented the results obtained by basic variants of all heuristic methods we implemented. Here, we consider only a few possible improvement directions. Obviously, methods can be improved in some other ways, but here, our first intention is to examine the reasons for such the large deviations from the optimum values observed and to propose ideas which might help to overcome that problem.

First, we perform an analysis similar to that one described in T. Davidović and T. G. Crainic (2003), but we also consider the parameters connected to the meta-part of the VNS and other heuristics. Let us set n=200 and describe in details the results for this size of task graphs. For the other values of n, results are similar. In Table 5 are given complete scheduling results for VNS (we presented only the summary line in Table 3). The optimal schedule length (SL_{opt} = 1200 computing cycles) is obtained by scheduling feasible permutation $1, 2, \ldots, 200$ using the ES rule.

Table 5: Complete scheduling results of VNS for random test instances with $n = 200$ and with known optimal solution $SL_{opt} = 1200$.

n	ρ	$S_L(CP)$	$S_L(LS)$	$S_L(VNS)$	t_{min}	t_{max}
200	0	1203	1200	1200	34.43	750.00
200	10	1955	1850	1801	672.01	750.00
200	20	2191	2111	2045	180.50	750.00
200	30	2266	2169	1574	660.43	750.00
200	40	2246	2192	1621	707.49	750.00
200	50	2282	2252	1532	502.69	750.00
200	60	2287	1248	1215	215.51	750.00
200	70	2305	1894	1210	669.56	750.00
200	80	2307	2284	1248	478.36	750.00
200	90	2354	1256	1200	32.33	750.00
av.	45.0	2243.67	1917.33	1494.00	461.58	750.00
% dev.		86.97	59.78	24.5	-	-

Looking at the results shown in Table 5 we can conclude that sparse and dense task graphs are easy for scheduling, while medium density ones (20-60%) present more difficulty. We can give the following explanation for this fact: For sparse task graphs the initial solution is very close to the optimal one (T. Davidović and T. G. Crainic (2003)) and it is not hard for VNS to improve it a little and obtain a good final solution; the initial solution for dense task graphs is not good, but the search space for VNS (set of feasible permutations) is small and again, large improvements, yielding to good final solution are easily obtained. For scheduling task graphs of medium density, VNS starts from an initial solution which is quite far from the optimal one and moreover, the search space is very large. Therefore, improvements are not very substantional.

Next, we focus our experiments on medium density task graphs of second type since they seemed to be "hard" for scheduling and the optimal solutions are known so we could measure the performance of the heuristic solutions. These task graphs were generated in such a way that the first feasible permutation in topological order (i.e. $1, 2, \ldots, n$) is that one yielding to the optimal solution (see T. Davidović and T. G. Crainic (2003), Y.-K. Kwok and I. Ahmad (1997)). Usually, this is not the only "optimal" permutation, but we know that this one certainly is the optimal one. It should be mentioned here that there are task graphs whose optimal solutions cannot be represented by feasible permutation (in the case ES scheduling rule is used, T. Davidović (2000)). Our examples were generated in such a way that a load balance between processors is achieved and no idle time intervals occur in the optimal solution. All processors complete their execution at the same time equal to the length of the optimal schedule (SL_{opt}). The communication is supposed to be intensive between tasks executed on the same processor. The generation algorithm is described in T. Davidović and T. G. Crainic (2003), Y.-K. Kwok and I. Ahmad (1997). An example of optimal schedule is given on Figure 7.

Let us now recall the implementation details in order to summarize the types of analysis that should be performed. In our implementation of heuristics based on LS procedure (TS,

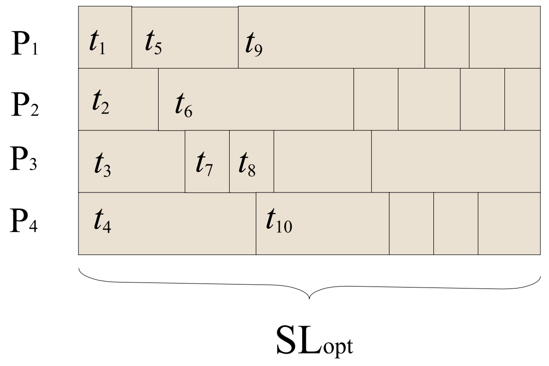


Figure 7: Structure of the optimal schedule for hard test examples

VNS, MLS) we defined the solution space to be the set of all permutation of tasks. Feasible solutions are feasible permutations, i.e. tasks orderings that obey precedence relations between tasks defined by the task graph. Starting from a permutation, the actual solution (schedule, i.e. the index of processor and the starting time instant for each task) can be determined by the use of some scheduling rule. We choose Earliest Start (ES) rule since it has been widely used in the literature. We also tried to use some other scheduling rules such as Idleness Minimization (IM), Preferred Path Selection, PPS (B. A. Malloy, E. L. Lloyd, and M. L. Soffa (1994)) and DeClustering, DC (T. Davidović (2000)), but the ES performed best and in the smallest amount of time as will be discussed later.

After the ES rule is applied, we calculate the objective function value, i.e., schedule length. Obviously, we have an implicit connection between the solution representation and the value of the objective function. The advantage of representing the solution as a feasible permutation is to lead to an efficient neighborhood search, since there are a lot of neighborhood structures that can be applied (Swap-1, Swap-2, ..., IntCh,...). The influence of neighborhood selection is described in Subsection 5.3. The disadvantage of such a representation is that several different permutations may produce the same schedule (after applying ES). An illustration of this observation is given on Figure 8.

To generate this figure, we performed a scheduling procedure for all FORWARD Swap-1 neighbors of known optimal solution for the task graph with $n = 200$ nodes and $\rho = 30\%$. As a result, we obtained 679 optimal schedules versus 437 non-optimal ones. Just for the illustration of neighborhood size, let us compare the number of tasks with the total number of FORWARD Swap-1 neighbors (679+437=1116$\approx 5.5 \times n$). Then, we sorted the so-obtained non-optimal schedules in nondecreasing order, calculated the corresponding deviations from the optimal solution and represented them graphically on Figure 8. As can be seen on Figure 8. there are different permutations that produce the same non-optimal schedule (parts of the graph parallel with the x-axis, not to mention the 679 permutations producing the optimal solution); note also that the worst solution in the Swap-1 neighborhood has a 90% deviation.

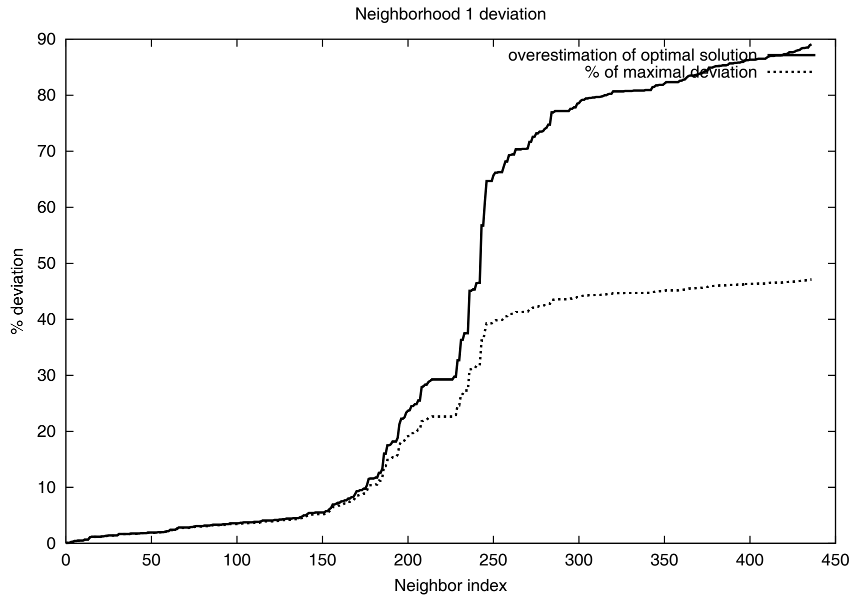


Figure 8: % deviation from the optimal value in the Swap-1 neighborhood of the known optimal solution

From this figure we can also conclude that small changes in the solution representation can produce large deviations in the objective function value. This fact means that we cannot control the search process strictly since the representation is implicit. But, this problem exists even if we look at the exact schedule, i.e., if we use the representation proposed in S.C. Porto and C.C. Ribeiro (1995). For example, let us look at the task graph presented on Figure 1 and its schedule onto a 2 processor system. The CP-based schedule (tasks on CP are allocated to one processor, while all others are scheduled to the second one) is of the length $SL_{CP} = 56$ computing cycles. If we present the solution using lists of tasks allocated to each processor, the CP-based solution would be

P_1: 1, 2, 4, 6, 9, 10
P_2: 3, 7, 5, 8

Let us define a Swap-1 move as moving a task from its original processor to an other one (all others, if there are more processors), and let us consider all such moves that do not violate the precedence relation between tasks. For our example, we come out with 26 neighbors, and a solution deviation range from 0.02% to 41.5% from the best solution in the neighborhood. The deviations could be even larger for the examples with more tasks and incomplete network of processors because of the communication delays that can appear with different coefficients (d_{kl}) in the procedure for calculation of the objective function value. To compare the two representations, we examined the permutation-based one for the example on Figure 1. Starting from the first topological permutation, $1, 2, 3, \ldots, 10$, and searching the Swap-1 neighborhood in the FORWARD direction, we obtained 7 neighbors with the deviations from the best one ranging from 1.88 to 3.77%. In the BACKWARD direction, there are 9 neighbors and maximum deviation is 9.43%. FORWARD search in Swap-1 neighborhood of the CP-based permutation yields 6 neighbors and a maximum deviation of 7.55%, while in the BACKWARD direction the maximum deviation among 9 neighbors is 3.77%.

The difficulties that are produced by communication delays and multiprocessor architecture are illustrated on Figures 9 and 10. Figure 9 contains the same results as those on Figure 8 but for the case when communication delays are neglected.

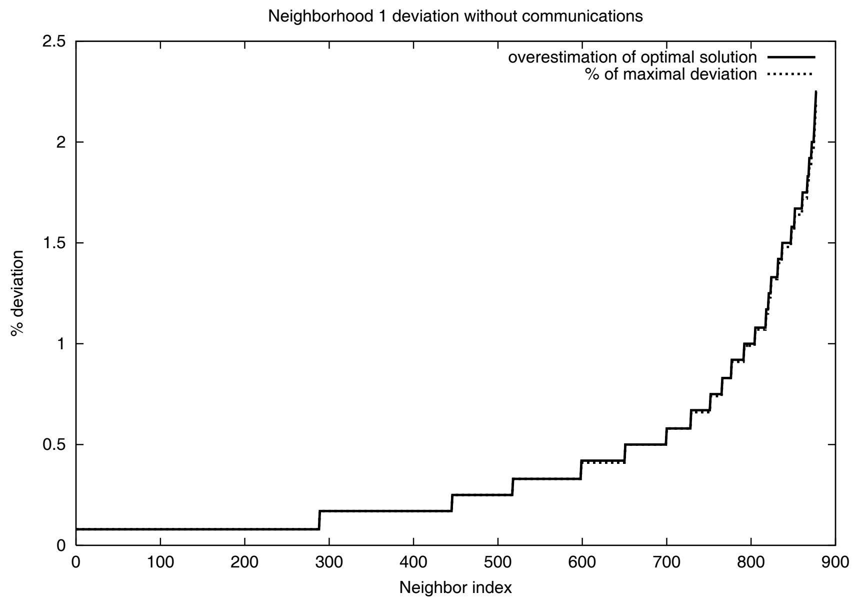


Figure 9: The schedule deviation without communications

We obtained only 237 optimal schedules and 879 non-optimal ones with deviations of non-optimal solutions from the optimal one represented on Figure 9. First, we conclude that in the presence of communication delays more optimal neighbors are found since the tasks are forced to be scheduled in such a way as to avoid expensive communications. Next, it can be seen that the deviation of the non-optimal schedules from the optimal ones is much larger for the case with communication delays. If communication is negligible, the tasks can be packed on processors with less idle time intervals, although optimal schedules seem to be harder to find.

Next, we examined the case when processors are completely connected, i.e. there exists direct a communication link between any two processors. That means

$$d_{kl} = \begin{cases} 0, & k = l, \\ 1, & \text{otherwise} \end{cases}$$

Here, communication time is included. The results of exploration of Swap-1 neighborhood in this case are given on Figure 10. The results are somewhere in between those of the two former cases: The number of optimal schedules is 434 versus 682 non-optimal ones, while the maximum deviation from the optimal solution is 13.83%. The number of optimal schedules is smaller than in the 2-dimensional hypercube case since ES forces some of the tasks to start earlier than would be optimum. This decision is guided by the fact that the distance data have to travel is equal for all transfers. The smaller deviations obtained can be explained by the fact that, thanks to the reduction in data transfer, tasks are packed in a better way, but still, the optimal schedule is hard to find.

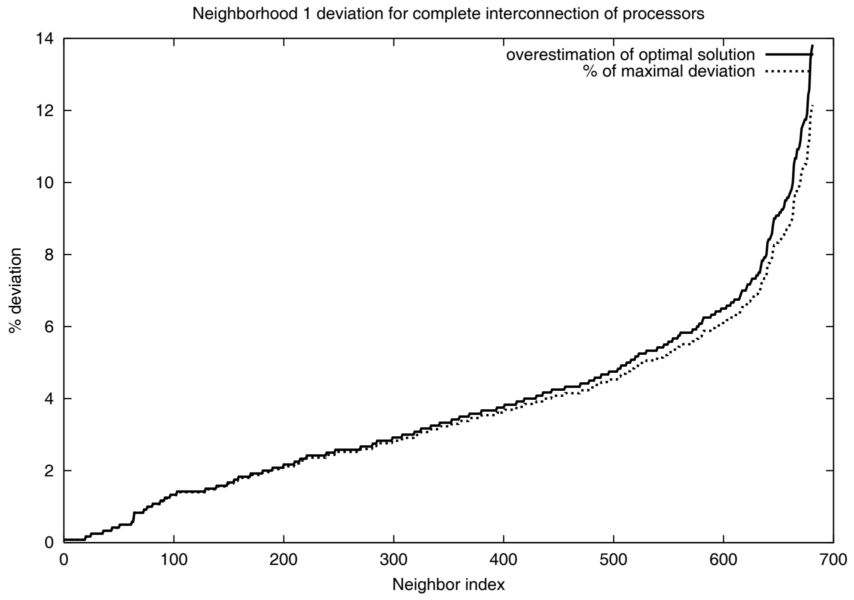


Figure 10: The schedule deviation with communications for complete connection between processors

Here, we were searching the Swap-1 neighborhood in FORWARD direction. The BACKWARD search gives us the following results 609 + 430 = 1039 solutions with a maximum deviation of 89% for scheduling communicating tasks onto a 2-dimensional hypercube (the first number is for optimal solutions). When scheduling dependent tasks without communications the number of optimal solutions was 266 versus 773 non-optimal ones and the maximal deviation of 1.667%, while for the last case we have 429 optimal and 610 non-optimal solutions and a maximum deviation of 6.08%. This means that the results are similar to those obtained in the FORWARD search case.

The fact that there is a large number of optimal solutions in the Swap-1 neighborhood of the known optimal solution, especially for the case we considered initially (incomplete connection of processors and significant communication delays) may seen strange: so many permutations lead to the optimal solution and yet it is so hard to find one by heuristic search. The number of optimal solutions is 1.5 times larger than the number of non-optimal ones. Therefore, we examined the neighborhood 2 of the known optimal solution, i.e. the neighborhood 1 of each neighbor of that solution.

Results obtained are summarized in Table 6. Each value in this table represent the percentage of optimal solutions for different scheduling cases considered also in neighborhood 1.

Table 6: Percentage of optimal solutions in neighborhood 2 of given optimal solution.

	2-hyp. comm.	2-hyp. without comm.	complete with comm.
FORWARD	37.13%	4.94%	17.60%
BACKWARD	34.47%	7.34%	17.11%

The conclusions are similar to those drown in the case of the neighborhood 1. The percentage of optimal solutions decreases, but still, for the most interesting case there are more than 50% permutations that lead to the optimal solution. Examining further, neighborhood 3, 4, . . . , for only the 2-hypercube case with communications in FORWARD directions, we obtained the graph presented on Figure 11 with solid line.

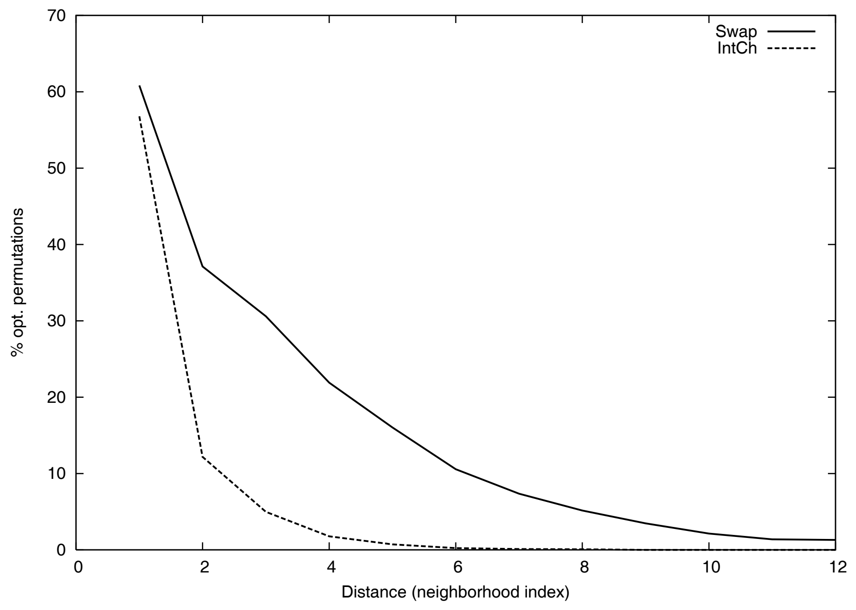


Figure 11: The percentage of optimal permutations in different neighborhoods of given optimal solution

As can be seen from the Figure 11, the optimal solutions are concentrated close to each other. We also examined IntCh neighborhoods and draw the corresponding graph with the dotted line on the Figure 11. For the IntCh neighborhood, the conclusion is that the number of optimal permutations is smaller (as well as the neighborhood size) and they are located even closer to each other since the difference between two neighbors is larger than within the Swap-1 neighborhood.

Since the optimal solutions are so close to each other it is obvious that the VNS concentrates its search in the regions that are far from those containing these optimal solutions. In addition, we definitely cannot search the whole space and therefore we do not know if maybe some other regions contains the group of optimal solutions. Anyway, it is necessary to diversify the search in order to cover larger parts of the solution space.

In the text to follow, we describe the experiments we performed trying to improve the solutions obtained by the basic variants of VNS heuristics. We can distinguish different parameters that can influence the quality of heuristic solution. For example, the search direction (FORWARD-BACKWARD) can be seen as a parameter, but in most of the cases the forward search performed better. We also experimented with best and first improvement search strategies and conclude that first improvement is faster and gives us better results in average, which confirms the result from P. Hansen and N. Mladenović (1999)a.

The best initial schedule does not necessarily lead to the best final heuristic solution. To examine the influence of the initial solution selection on the quality of the minimal schedule obtained we generated initial solutions by the use of several different strategies and performed the VNS heuristic for the task graphs with known optimal solutions and $n = 200$ within given CPU time. A CP initial solution, which is based on the Critical Path rule, is used in all previous experiments. Largest Processing Time (LPT) is the order of tasks obtained when LPT rule is combined with the precedence relation. Directed by the precedence relation, the task order that forces tasks with the largest number of successors we call SUCC initial solution. Similarly, the initial solution that we call COMM is obtained by forcing tasks that require largest amount of communication, i.e., the task priority is defined by the sum of all communications (with both, predecessors and successors). The last initial solution we considered was the RND, the randomly generated one. According to the previous experience, we did not perform as detailed experiments as before. We just combined best variants of Swap-1 LS with different initial solutions. Since no improvement (comparing to the CP initial solution) occurred, we do not present results here and pass to the next step of our analysis, i.e., examination of the influence of the scheduling rule.

As we already mentioned, the number of existing constructive scheduling heuristics is very large, and, moreover, from most of them, we can extract the scheduling rule and apply it to the list of tasks defined by a feasible permutation.

In this work we apply several different scheduling rules during the LS procedure of VNS heuristic. The first of them is Idleness Minimization (IM), guided by the following idea: we know that there is no idle time intervals in the optimal solution, therefore we decide not to force the processor at which task will start earlier but the one where minimal (preferably none) idle time interval is produced. The second scheduling rule we tried is Preferred Path Selection, PPS from B. A. Malloy, E. L. Lloyd, and M. L. Soffa (1994), where depth-first scheduling is performed in such a way that the task from the highest level together with one whole path through the task graph is associated to the same processor. In the original heuristic the task selection is strictly defined based on the CP-priorities and the number of predecessors already associated to the given processor. Here, we adopted this method to explore task selection based on feasible permutation. The final scheduling rule we investigate in this paper is DeClustering, DC (T. Davidović (2000)) which was initially designed to work with feasible permutations. The DC heuristic is based on improvements of a sequential execution defined by a given permutation. These improvements are obtained by moving tasks from the first processor to the others as long as the schedule length decreases.

It appears that the best performing scheduling rule is ES. This does not mean that a better one cannot be found, but it is beyond the scope of this paper to search for the "best scheduling rule".

In the following subsections we investigate the influence of some other search parameters (maximal number of neighborhoods and the related parameters, neighborhood definitions, i.e. combinations and restrictions, stopping criterion) on the deviation of the heuristic solution from the optimal one.

5.1 The influence of k_{max}, k_{step}, *plateaux*

According to the previous analysis, increasing k_{max} is an elementary way to diversify the searching process. The results for different values of k_{max}, constant ones as well as ones depending upon the number of tasks, are given in Table 7.

Table 7: Scheduling results with different k_{max}

n	ρ	$k_{max} = 10$	$k_{max} = 20$	$k_{max} = 50 = n/4$	$k_{max} = n/2$	$k_{max} = 3n/4$
200	0	1200	1200	1200	1200	1200
200	10	1759	1801	1794	1794	1794
200	20	2045	2045	2045	2045	2045
200	30	1692	1574	2066	2117	2117
200	40	1716	1621	1631	1631	1631
200	50	1656	1532	1271	1271	1271
200	60	1224	1215	1215	1215	1215
200	70	1230	1210	1210	1210	1210
200	80	1248	1248	1248	1248	1248
200	90	1200	1200	1200	1200	1200
av.	45.0	1497.0	1494.00	1488.0	1493.18	1493.18
% dev.		24.75	24.5	24.0	24.43	24.43

From this table, we can conclude that the results do not depend much an the value for k_{max}. This is because of the FI philosophy of the VNS heuristic: as soon as the better solution is found, the search continues in $\mathcal{N}_1$. That conclusion can be supported by the fact that k_{max} is reached more than 5 times if it equals 10 and only once (or even not reached at all) for the values $n/4$, $n/2$ and $3n/4$. One may try to overcome this problem by introducing the k_{step} parameter and consequently, by skipping some neighborhoods during the search, but then there is a possibility to lose some good solutions in the neighborhoods that are not visited. Another parameter that could influence the search directions is *plateaux*, i.e., a probability to accept a solution with the same scheduling length as the current incumbent to be the new best solution. As we already noticed, there are different feasible permutations yielding to schedules with the same length. If we change the current best (with some probability) we may diversify the search and explore different regions. We experimented with values $k_{step} = k_{max}/10$, $k_{step} = k_{max}/5$, and $plateaux = 0.5$, $plateaux = 1.0$. The scheduling results for different combination of the three parameters are summarized in the Table 8.

Table 8: Scheduling results with different combination of k_{max}, k_{step}, and *plateaux*

	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
$k_{max} \quad \backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
10*	24.75	27.31	21.52	28.69	27.76	24.07	25.91	30.36	28.92
20	24.5	30.69	28.89	29.24	35.39	26.24	23.46	30.43	26.08
50	24.0	23.95	20.34	29.32	27.31	23.11	24.07	18.55	18.48
100	24.43	23.29	22.78	29.32	21.27	22.66	24.29	21.92	24.06
150	24.43	19.67	22.59	29.32	19.28	23.27	23.83	22.96	22.13

* for this value of k_{max}, k_{step} values are 1, $k_{max}/5$, and $k_{max}/2$.

The results show quite a chaotic behavior of the scheduling process, although usually, it is better if we skip some neighborhood and explore different regions.

On the other hand, each time the neighborhood k is visited, only one solution generated by the shaking procedure is explored. In the next subsection experiments with different shaking procedures (neighborhoods and solution generation process) are described.

5.2 Shaking rule

The second simplest way to assure diversification of the VNS search is to redefine neighborhoods for the shaking procedure. If the IntCh neighborhood is used, the changes in shaken solutions are larger and may lead to different regions. Moreover, we tried the rules proposed by K. Fleszar and K. S. Hindi (2001): move all tasks independently from the chosen one to the left (right), H_l (H_r) shake for short. This means that for the chosen task only its predecessors (successors) are before (after) it in a feasible permutation.

The shaking procedure of the VNS heuristic of T. Davidović, P. Hansen, and N. Mladenović (2001)b, T. Davidović, P. Hansen, and N. Mladenović (2001)a is realized in two variants: 1) k-Swap, meaning that k times we pick up a task and change its position in the feasible permutation and 2) k-IntCh, which consists in k interchanges performed on the current incumbent solution. Another way to change the shake procedure (P. Hansen and N. Mladenović (2001)a, P. Hansen and N. Mladenović (2003)) is to introduce a new parameter b, to select b random solutions in the neighborhood k, and to choose the best one (or one of the several bests) to be the new initial solution for the LS procedure. We experimented with different values for b, and combined these values with the different values of the other parameters. The results are summarized in Table 9.

Table 9: Scheduling results with different shaking rules

	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
Shake IntCh									
50	24.21	25.53	23.33	26.73	29.29	22.67	27.07	24.04	20.74
100	26.12	19.12	31.35	26.73	22.42	23.55	27.06	24.21	22.2
150	26.12	21.44	24.78	26.73	18.68	25.67	27.06	16.66	24.62
Shake Hl									
100	23.02	23.84	26.08	26.53	23.70	25.57	27.90	25.68	26.64
150	23.02	31.27	29.14	26.53	31.78	30.72	27.90	31.87	24.32
Shake Hr									
100	22.48	26.52	25.51	30.27	25.78	25.4	31.78	26.54	22.80
150	22.48	27.35	25.07	30.27	27.52	25.24	31.78	24.17	25.24
Shake with $b = 5$									
50	33.10	30.54	22.07	24.16	28.76	35.07	34.72	27.34	27.88
100	33.01	25.49	25.49	24.16	19.54	25.99	34.83	18.12	25.69
100	33.01	21.97	28.48	24.16	18.73	26.74	34.83	19.22	24.75
Shake with $b = 10$									
50	37.63	27.51	31.68	36.12	29.86	28.72	38.18	32.52	26.12
100	37.68	27.3	19.88	36.12	29.47	20.37	38.18	25.27	24.62
150	37.68	23.76	24.92	36.12	27.89	24.37	38.18	25.44	21.28
Shake with $b = 20$									
50	27.45	35.52	32.37	29.13	24.46	27.27	27.32	25.00	37.53
100	27.72	22.24	32.93	29.19	28.43	23.67	27.32	21.74	26.18
150	27.86	24.67	23.02	29.65	22.47	23.84	27.17	26.22	19.35

These results show that deviations from the optimal solutions are still large and that the search cannot be directed by the parameters tested so far.

5.3 Neighborhood definitions

All these obvious methods to change the VNS procedure did not yield significant improvements. The main reason seems to be the large solution space that has to be searched during the LS procedure. Moreover, each time a new value for the scheduling length (objective function value) has to be determined. Our first try is to redefine the neighborhood for the LS procedure by considering the Swap-321, i.e., we apply a Variable Neighborhood Descent (VND) approach within VNS (in step 2b of VNS we substituted LS with VND). This neighborhood, although larger than the previous one, may diversify the search to the regions far from the current incumbent. Since the FI search is applied, we hope to improve the solution by coarse moves, and then to refine the search in the closest neighborhoods of the newly found best solution. In Table 10 we summarize the results of VNS-VND combination as well as the results obtained by combining VNS-VND with VNS-IntCh and search in both directions.

Table 10: Scheduling results with different neighborhood combinations

VNS-VND									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} \ k_{step}	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
100	28.47	23.82	22.45	25.85	20.77	21.92	28.05	21.07	25.21
150	28.47	21.16	21.46	25.85	16.69	23.37	28.05	17.31	19.75
VNS-VND-IntCh-FB									
100	41.00	19.07	21.22	34.56	23.08	23.66	37.57	25.5	24.39
150	38.87	25.43	28.45	34.72	24.91	21.73	36.6	30.38	30.28
VNS-VND-IntCh-FB, BI search									
100	34.16	27.60	24.65	31.72	35.10	24.42	32.73	24.64	31.20
150	33.95	26.72	23.12	31.53	27.67	30.45	32.72	29.42	29.57
VNS-VND-IntCh-FB, complete interconnection between processors									
100	6.69	7.17	6.87	6.64	7.23	7.16	6.70	6.67	7.16
150	6.75	6.47	7.07	6.64	6.55	7.28	6.70	6.61	7.05
VNS-VND-IntCh-FB, complete connection, neglected communication time									
100	0.34	0.52	0.53	0.39	0.53	0.55	0.44	0.54	0.57
150	0.34	0.53	0.61	0.36	0.52	0.51	0.45	0.52	0.53

As can be seen from this table the deviations are still above 20%. Even if the IntCh neighborhood is introduced and the LS procedure is performed in both directions. Looking at the output of the search process, we can conclude that this huge solution space cannot be explored efficiently. The reason lies in the fact that FI search in large neighborhoods leads to the "random walk" behavior of the search procedure. On the other hand, the greedy BI search always leads to worse final solution, as can be seen in the third part of Table 10.

On the contrary, for the simpler cases, i.e. 1) when processors are completely connected and 2) when communication is neglected the solution deviation is significantly reduced (last two parts of the Table 10). This is an illustration of the original problem difficulty, but still, the known optimal solution cannot be reached.

According to the large size of all these neighborhoods, our next step was to restrict them in some constructive way. The main idea is to force only the "promising" moves. Therefore, we defined several types of restricted neighborhoods. The first of them is to move only the tasks allocated to the processor which defines the value of objective function, i.e. the latest one to complete its execution. We record all the tasks that should be moved and explore restricted Swap-1 (RPVNS) or Swap-321 (RPVND) neighborhoods in both directions (neighbor duplications are not too large in this restricted case). The results presented in the first 6 rows of Table 11, show that the simpler version (RPVNS) performs better and improves previous results to 16-20% deviations.

The second idea is to move the predecessors of heavily communicating task. First, we calculate all the communications that each task requires in a given schedule with both predecessors and successors and then, mark for moving only the predecessors of the task with the largest amount of required communications. In other words, in order to prevent this large number of communication, we try to avoid it by rearranging the task predecessors. We may choose the successors, but it will not influence the first part of scheduling. On the other hand, changing the order of predecessors has an impact on the reminding part of the schedule. Here again, marked tasks are moving in both directions. The results are presented in the next 3 rows of Table 11 (RCVNS). We also experimented with different value of parameter b in the shaking procedure (fourth and fifth part of Table 11). As can be seen from this table the results are still not significantly better. Therefore, we added another restricted case inspired by the decomposition. The first such restricted neighborhood is obtained by selecting randomly p_j, $j = 1, 2, 3, \dots, p$ processors in each "step" and select only tasks allocated to these processors. Here "step" means a single iteration of LS procedure.

The second variant is obtained by random selection of n_i, $i = 1, 2, 3, \dots, n$ tasks to be moved in each step. Scheduling results for these variants are given in the last two groups, containing 3 rows each, of Table 11 (RVNDS1 and RVNDS2). They both improve the results of the basic variant of the VNS heuristic, but summing up all the results it turns out that RCVNS performs best.

5.4 Stopping criterion

For the stopping criterion, we can choose among several options: number of GA iterations, number of MLS restarts, number of unimproved VNS iterations, time limit. According to the different character of these heuristics, the only fair criterion is time limit. In the previous experiments we used a time limit defined by the 600 generations of basic GA. That seemed to be a fair criterion since the fixed number of permutation is generated and rescheduled in each generation. If we would use the restarts of MLS instead, the number of neighborhood visited could be different because of the FI strategy and the number of LS iteration in each restart which is not constant. To investigate the behavior of VNS we let its basic variant (VNS, Swap-1, FI, FOR, $k_{max} = n/2$, $k_{step} = k_{max}/10$) run until 5 unimproved iterations, i.e. until all neighborhoods (from 1 to k_{max}) are explored 5 times without any improvement of the current incumbent. The average resulting CPU time is 14038.64 *sec* (half of that was unproductive according to the new stopping criterion) and the average value over 10 examples for obtained SL is 1318.5, which corresponds to a 9.9% deviation. Within the same CPU time, MLS obtained a deviation of 13.96%, while the deviation produced by GA and TS were of 33.73% and 34.67% respectively.

The best restricted VNS variant (RCVNS) is faster, and therefore we let it work until 50 unimproved iteration. Then the obtained deviation is 12.28% for 1150.71 *sec* CPU time with the total working CPU time equal 4192.62 *sec*, all in average for 10 examples with 200 tasks. Within previously defined time (14038.64 *sec*), the result shows 10.45% deviation obtained for 6513.88 *sec* average CPU time.

Table 11: Scheduling results with different restricted neighborhoods

RPVNS									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	15.23	19.85	21.45	17.83	18.82	16.20	16.32	16.93	17.15
100	16.75	19.23	24.87	17.12	17.12	20.53	16.07	16.48	16.06
150	16.87	16.61	19.53	18.20	17.63	21.99	16.95	16.52	21.56
RPVND									
	plateaux=0.0			*plateaux*=0.5			*plataux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	18.57	23.08	26.57	20.54	23.13	23.12	22.92	24.42	24.41
100	20.46	21.19	26.70	18.92	20.54	24.30	26.47	18.34	23.44
150	25.88	21.39	23.81	20.02	27.10	22.88	27.41	22.03	31.80
RCVNS									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	18.61	17.02	16.63	16.73	16.41	18.79	18.49	16.61	18.65
100	18.47	13.94	17.17	16.84	14.97	16.83	18.11	13.89	16.30
150	18.59	16.43	15.64	16.96	16.96	15.12	18.24	16.84	15.27
RCVNS, shake with $b = 5$									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	18.36	16.96	18.24	17.96	15.82	15.42	17.59	17.32	16.65
100	16.73	16.14	14.97	17.72	16.41	15.47	15.44	16.22	15.49
150	16.73	16.33	16.77	17.79	16.39	15.04	15.44	16.98	15.77
RCVNS, shake with $b = 10$									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	17.18	17.38	16.74	16.87	16.89	17.37	17.2	16.7	17.65
100	17.47	16.54	18.32	17.43	16.19	19.00	17.43	16.94	17.39
150	17.33	16.35	13.80	17.46	16.22	13.29	17.43	16.22	14.57
RVNDS1									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	22.20	16.53	16.62	21.56	20.17	23.97	20.55	21.91	21.92
100	21.12	14.02	21.55	22.37	19.58	20.74	23.59	20.63	18.84
150	25.56	24.59	23.49	19.46	22.57	22.89	18.47	24.19	22.32
RVNDS2									
	plateaux=0.0			*plateaux*=0.5			*plateaux*=1.0		
k_{max} $\backslash k_{step}$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$	1	$k_{max}/10$	$k_{max}/5$
50	20.09	19.69	33.17	19.57	20.73	23.50	23.67	23.50	23.86
100	21.36	20.15	21.64	22.53	20.66	19.38	15.66	21.91	18.66
150	23.54	18.41	17.12	21.32	20.27	25.34	19.67	21.02	26.46

6 Conclusion

In this paper we developed several heuristic methods for solving Multiprocessor Scheduling Problem with Communication Delays (MSPCD) within the framework of well-known metaheuristics. All of them are based on a permutation representation of solutions. Here, we analyzed the performance of this representation as well as the implementation parameters of the proposed methods. Some improvements of the basic variants of the heuristics are proposed based on this analysis in order to direct the search towards known optimal solutions of benchmark examples. The analysis shows that MSPCD is a very difficult combinatorial problem. Metaheuristic approaches, that can significantly improve an initial solution or even that one obtained after the local search (LS) procedure are applied. Still, for the given set of benchmarks with known optimal solution it is hard to obtain these solutions with the current implementation of heuristic methods. The reasons are a large solution space, a large number of "similar" solutions yielding the same objective function value(plateaux problem), solution symmetry as a consequence of processor numeration (which cannot be easily broken). Further improvement directions could be parallelization, possible decomposition of the scheduling problem or different solution representations. A permutation-based solution representation has many advantages: it is easy to manipulate and store in computer memory (a solution update requires $O(1)$ computing steps, all the solutions have the same structure), a large number of possible neighborhoods can be defined. The main disadvantage is indirect connection between solution and value of the objective function. The problem is that this disadvantage can not be easily overcome by changing the solution representation since in both cases rescheduling must be performed due to the change in values for communication delays depending upon the target multiprocessor architecture.

Acknowledgments

This project was partially supported by NSF of Serbia, grant no. 1583, under project "Mathematical optimization models and methods with applications".

References

I. Ahmad and M. K. Dhodhi, (1996), Multiprocessor scheduling in a genetic paradigm, *Parallel Computing*, 22:395–406.

J. Blazewicz, M. Drozdowski, and K. Ecker, (2000), Management of resources in parallel systems, In J. *et al.* Blazewicz, editor, *Handbook on Parallel and Distributed Processing*, pages 263–341. Springer, Berlin.

P. Brucker, (1998), *Scheduling Algorithms*, Springer, Berlin.

B. Chen, (1993), A note on lpt scheduling, *Operations Research Letters*, 14:139–142.

T. Davidović, (2000), Exaustive list–scheduling heuristic for dense task graphs, *YUJOR*, 10(1):123–136.

T. Davidović and T. G. Crainic, (2003), New benchmarks for static task scheduling on homogeneous multiprocessor systems with communication delays, *Centre de Recherche sur les Transports, Technical report*, CRT-2003-04.

T. Davidović, P. Hansen, and N. Mladenović, (2001)b, Scheduling by vns: Experimental analysis, In *Proc. Yug. Symp. on Oper. Res., SYM-OP-IS 2001*, pages 319–322, Beograd.

T. Davidović, P. Hansen, and N. Mladenović, (2001)a, Variable neighborhood search for multiprocessor scheduling problem with communication delays, In *Proc. MIC'2001,* 4^th *Metaheuristic International Conference*, pages 737–741, Porto, Portugal.

T. Davidović, N. Maculan, and N. Mladenović, (2003), Mathematical programming formulation for the multiprocessor scheduling problem with communication delays, In *Proc Yug. Symp. on Oper. Res., SYM-OP-IS 2003*, Herceg-Novi.

T. Davidović and N. Mladenović, (2001), Genetic algorithms for multiprocessor scheduling problem with communication delays, In *Proc. 10. Congress of Yugoslav Mathematicians*, Beograd.

G. Djordjević and M. Tošić, (1996), A compile-time scheduling heuristic for multiprocessor architectures, *The Computer Journal*, 39(8):663–674.

M. Drozdowski, (1996), Scheduling multiprocessor tasks - an overview, *European Journal of Operational Research*, 94:215–230.

K. Fleszar and K. S. Hindi, (2001), Solving the resource-constrained project scheduling problem by a variable neighborhood search, *European Journal of Operational Research*, accepted for publication.

F. Glover and M. Laguna, (1997), *Tabu Search*, Kluwer Academic Publishers.

D. E. Goldberg, (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publ. Comp., Inc.

P. Hansen and N. Mladenović, (1999), First improvement may be better than best improvement: an empirical study, Technical report, GERAD report G-99-54.

P. Hansen and N. Mladenović, (1999), An introduction to variable neighborhood search, In S. *et al.* Voss, editor, *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Dordrecht.

P. Hansen and N. Mladenović, (2001)b, Fundaments of variable neighborhood search, In *Proc. 10. Congr. Yugoslav Mathematicians*, Beograd.

P. Hansen and N. Mladenović, (2001)a, Variable neighborhood search: Principles and applications, *European J. of Oper. Res.*, 130:449–467.

P. Hansen and N. Mladenović, (2002), Developments of the variable neighborhood search, In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers.

P. Hansen and N. Mladenović, (2003), Variable neighborhood search, In F. Glover and G. Kochenagen, editors, *State-of-the-art Handbook of Metaheuristics.* Kluwer, Dordrecht.

E. S. H. Hou, N. Ansari, and H. Ren, (1994), A genetic algorithm for multiprocessor scheduling, *IEEE Trans. on Parallel and Distributed Systems*, 5(2):113–120.

V. Krishnamoorthy and K. Efe, (1996), Task scheduling with and without communication delays: A unified approach, *European Journal of Operational Research*, 89:366–379.

Y.-K. Kwok and I. Ahmad, (1997), Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, *J. Parallel and Distributed Computing*, 47:58–77.

B. A. Malloy, E. L. Lloyd, and M. L. Soffa, (1994), Scheduling DAG's for asynchronous multiprocessor execution, *IEEE Trans. on Parallel and Distributed Systems*, 5(5):498–508.

D. A. Menascé and S. C. S. Porto, (1992), Processor assignment in heterogeneous parallel architectures, In *Proc. of the IEEE Int. Parallel Processing Symposium*, pages 186–191, Beverly Hills.

N. Mladenović and P. Hansen, (1997), Variable neighborhood search, *Computers and Operations Research*, 24(11):1097–1100.

I. Or, (1976), *Traveling salesman - type combinatorial problems and their relation to the logistics of regional blood banking*, PhD thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University.

S.C. Porto and C.C. Ribeiro, (1995), A tabu search approach to task scheduling on heterogeneous processors under precedence constraints, *Int. J. High-Speed Computing*, 7:45–71.

S.C. Porto and C.C. Ribeiro, (1996), Parallel tabu search nessage-passing synchronous strategies for task scheduling under precedence constraints, *J. Heuristics*, 1(2):207–223.

G. C. Sih and E. A. Lee, (1993), A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, *IEEE Trans. on Parallel and Distributed Systems*, 4(2):175–187.

A. Thesen, (1998), Design and evaluation of a tabu search algorithm for multiprocessor scheduling, *Journal of Heuristics*, 4(2):141–160.

J. D. Ullman, (1975), NP-complete scheduling problems, *J. Comput. Syst. Sci.*, 10(3):384–393.

T. A. Varvarigou, V. P. Roychowdhury, T. Kailath, and E. Lawler, (1996), Scheduling in and out forests in the presence of communication delays, *IEEE Trans. on Parallel and Distributed Systems*, 7(10):1065–1074.

B. Veltman, B. J. Lageweg, and J. K. Lenstra, (1990), Multiprocessor scheduling with communication delays, *Parallel Computing*, 16:173–182.