[1]MIC'2001 - 4th Metaheuristics International Conference [MIC'2001 - 4th Metaheuristics International Conference]1 [Porto, Portugal, July 16-20, 2001] []Porto, Portugal, July 16-20, 2001 0.4pt

Variable Neighborhood Search for Multiprocessor Scheduling Problem with Communication Delays

Tatjana Davidović [*] Pierre Hansen [†] Nenad Mladenović [* †]

[*] Mathematical Institute, Serbian Academy of Science and Arts,
Kneza Mihaila 35, 11000 Belgrade, Yugoslavia,
Email: {tanjad, nenad}@mi.sanu.ac.yu

[†] GERAD and École des Hautes Études Commerciales
3000 chemin de la Côte-Sainte-Catherine, Montréal H3T 2A7, Canada
Email: {pierreh, nenad}@crt.umontreal.ca

1 Introduction

The multiprocessor Scheduling Problem with Communication Delays (MSPCD) is defined as follows: tasks (or jobs) have to be executed on several processors; we have to find where and when each task will be executed, such that the total completion time is minimum. The duration of each task is known as well as precedence relations among tasks, i.e., which tasks should be completed before some others 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 of parallel tasks among processors with and without communication delays is a NP-hard problem [14]. However, some special cases can be solved in polynomial time [8, 15]. There are many extended and restricted versions of the multiprocessor scheduling model suggested in the literature (see e.g. [4] for a recent survey), but the most studied is the case without precedence relations among tasks and/or without communication delays.

Among classical constructive heuristics, the best known are LPT (Largest-Processing-Time-first) [2] and CP (Critical Path) [12]. Many papers proposing constructive heuristic solutions can be found in the literature [4]. Recently, metaheuristic approaches have appeared as well.

Genetic Algorithms (GA) have been proposed in [1, 7, 9]. The algorithms developed in [1, 7] assume communication time to be negligible and perform scheduling onto a complete crossbar interconnection network of processors. In [1] a genetic algorithm is combined with a list-scheduling heuristic. The population members (chromosomes) are represented by arrays of task priorities. The first member in the initial population is determined by using the CP method in calculating task priorities. The remainder of the chromosomes in the initial population are generated by random perturbation in the priorities (genes) of this first chromosome. In each generation, the list-scheduling heuristic is applied to all of the chromosomes to obtain corresponding cost function values (schedule lengths).

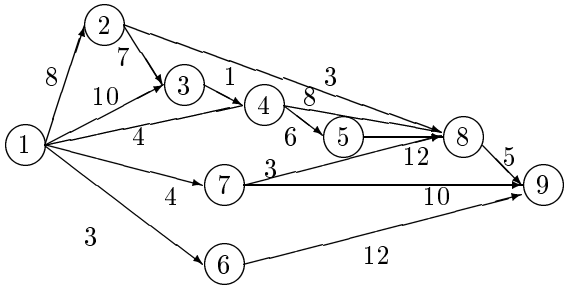
In the parallel GA proposed in [9] MSPCD is considered and members of the population are represented by feasible permutations of tasks.

In [13] Tabu Search (TS) is used to schedule a set of n independent tasks on a complete network of processors. The TS approach is also used in [11] for solving MSPCD on heterogeneous processors.

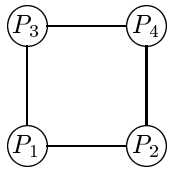
In this paper we develop several heuristics based on the same solution representation for solving MSPCD. The solution is represented by a feasible permutation of tasks, i.e., a permutation obeying dependences between tasks. To obtain the criterion function value, the Earliest Start (ES) scheduling heuristic is used. We develop basic Variable neighborhood search (VNS, [10], [5], [6]), and TS approaches and compare them with each other and with the Multistart Local Search (MLS) and Problem Space Genetic Algorithm (PSGA) proposed in [1] and modified for the MSPCD case. All heuristics are compared within the same CPU time limit on two sets of random task graphs. First type task graphs are arbitrary random graphs with given edge densities, while the second type are task graphs which have known optimal solutions for given multiprocessor architectures.

2 Multiprocessor scheduling problem

The tasks to be scheduled are represented by a directed acyclic graph (DAG) defined by a 4-tuple $\mathcal{G} = (T,\ E,\ C,\ L)$ where $T = \{t_1, \ldots, t_n\}$ denotes the set of tasks; $E = \{e_{ij} \mid t_i,\ t_j \in T\}$ 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 have completed their execution and all relevant data is available. Task preemption and redundant execution are not allowed. An example of task graph is given on Fig. 1a). Node labels represent task numeration, task lengths are given in the corresponding table, while edge labels denote communication costs.



t_i	1	2	3	4	5	6	7	8	9
l_i	60	30	15	40	15	30	35	50	40



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

Figure 1: a) An example of task graph; b) 2-dimensional hypercube multiprocessor architecture.

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 modeled by a *distance matrix* [3]. The element $(i,\ j)$ of the distance matrix $D = [d_{ij}]_{p \times p}$ is equal to the minimum distance between the 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. Fig. 1b) contains the picture of a 2-dimensional hypercube containing $p = 4$ identical processors and the corresponding distance matrix D.

The scheduling of DAG $\mathcal{G}$ onto $\mathcal{M}$ consists of determining the index of the associated processor and starting time instant 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).

3 Variable Neighborhood Search in MSPCD

In order to apply metaheuristics for solving MSPCD we first need to define a solution space S and $X \subseteq S$ a set of so called *feasible* solutions. Let S be a set of all permutations of n tasks, and let x, $(x \in X)$ be a feasible solution (a feasible permutation means that the order of the tasks in that permutation obeys the precedence constraints defined by the task graph: a task cannot appear in a feasible permutation before any of its predecessors or after any of its successors). Having a feasible permutation x, we are able to evaluate the objective function value in a unique way, if we follow always the same rules of assigning tasks to processors (for example, ES rules) in the order given by that permutation (see [1] and [3]). Therefore, the solution set of MSPCD can be represented by S.

By presenting a solution of MSPCD as permutation of tasks, we can use several well known neighborhood structures used in solving the Traveling salesman problem, such as 2-opt, 3-opt, Or-opt etc. Since most of the solutions in the neighborhood defined by k-opt are not feasible, we used 1-Or-opt neighborhood structures in a local search routine in MLS, TS and VNS.

A 1-Or-opt (1-Swap) neighbor of a feasible solution x is defined by moving a single task from one position to another. All such possible replacements of tasks define neighborhood $\mathcal{N}(x)$; a 2-Swap neighbor is obtained by changing positions of two arbitrary tasks (not necessary succeeding ones), etc. To obtain a k-Swap neighbor of a solution x we have to pick up a task and move it from its position to another feasible one, k times.

The next step in developing the heuristics 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.

In all heuristics we start with an initial solution x which can be randomly chosen or determined as the first feasible permutation in topological order, or a permutation obtained by the use of some constructive heuristic: a) nonincreasing critical path priority of the tasks (CP), b) nonincreacing processing time of tasks (LPT), etc. Local search (LS) in such a neighborhood is defined as performing scheduling for all feasible Or-opt neighbors of the given initial solution x and calculating the so-obtained schedule lengths. If a better solution is found we move there and look for improvement in the neighborhood of this new solution again. This process is repeated until there is no better solution in $\mathcal{N}(x)$.

MLS is realized by restarting LS from a random initial feasible permutation until the stopping criterion (number of restarts, CPU time limit) is satisfied and saving the best so obtained local optimum.

To describe the VNS approach ([10], [5], [6]), 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. The main parameter for the VNS is k_{max}–maximum number of neighborhoods. Starting from an initial solution we perform the following steps: 1) shaking in k-th neighborhood, 2) local search and 3) move and update k; until stopping condition is satisfied (for further details on VNS see [10] and for most recent surveys see [5] and [6]).

We also develop a basic TS for solving MSPCD, with the same solution representation and LS procedures. In the TS heuristic we use a variable size 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.

We adapted the PSGA approach developed in [1], to solve the MSPCD by taking into account required communications and multiprocessor architecture.

All these heuristics are compared on random task graphs within the same CPU time limit and results are reported in the next section.

4 Experimental results

We implemented the proposed metaheuristics in C programming language on Intel Celeron processor (468 MHz) with Linux operating system.

To illustrate the efficiency of our approach we tested it on two types of randomly generated task graphs. The first group of task graphs is generated with preselected edge existence density ρ ranging from 0.2 to 0.8. We generated random task graphs with up to $n = 300$ tasks and six different values for communication-to-computation ratio (CCR). Therefore, for each n thirty different task graphs are generated. 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 3-dimensional hypercube ($p = 8$). Task graphs with $\rho = 0.4, 0.5$ are scheduled onto 2-dimensional hypercube (ring of four processors, Fig. 1b)), while dense task graphs ($\rho = 0.6, 0.8$) are scheduled to be executed on $p = 2$ processors. We obtain the following conclusion experimentally: it appears that dense graphs cannot benefit from adding new processors to the multiprocessor system since too much time is spent on data transfer between tasks scheduled to different processors.

We compare the initial solution length (obtained by scheduling permutation defined with task priorities set by the use of the CP heuristic) and LS obtained schedule to the minimum schedule length obtained by the use of VNS, TS, MLS, and PSGA heuristics. Comparative results are presented in Table 1. The first column of this table contains the number of tasks in the tasks graphs, the second one the makespan of the best obtained schedule (in average), while in the remining six columns average values of schedule lengths obtained by the use of CP constructive heuristic, LS and each of the new heuristics are presented. The last four columns contain CPU time spent by each heuristic to find its minimum schedule (in average).

The second type of randomly generated task graphs were graphs with known optimal schedule lenghts (the generation procedure is described in [9]). For these task graphs we set $p = 4$ (processor ring, Fig. 1b)) while for edge density we set $\rho = 0.33$ and $\rho = 0.67$. The number of tasks in second type task graphs is varying from 50 to 500.

Table 2 contains the similar results for sparse task graphs of second type: i.e., with a known optimal solution which is given in second column. For dense task graphs ($\rho = 0.67$) optimal solution was obtained as initial CP heuristic schedule or, at worst, by applying only LS to initial solution.

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

	Best	*% Deviation*						*Time (seconds)*			
n	(av.)	CP	LS	VNS	TS	MLS	PSGA	VNS	TS	MLS	PSGA
50	584.67	3.56	0.34	0.03	0.33	0.18	0.48	0.54	0.99	0.34	1.36
100	1207.73	7.38	1.05	0.07	0.57	0.77	2.03	26.40	22.15	7.32	19.36
200	2435.03	10.68	1.67	0.07	0.93	1.20	4.02	246.11	168.87	80.93	187.34
300	3599.83	14.15	2.37	0.09	1.22	1.85	6.63	1235.44	934.11	439.95	1015.84

It appears that for both types of random test instances VNS performs best. However, the possible reason why the best solution found by VNS is still more than 60% above the optimal one (in Table 2) should be the subject of further investigation.

References

[1] I. Ahmad and M. K. Dhodhi. Multiprocessor scheduling in a genetic paradigm. *Parallel Comput.*, 22:395–406, 1996.

[2] B. Chen. A note on lpt scheduling. *Oper. Res. Lett.*, 14: 139–142, 1993.

Table 2: The Multiprocessor scheduling on random test instances with known optimal solution.

		% Deviation						*Time (seconds)*					
n	f_{opt}	CP	LS	VNS	TS	MLS	PSGA	Init	LS	VNS	TS	MLS	PSGA
50	600	51.67	36.67	25.17	36.67	29.00	34.33	0.00	0.23	5.99	0.05	2.67	8.36
100	800	74.50	65.75	56.88	67.62	66.75	66.88	0.00	5.25	39.36	23.42	40.86	32.05
150	1000	83.50	60.90	10.50	60.90	77.20	78.60	0.01	22.69	346.49	32.03	56.60	33.52
200	1200	86.92	83.25	71.67	83.25	78.33	83.50	0.01	27.60	190.21	16.96	550.84	239.12
250	1400	87.00	86.43	83.50	85.50	84.86	84.29	0.02	19.03	721.65	94.72	297.81	151.14
300	1600	88.62	88.62	81.62	85.50	81.94	88.19	0.03	34.44	1392.77	1667.78	547.53	267.18
350	1800	93.06	88.28	86.78	88.17	83.72	89.39	0.03	176.83	1777.05	226.68	314.64	1128.29
400	2000	92.50	90.90	88.70	90.10	43.50	90.50	0.05	382.83	3635.18	4044.16	4043.25	827.54
450	2200	94.59	92.77	80.04	91.09	91.23	92.64	0.06	377.16	6685.96	2358.65	1002.72	3582.86
500	2400	95.29	90.83	43.50	89.08	90.67	92.08	0.07	479.24	9091.91	7305.25	1242.98	2220.48
Average		84.76	78.44	62.84	77.79	72.72	80.04	0.03	152.53	2382.36	1576.97	809.99	849.05

[3] Tatjana Davidović. Exaustive list–scheduling heuristic for dense task graphs. *Yugoslav J. Oper. Res.*, 10 (1): 123–136, 2000.

[4] M. Drozdowski. Scheduling multiprocessor tasks - an overview. *European J. Oper. Res.*, 94: 215–230, 1996.

[5] P. Hansen and N. Mladenović. An introduction to Variable Neighborhood Search. in S. Voss, editor, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458, Kluwer Academic Publishers, Dordrecht, 1999.

[6] P. Hansen and N. Mladenović. Variable Neighborhood Search: Principles and Applications. *European J. Oper. Res.* 130 (3): 449-467, 2001.

[7] E. S. H. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel and Distrib. Sys.*, 5 (2):113–120, Feb. 1994.

[8] V. Krishnamoorthy and K. Efe. Task scheduling with and without communication delays: A unified approach. *European J. Oper. Res.*, 89: 366–379, 1996.

[9] Y.-K. Kwok and I. Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *J. Parallel and Distrib. Comput.*, 47:58–77, 1997.

[10] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers Oper. Res.*, 24 (11): 1097–1100, 1997.

[11] S.C.S. Porto and C.C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *Int. J. High Speed Comput.*, 7:45–71, 1995.

[12] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel and Distrib. Sys.*, 4(2):175–187, February 1993.

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

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

[15] T. A. Varvarigou, V. P. Roychowdhury, T. Kailath, and E. Lawler. Scheduling in and out forests in the presence of communication delays. *IEEE Trans. Parallel and Distrib. Sys.*, 7(10):1065–1074, Oct. 1996.