Benchmark-Problem Instances for Static Scheduling of Task Graphs with Communication Delays on Homogeneous Multiprocessor Systems

Tatjana Davidović
Mathematical Institute,
Serbian Academy of Science and Arts,
Kneza Mihaila 35, 11000 Belgrade, Serbia and Montenegro
tanjad@crt.umontreal.ca

Teodor Gabriel Crainic
Departement management et technologie,
École des sciences de la gestion,
Université du Québec à Montréal and
Centre de recherche sur les transports, Université de Montréal
theo@crt.umontreal.ca

August 17, 2004

Abstract

Scheduling program tasks on processors is at the core of the efficient use of multiprocessor systems. Most task scheduling problems are known to be NP-Hard and, thus, heuristics are the method of choice in all but the simplest cases. The utilization of acknowledged sets of benchmark-problem instances is essential for the correct comparison and analysis of heuristics. Yet, such sets are not available for several important classes of scheduling problems, including *Multiprocessor Scheduling Problem with Communication Delays* (*MSPCD*) where one is interested in scheduling dependent tasks onto homogeneous multiprocessor systems, with processors connected in an arbitrary way, while explicitly accounting for the time required to transfer data between tasks allocated to different processors. We propose test-problem instances for the *MSPCD* that are representative in terms of number of processors, type of multiprocessor architecture, number of tasks to be scheduled, and task graph characteristics (task execution times, communication costs, and density of dependencies between tasks). Moreover, we define our task-graph generators in a way appropriate to ensure that the corresponding problem instances obey the theoretical principles recently proposed in the literature.

Key words: Multiprocessor systems, task scheduling, communication delays, benchmark-problem instances

Résumé

L'allocation et l'ordonnancement de tâches sur les processeurs d'un ordinateur multi-processeurs est au coeur de l'utilisation efficace de tels systèmes. Les problèmes d'allocation de tâches sont, en géneral, NP-durs, et les méthodes heuristiques sont utilisées presque dans tous les cas. L'évaluation et l'analyse comparative des méthodes heuristiques demandent l'utilisation d'ensembles de problèmes tests reconnus. Malheureusement, de tels ensembles ne sont pas disponibles pour plusieurs classes de problèmes d'allocation et d'ordonnancement, en particulier pour le problème d'allocation de tâches avec delais de communication (*MSPCD*). Nous proposons des ensembles de problèmes tests pour le *MSPCD* qui diffèrent selon le type d'architecture du système, le nombre de processeurs, le nombre de tâches, ainsi que les caractéristiques du graphe telles la densité des liens de communication inter-tâches, la durée des tâches, et ainsi de suite. Les ensembles de problèmes tests proposés obéissent également aux critères théoriques de correctitude proposés récemment dans la litérature.

Mots-clefs : Ordinateurs multi-processeurs, allocation et ordonnancement de tâches, delais de communication, problèmes tests

1 Introduction

Scheduling program modules (tasks) to processors is at the core of the efficient use of multiprocessor systems and is being studied for over forty years now [14]. Most task scheduling problems are known to be NP-Hard [12, 27]. There are only a few special cases that can be solved optimally in polynomial time [4, 16, 28]. For all the other problem variants, heuristics have been mainly proposed [11, 18, 22, 24, 25]. See [2, 23] for recent surveys on these topics.

We are particularly interested in the problem of scheduling dependent tasks onto homogeneous multiprocessor systems with processors connected in an arbitrary way, while explicitly accounting for the time required to transfer data between the tasks allocated to different processors. We refer to this variant of task scheduling problem as the *Multiprocessor Scheduling Problem with Communication Delays* (*MSPCD*).

It is generally difficult to analyze and compare the performance of heuristics, even when they address the same scheduling problem variant. On the one hand, theoretical performance measures are difficult to come by. On the other hand, the experimental analysis is hampered by the fact that most papers propose new test instances and report performance results for these examples only. The utilization of acknowledged sets of benchmark-problem instances (e.g., the OR Library at chttp://mscmga.ms.ic.ac.uk/info.html) to report computational results contributes to alleviate this issue. Such sets of benchmark-problem instances are not available for all classes of scheduling problems, however. Thus, very few benchmark-problem instances have been proposed for task scheduling on multiprocessor systems [5, 20, 26]. Moreover, it turns out that these problems are often addressing particular multiprocessor architectures, do not involve communication delays, and are not very challenging with respect to some other parameters of the problem. There is thus the need for a more complete and challenging set of benchmark-problem instances.

The objective of this paper is to propose benchmark-problem instances for the *MSPCD* that are representative in terms of number of processors, type of multiprocessor architecture, number of tasks to be scheduled, and task graph characteristics (task execution times, communication costs, and density of dependencies between tasks). We show that the problem sets we propose overcome most of the drawbacks displayed by other sets of test problems available in the literature. Moreover, task-graph generators are defined such that the corresponding problem instances obey the principles proposed recently by Hall and Posner [13].

The scheduling heuristics used to evaluate the proposed test-problem instances, as well as those present in the literature and related to the *MSPCD*, provide a fair representation of the main heuristic approaches: breath/depth first search, explicit/implicit treatment of communications, priority/dependency-based selection of tasks, minimization/maximization of the number of processors used, and so on. We may thus not only confirm the well-known fact that there are "easy" and "hard" graphs for each heuristic but, most importantly, to explain how scheduling results depend on the task graph and the characteristics of the multiprocessor system.

To sum up, the main contributions of the paper are: 1) a critical analysis of existing test-problem sets for scheduling problems related to the *MSPCD*; 2) new sets of benchmark-task graphs for the *MSPCD*, a problem variant not yet directly considered in the literature, with parameter variations that ensure a fair evaluation of scheduling algorithms; and 3) the theoretical analysis of the proposed sets of benchmark-problem instances.

The paper is organized as follows. The *MSPCD* scheduling problem is described in Section 2, together with the scheduling heuristics used for experimental evaluations. Section 3 reviews the literature and analyzes the existing sets of test-problem instances, indicating weaknesses and pointing to the need for new problem sets. The benchmark-problem instances that we propose are described in the next section, together with the corresponding task graph generators. Section 5 presents an experimental analysis of these sets. Section 6 concludes the paper.

2 Multiprocessor Scheduling Problem with Communication Delays

In this section, we first recall the graph-based combinatorial formulation of the *MSPCD* problem. We then describe the characteristics of the scheduling heuristics used to evaluate the sets of test-problem instances.

The tasks to be scheduled are represented by a *Directed Acyclic Graph* (DAG) [6, 17, 25] defined by a 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 precedence/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 $c_{ij} \in C$ corresponds to the amount of data transferred between tasks t_i and t_j when executed on different processors. When both tasks are assigned to the same processor, the communication cost equals zero. The set E defines precedence relations between tasks. A task cannot be executed unless all its predecessors have completed their execution and all relevant data is available. Task preemption and redundant executions are not allowed in the problem version considered in this paper.

The multiprocessor system $\mathcal{M}$ is assumed to contain p identical processors with their own local memories. Processors communicate by exchanging messages through bidirectional links of equal capacity. The architecture is modelled by a *distance matrix* [6, 11]. The element $(k,\ l)$ of the distance matrix $D = [d_{kl}]_{p \times p}$ equals the minimum number of links connecting the nodes p_k and p_l. The distance matrix is thus symmetric with zero diagonal elements. We also assume that each processor constituting the multiprocessor system has I/O processing units for all communication links so that computations and communications can be performed simultaneously.

The scheduling of DAG $\mathcal{G}$ onto $\mathcal{M}$ consists in determining a processor index and starting-time instant for each task in $\mathcal{G}$ in such a way as to minimize a given objective function. An often used objective function (that we use in this paper as well) represents the 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 is determined by the completion times of its predecessors and the amount of time needed to transfer the associated data from the processors executing these predecessors to the processor that executes the task t_i. The communication time between tasks t_i and t_j, executed on processors p_k and p_l, respectively, may be calculated as

$$\gamma_{ij}^{kl} = c_{ij} * d_{kl} * ccr,$$

where *ccr* is architecture-dependent and represents the *Communication-to-Computation-Ratio,* defined as the ratio between the transfer time of a unit of data and the time required to perform a single computational operation. (Note that this definition characterizes the multiprocessor architecture; *ccr* is thus different from the CCR parameter of Kwok and Ahmad [19, 20], which is defined as the ratio between the total communication and computation times for the given task graph and thus corresponds to a characterization of the task graph.) When $l = k$, $d_{kl} = 0$ implying that $\gamma_{ij}^{kl} = 0$.

Scheduling problems are NP-Hard in most cases [12, 27] and are usually addressed by using heuristic methods in all but the simplest cases. This is also the case for task-graph scheduling problems in general [11, 18, 22, 24, 25] and, in particular, for the *MSPCD* problem for which a linear programming formulation displays n^6 variables [10].

Most heuristic methods proposed for task-scheduling problems are based on some priority-ordering scheme and include two phases: task ordering and task-to-processor allocation. Heuristic methods may then be classified according to the order of these two phases [15, 23]. A first class of heuristics starts by sorting the tasks according to some priority scheme, followed by the second phase where each task is assigned to the processor which is selected by some scheduling rule. Heuristics that belong to the second class, first assign tasks to processors, while the actual schedule, i.e., the definition of the order among tasks and the starting time for each task, is performed during the second phase. For commodity of presentation, we refer to the two classes as *list-scheduling* and *clustering* heuristics. The difference between the two classes is that the former orders all tasks according to specified priorities prior to assignment, while the latter first generates subsets of tasks and then orders each subset separately.

Both types of algorithms involve heuristic rules for the task ordering and assignment phases. Task priorities may be defined based on task execution times, communication requirements, number of successor tasks, and so on. Several criteria may be combined and considered simultaneously. The heuristic task assignment rule may be breath first, when mutually independent tasks are considered, or depth first in which case one examines tasks along paths in the task graph. Scheduling problems and algorithms may also be distinguished according to the role associated to the number of processors: one may either use all the processors available or attempt to minimize the number of processors used.

We selected a set of heuristics that reflect these methodological approaches and used them to investigate the characteristics of different sets of benchmark-problem instances proposed in the literature as well as our own. Four list-scheduling and two clustering heuristics represent our choice.

The first list-scheduling heuristic is denoted *CPES* and is a modification of the *Dynamic Level Scheduling* (*DLS*) heuristic [25]. The basic idea of *DLS* is to calculate static task priorities based on the *Critical Path* (*CP*) method and use them to order the tasks and then, to modify them during the scheduling phase to reflect the partial schedules already generated (the best (task, processor) pair is selected for scheduling). The dynamic update of priorities improves somewhat the performance of the scheduling phase, but it also increases its computational burden significantly. Therefore, we decided to use only static, *CP*-based priorities. Notice that the calculation of the critical path does not involve communication times, because these depend upon the final task-to-processor assignment and schedule. Thus, only task execution times l_i are considered. An *Earliest Start* (*ES*) heuristic is used for scheduling, meaning that one computes the starting time $st(t_i, p_k)$ for each task t_i and processor p_k, the task being allocated to the processor with the smallest associated starting time value [6].

We also implemented the *DeClustering* (*DC*) list-scheduling algorithm [6]. The first phase is based on a *CP*-based priority list, which defines a sequential order of execution (i.e., the order of tasks when all assigned to a single processor). In the second phase, tasks are moved one by one to the other available processors for as long as it improves the current schedule. This approach proved very efficient for dense task graphs for which most heuristics yield solutions with large number of idle time intervals due to inter-process data exchanges, resulting in longer parallel execution times than the sequential one [6, 15].

The literature hints that the *CP* rule might not be efficient when communication times are involved [2, 11, 15]. Consequently, we also implemented two variants of the previous methods, *LPTES* and *LPTDC*, which make use of the *Largest Processing Time* (*LPT*) [3] rule to build the priority list of the first phase.

The *Preferred Path Selection* (*PPS*) [21] and the *Load Balancing with Minimized Communications* (*LBMC*) [24] are the two clustering heuristics that we implemented. The main idea of the second method is to cluster tasks in the first phase such that processor loads are balanced and inter-processor communications are minimized. The *PPS* method builds paths through the given DAG such that a task is included in the path if at least one of its predecessors already belongs to the path. For a task, the path containing the highest number of predecessors is identified as *preferred*. Preferred paths are then used to determine clusters. Both methods include a second phase to compute the actual schedules on each processor.

Let us summarize here the differences among the selected scheduling algorithms: *CPES* is a greedy breath-first heuristics, while *PPS* performs a depth-first search {or chaining [11]}. *LBMC* explicitly considers the minimization of communications, while *DC* includes communication times to improve upon the sequential schedule. Moreover, while *CPES* and *DC* aim to minimize the number of processors, *LBMC* and *PPS* are using all available processors. Thus, the selected procedures cover the main scheduling heuristic designs and were used in all the experiments reported in this paper.

3 Existing Problem Sets

Most papers describing benchmark-problem instances for scheduling problems consider simpler cases than *MSPCD*, such as DAGs without communication costs or completely connected multiprocessor architectures. Thus, only a few sets of test-problem instances may be found in the literature that are somewhat related to the *MSPCD*. We review these efforts in this section. Our goal is to qualify their usefulness to benchmark heuristics for the MSPCD. We base our evaluation both on experimental results and on the criteria proposed by Hall and Posner [13].

Hall and Posner [13] addressed the issue of the generation of representative test-problem instances for analyzing and comparing heuristic algorithms for scheduling problems. They pointed out the need for sets of test problems that address a given scheduling problem variant but are independent of the problem characteristics, and may thus be used for a fair evaluation of corresponding algorithms. Hall and Posner defined principles that representative sets of test examples should satisfy. They illustrated these principles by describing the generation process of test problems for two scheduling problems: the minimization of the maximum tardiness with release dates, and the minimization of the weighted completion times with deadlines. These problems are more complex than the *MSPCD*. We thus assume that the principles and properties proposed by Hall and Posner hold for this simpler case. The principles that should govern the generation of test-problem instances are: 1) *Purpose*, the generated examples must be functional for the analysis and comparison of algorithms for the given scheduling problem; 2) *Compara-*

bility, examples should not depend on any characteristic of the experimental environment; 3) *Unbiasedness*, the generation process should avoid incorporation of any problem characteristic; and 4) *Reproducibility*, the generation process should be simple and clear so as to be easily repeated for the generation of new adequate data. The generation procedure should display the following properties: 1) *Variety*, the problem generator must create a wide range of problem instances; 2) *Relevance,* i.e., create test instances that model real world situations; 3-4) *Scale* and *size invariance* to ensure that the results will not depend on problem instance size; 5) When identical types of input are treated in a similar manner, the generation procedure satisfies the *regularity* property; 6-8) *Describability, efficiency*, and *parsimony* to guarantee that the generation procedure is easy to describe, easy and efficient to implement, use and replicate and, finally, that only parameters that may affect the analysis are varied. The relations between principles and properties are explained in [13]. In the following, we characterize existing sets of problem instances with respect to them.

Coll, Ribeiro, and de Sousa [5] proposed a set of problems of relatively small sizes with no communication delays. Moreover, scheduling results are available for completely connected heterogeneous multiprocessor systems only. Therefore, these test-problem instances are not appropriate for *MSPCD.*

A similar conclusion is reached relative to the very large set of problem instances proposed by Tobita and Kasahara [26]. The problem-instance generators satisfy most of the principles and properties of [13] (the parsimony criterion does not appear to be strictly enforced). Communication delays and arbitrary multiprocessor architecture were not considered, however, which make these test-problem instances not suitable for *MSPCD.* (Appendix 2 presents scheduling results using the selected heuristics on some of these problem instances.)

According to the authors' knowledge, Kwok and Ahmad [20] proposed the only existing sets of problem instances closely related to the problem addressed in this paper. They considered the communication delays. Multiprocessor architecture was assumed to be completely connected, however.

Kwok and Ahmad [20] collected 15 scheduling algorithms proposed in the literature and generated several sets of test instances to compare them. The authors divided the scheduling algorithms into five somewhat homogeneous groups and compared the algorithms within groups. The five groups of algorithms were classified as *SRC* (*Scheduling Restricted Graphs*), *TDB* (*Task Duplication Based*), *UNC* (*Unbounded Number of Clusters*), *BNP* (*Bounded Number* of (completely connected) *Processors*), and *APN* (*Arbitrary Processor Networks*). *APN* is thus the only group of algorithms that appears suitable for *MSPCD* but, it was not analyzed in details in [20].

Test instances were also partitioned into groups. Not all scheduling algorithms were applied to all groups. The proposed groups of task graphs were *PSG* (*Peer Set Graph* containing small-sized graphs collected from the literature), *RGBOS* (*Random Graphs* with *Branch*-and-bound obtained *Optimal Solutions*), *RGPOS* (*Random Graphs* with *Preselected Optimal Solutions*), *RGNOS* (*Random Graphs* with *Not-known Optimal Solutions*), and *TG* (*Traced Graphs,* representing real-world applications). *PSG* and *TG* contain graphs of particular structure (e.g., triangle– or diamond–like) and their characteristics, scheduling environment, and results were not detailed. Therefore, we focus our analysis on the remaining three groups, *RGBOS*, *RGPOS*, and *RGNOS*, that consider communication delays, even though results were reported for completely connected processor networks only. The set of scheduling algorithms that considers arbitrary processor networks (*APN*) can be applied to these instances. All the scheduling algorithms we implemented can be classified as *APN*, some being similar to those used in [20] (e.g., *DLS* and *DC*).

The *RGBOS* set of test examples contains 36 graphs, 3 graphs with different values for the CCR parameter (0, 1, and 10) for each of the 12 DAG sizes corresponding to a number of tasks n ranging from 10 to 32 by increment of 2. Optimal solutions for these small examples were determined by using an A^* enumeration algorithm [1]. The heuristic results reported in [20] for scheduling these task graphs on a 2-processor system displayed 1% to 8% deviations from the optimum. The results obtained using the heuristics we selected are given in Table 1. *CPES* and *LPTES* always yielded optimal solutions.

The second group of test-problem instances, *RGPOS*, consists of 30 instances with the preselected optimal solution for completely connected multiprocessor systems. For these examples, n ranges from 50 to 500 by an increment of 50 and for each n, the same three values were used for the CCR parameter. Scheduling results were reported for 11 scheduling algorithms with an average deviation from the optimum of 2% to 15%. The optimal solutions were obtained only in a few cases. Surprisingly, in the [20] study, most scheduling algorithms from the *APN* group were not applied to the test instances from the *RGBOS* and *RGPOS* sets. The fact that these sets are not representative for arbitrary multiprocessor architectures does not preclude the utilization of *APN* algorithms. The results of our experiments with the *RGPOS* problem instances are displayed in Table 2. Surprisingly, as previously, the *CPES* and *LPTES* heuristics yielded optimal solutions for **all** problem instances.

The performance of the *CPES* and *LPTES* heuristics cannot be explained by their design or intrinsic qualities. Our experience shows, for example, that the results obtained by *CPES* can deviate by more than 100% from the optimum [8, 9]. We thus analyzed the task graphs in the two sets and noticed that they all display very similar edge (connection) densities. The edge density of a graph, ρ, is defined as the number of edges divided by the number of edges of a completely connected graph with the same number of nodes (i.e., $\rho = |E|/n(n-1)/2$). For *RGBOS*, ρ varies from 30% to 40%, while for graphs in *RGPOS,* the value of ρ is between 4% and 10%. To conclude, both sets contain only

Table 1: Scheduling results for RGBOS problem instances

Graph	p	Opt CPES LPTES	LBMC	DC	LPTDC	PPS
r10_0	2	271	488	352	275	284
r10_1	2	207	281	279	283	289
r10_10	2	266	1342	902	902	1169
r12_0	2	307	522	395	339	414
r12_1	2	246	366	320	356	352
r12_10	2	232	916	780	701	970
r14_0	2	281	428	339	326	399
r14_1	2	271	629	370	368	373
r14_10	2	267	1212	843	882	678
r16_0	2	332	441	485	373	428
r16_1	2	326	620	358	428	489
r16_10	2	416	2175	1040	1109	2517
r18_0	2	420	815	512	481	569
r18_1	2	428	892	476	732	526
r18_10	2	390	1860	1201	1255	2147
r20_0	2	477	962	566	547	579
r20_1	2	378	783	469	521	596
r20_10	2	457	3361	1249	1249	2989
r22_0	2	585	1023	686	708	784
r22_1	2	488	905	488	605	996
r22_10	2	575	2567	1292	1356	3370
r24_0	2	480	882	634	557	625
r24_1	2	618	1276	800	817	961
r24_10	2	594	3843	1349	1372	2691
r26_0	2	737	1319	893	781	934
r26_1	2	614	1097	687	687	1150
r26_10	2	529	909	1128	1196	4287
r28_0	2	680	1269	750	731	958
r28_1	2	671	1194	875	812	1279
r28_10	2	623	4507	1467	1467	3189
r30_0	2	750	1435	849	790	1068
r30_1	2	811	1842	859	1144	1530
r30_10	2	653	3519	1388	1333	4692
r32_0	2	749	1337	894	821	1186
r32_1	2	886	1806	886	1353	1698
r32_10	2	941	5487	1675	1713	6453

Table 2: Scheduling results for RGPOS problem instances

Graph	p	Opt CPES LPTES	LBMC	DC	LPTDC	PPS
50_0	3	933	2085	1690	1473	1864
50_1	3	956	2403	1807	1790	2176
50_10	3	811	6166	2281	2360	7255
100_0	5	898	2601	2435	1939	1501
100_1	5	831	2583	2353	2032	1704
100_10	5	929	7138	4602	4650	7349
150_0	6	1215	3442	3973	3166	2253
150_1	6	1104	3490	3464	3342	2539
150_10	6	1186	11269	6583	6319	11879
200_0	7	1351	4807	4895	4091	3175
200_1	7	1345	5343	4663	4450	3336
200_10	7	1446	18640	8824	8473	17132
250_0	7	2553	9505	9696	9855	8532
250_1	7	2377	9399	8750	8448	7127
250_10	7	2357	27198	12238	12835	26373
300_0	8	2464	8798	10316	9601	5850
300_1	8	2250	10388	9831	8098	6520
300_10	8	2397	28860	14076	14767	23844
350_0	9	2342	8986	11177	9880	5299
350_1	9	2371	9582	11012	10590	7879
350_10	9	2409	28824	16906	16855	24231
400_0	10	1796	6218	9258	7387	3809
400_1	10	1798	7290	8854	7504	4554
400_10	10	1781	24140	13889	14626	20964
450_0	10	2763	10363	15207	12273	6597
450_1	10	2872	12478	14645	14470	9924
450_10	10	3168	42928	22035	23214	33480
500_0	11	2085	7356	11864	9744	4663
500_1	11	2050	9288	11041	9789	5488
500_10	11	2054	28111	16049	17545	24285

sparse task graphs, with *RGPOS* graphs containing almost independent tasks.

Further analysis shows that, in fact, the two sets of problem instances violate several properties required by Hall and Posner [13]. Thus, *variety* is not ensured because the graph density does not vary and differences in multiprocessor architectures are not accounted for. Indeed, only one multiprocessor system, with fixed number of completely connected processors, is considered for each size n. *Relevance* is not provided either, since only completely connected multiprocessor systems are considered. *Regularity* is also violated for RGPOS examples since the structure of the task graphs and the optimal solutions is predetermined. It may thus be easy to define scheduling procedures that will always generate optimal schedules. Consequently, the *RGBOS* and *RGPOS* sets of problem instances proposed in [20] are not adequate to benchmark algorithms for the *MSPCD* problem.

A note on the behavior of the other heuristics on these two sets. As can be seen from Tables 1 and 2, *LBMC* and *PPS* displayed the worst performance. The main issue with *LBMC* is the poor approximation during clustering of the communication delays that are generated by assigning tasks to clusters. Simple heuristics, such as considering the maximum inter-task communication delay [24] or the weighted sum of communication delays, do not seem to work. As for the *PPS* heuristic, it was originally designed for fine-grained task execution times and communication delays, which is not the case here.

The third set of problem instances proposed in [20] that could be used for heuristics designed for general networks is the set *RGNOS* composed of 250 random task graphs generated similarly to the ones in *RGBOS*, but with number of tasks in the same range as for the *RGPOS* set. The authors provided very little information regarding this set and the *APN* algorithms. Neither optimal solutions, nor best-known solutions were provided and the number of processors was not specified either. Comparisons are thus very difficult to undertake. We applied the selected heuristics, however, to the subset of 50 tasks with $CCR = 1.0$ with $p = 2$. The relative behavior of the heuristics is similar to that observed previously. Complete results may be obtained as indicated in Appendix 2. It is worth noticing that the density of the task graphs in *RGNOS* is again very low, varying from 8% to 15%. As for the properties required by [13], *RGNOS* does not violate *relevance* and *regularity*, but *variety* is still not satisfied since only sparse graphs were generated.

4 New Sets of Benchmark Problem Instances

We propose two new sets of task graphs for analyzing and comparing scheduling heuristics addressing the *MSPCD* problem. The first is composed of randomly-generated DAGs. The generation procedures and the parameters settings (e.g., edge density) are such that the rules and principles of [13] are enforced. The second set contains task graphs with known optimal solutions for given multiprocessor architectures. In this section, we describe both the problem instances and the corresponding task graph generators for the two sets. The task graphs may be obtained from the authors as indicated in the Appendix 1.

Two generators were built for the first set of problem instances. The first produces task graphs with given heights and widths (i.e., with preselected number of "layers" and tasks within a layer; it should be noted, however, that we do not enforce the multi-layer design). Such task graphs display a predetermined level of parallelism and are thus suitable for experimentations involving different multiprocessor system architectures. The generator also allows to control the input parameters that affect other characteristics of the task graphs, such as granularity, the CCR factor ([20]), and the density.

The second generator yields task graphs with preselected densities. One thus overcomes the main drawback of the sets proposed in [20]. Task execution times and communication delays are determined randomly according to given input parameters. One may generate several graphs for the same input parameter values. The randomness of the procedure ensures that the resulting graphs are different, however.

Several input parameters are the same for the two generators: number of graphs to be generated, number of tasks within each task graph, maximum value of task duration (execution time), and maximum value of communication delay. The maximum number of levels (height) in each graph and the maximum number of successors for each task are specific for the first generator, while the density of the generated task graphs appears only for the second. The actual values for these parameters are determined randomly during the generation process for each particular task graph that is generated. Because no assumptions are made during the generation process regarding the problem-specific parameters, these generators obey the principles proposed in [13].

These two generators were used to build the first set of test examples. It consists of 180 random task graphs with $n \in \{50, 100, 200, 300, 400, 500\}$ and $\rho \in \{20, 40, 50, 60, 80\}$. For each (n, ρ) combination, we generated 6 graphs with different combinations of the parameters controlling the task computation time and communication requirements. Three task graphs of the same size were obtained by the first generator, the other three by the second. For each n, there are thus 30 task graphs with the same number of tasks, while for each ρ, there are 36 graphs with the same density. The proposed task graphs do not depend on the multiprocessor architecture. They can be scheduled to arbitrary multiprocessor systems.

We also propose a new set of randomly generated test instances with known-optimal solutions that overcomes some of the limitations of [20]. The generation process is as follows. A number of graphs m (set here to 10) is generated for each combination of multiprocessor architecture, defined by the number of processors p and the distance matrix $D_{p \times p}$, number of tasks n, and length of the optimal schedule SL_{opt}. All m graphs contain the same number of tasks, with the same task durations but with different edge densities ρ. First, the optimal schedule is generated in such a way that each processor p_k executes approximately the same number of tasks x_k (with 10% allowed deviations). The task durations are determined randomly, using a uniform distribution (mean equal to SL_{opt}/x_k and 10% deviation). All processors are completing the execution at the same time (SL_{opt}) and there are no idle time intervals between tasks. Tasks are then numbered according to their starting times: the first task on the first processor obtains number 1, the first task on the second processor obtains number 2, and so on; $p + 1$ will be the number of the second task with the smallest starting time (regardless of the index of the corresponding processor); $p + 2$ will be associated to the task that is the next one to begin its execution (the task could be on the same processor if the previous one has a small duration); etc. Figure 1 illustrates the structure of the optimal schedule obtained by this procedure.

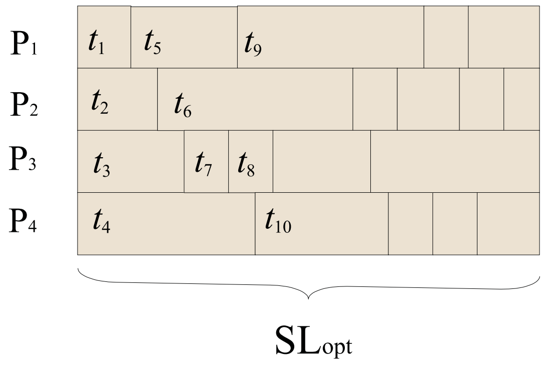

Figure 1: Structure of the optimal schedule for problem instances in set 2

Precedence relations among tasks are defined by adding edges to the graph and assigning communication loads to these edges. The maximum allowed density ρ_{max} is calculated first, because there cannot be an edge from task t_i to t_j if $st(t_i, p_k) + l_i > st(t_j, p_l)$. Then, for each $i = 0, ..., m - 1$, the number of edges corresponding to the density $\rho_{max} \cdot i/m$ is calculated and new edges are added randomly to the task graph to reach that number. The communication requirement along each edge (edge weight) is calculated by the following rule:

$$c_{ij} = \begin{cases} \infty, & \text{if } k = l, \\ (st(t_j, p_l) - (st(t_i, p_k) + l_i))/d_{kl}, & \text{otherwise.} \end{cases}$$

i.e., when tasks are executed on the same processor, the amount of data exchanged can be arbitrary large, otherwise, the communication amount is defined by the interval between the completion time of the first task and the starting time of the second one divided by the distance between the corresponding processors. The resulting task graph becomes the starting point for building the instance with the next higher density (one needs to add only a few new edges).

The input parameters for this generator thus are: the number of task graphs to be generated, the number of tasks in each task graph, the number of processors, the corresponding distance matrix, and the length of the optimal schedule. The number of tasks per processor, the task duration times and communication requests are then calculated as indicated previously.

The set is made up of 700 problem instances with different characteristics in terms of the multiprocessor system architecture (number of processors and distance matrix), number of tasks, and density of the task graphs. We selected 7 different multiprocessor system configurations defined by the connection architecture and the number of processors p: four hypercubes of dimensions 1, 2, 3, and 4 (i.e., 2, 4, 8, and 16 processors), and three mesh configurations of 6, 9, and 12 processors. 100 problem instances were generated for each of the 7 system configurations by combining the number of tasks n (10 values from 50 to 500 by increment of 50) and the connection density ρ (10 values from 0 (independent tasks) to 90% of the maximum allowed density ρ_{max}) of the corresponding task graph. These test instances do not satisfy all properties proposed in [13]: the second and third properties are not satisfied because the multiprocessor architecture and the structure of the final (optimal) schedule are pre-specified. The set of problem instances may still be quite useful, however, for the estimation of the solution quality offered by the given heuristic.

The lengths of the optimal schedules for each number of tasks n in the DAG are given in Table 3. These lengths do not depend on the values for p and ρ. The proposed set of problem instances can also be used for scheduling on completely connected architectures, as well as when communication delays are not relevant.

Table 3: Optimal schedule lengths for random task graphs

n	50	100	150	200	250	300	350	400	450	500
SL_{opt}	600	800	1000	1200	1400	1600	1800	2000	2200	2400

5 Analysis of Scheduling Results on the New Problem Sets

In this section we report and analyze results of applying the selected heuristics to the two sets of task graphs introduced in the previous section. The quality of the heuristic solutions is examined relative to the task graph characteristics and the multiprocessor architecture: number of tasks n, edge density ρ, number of processors p, interconnection architecture, and communication delay. It is worth emphasizing that our objective is not the comparison of scheduling heuristics, but rather the illustration of the value of the proposed sets of problem instances as benchmarks. We therefore emphasize the characteristics of the test examples with respect to parameter variations and algorithm behavior.

The hypercube is a widely used multiprocessor architecture. We thus experimented with 1-, 2-, and 3-dimensional hypercubes, i.e., we set $p = 2, 4, 8$. The distance matrix for 3-dimensional hypercube is

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

For $p = 2$ and $p = 4$ the respective distance matrices are represented by the corresponding upper left blocks of matrix D. To show that the proposed problem instances may be used even in extreme cases, we also considered completely connected architectures and systems with negligible communication delays.

We first present the results on the first set of problem instances (Set 1), which contains random task graphs with no known optimal solutions. The complete set of results is too large to be included in the paper. Only average results are presented. The complete set of results may be obtained from the authors as indicated in Appendix 2.

Table 4 displays comparison results of scheduling onto 1-, 2-, and 3-dimensional hypercube networks with significant communication delays. The first column indicates the number of tasks and the second displays the number of processors. Averages over 30 test instances of equal size are presented in the next columns: column three displays the average best known result (scheduling length), while the remaining six columns contain average percentage deviations from the best schedule reported in the third column. In most cases, the best solution was obtained by *CPES*. Among the other heuristics, *LPTES*, *DC,* and *LPTDC* generated better solutions than *CPES* in a few cases, although, on average, they do not perform well. In the following, we focus on illustrating how variations in the characteristics of the proposed problem instances (due to parameter variation during generation) induce the appropriate variations in algorithm behavior, thus showing the interest of the proposed set of problems for the evaluation of heuristics for the *MSPCD* problem.

The figures in Table 4 illustrate the dependency of the scheduling results on the num-

Table 4: Scheduling results for Set 1 problems: Hypercube multiprocessor systems and significant communication delays

		Av.	Percentage of deviation					
n	p	Best.	CPES	LPTES	DC	LPTDC	PPS	LBMC
50	2	517.47	22.21	27.71	40.85	41.10	54.86	70.64
50	4	500.47	19.55	25.37	42.35	40.50	64.38	104.09
50	8	537.43	11.30	16.26	32.62	30.84	63.22	115.81
100	2	1128.97	16.73	21.97	36.91	37.31	45.28	70.10
100	4	1071.87	16.24	21.75	71.47	40.68	52.97	102.52
100	8	1129.80	10.13	14.91	34.10	33.38	49.37	115.28
200	2	2355.57	12.63	18.10	29.11	31.46	38.90	64.44
200	4	2261.43	10.57	15.43	31.42	32.42	42.18	92.88
200	8	2349.03	6.40	10.65	26.50	27.57	41.84	111.30
300	2	3586.10	9.77	14.40	25.78	28.60	35.80	62.43
300	4	3396.83	9.07	14.08	29.65	31.91	42.33	94.23
300	8	3497.17	5.72	10.45	25.93	27.08	42.11	113.81
400	2	4835.76	9.45	13.84	28.44	30.63	33.60	61.88
400	4	4620.33	8.33	12.83	32.02	33.15	37.69	89.96
400	8	4759.20	5.06	9.23	28.18	29.10	36.39	110.39
500	2	6071.00	8.81	13.00	25.26	27.96	33.27	60.93
500	4	5784.63	7.93	12.29	28.85	30.43	37.07	91.60
500	8	5940.50	5.05	9.08	25.40	26.90	36.76	112.14

ber of processors. As expected, for breath-first heuristics, such as *CPES* and *LPTES*, which try to minimize the number of processors used, the deviations decrease when the number of processors increases. The opposite trend may be observed for the other heuristics, which may be explained by the additional communication burden corresponding to a larger number of processors. Figure 2 illustrates the performance of the heuristics in terms of deviations from the optimal schedule length, for $n = 200$ and various graph densities. The same behavior may be observed for the other values of n. To make for a clearer presentation, LPT-based heuristics are not included, since *LPTES* and *LPTDC* display a behavior similar to that of *CPES* and *DC*, respectively. The results displayed in Figure 2 indicate that the performance of all heuristics increases with the density of the task graphs (i.e., the respective deviations from the best known solutions decrease). This means that sparse task graphs are generally harder to address by scheduling heuristics, while it is easier for more dense graphs. The results also illustrate the previously noted relations between the number of processors and the quality of the solution obtained by each heuristic: the performance of *CPES* improves with the number of processors, while it decreases for the other heuristics.

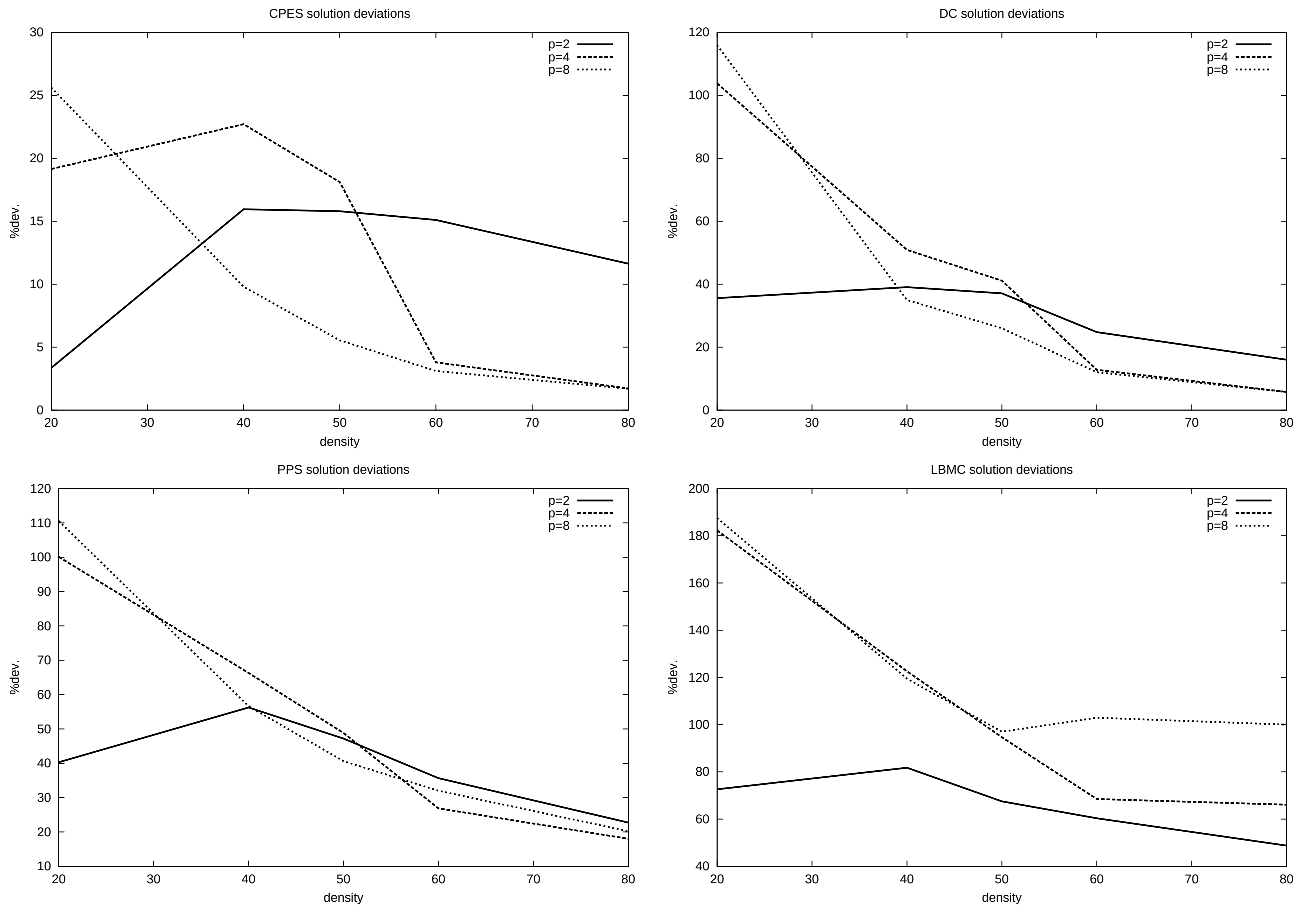

Figure 2: Solution deviations for task graphs with $n = 200$ with communication delays scheduled onto hypercube networks

Tables 5 and 6 display scheduling results for completely connected networks of processors (for $p = 2$, the hypercube and the complete interconnection network of processors are the same system) and for DAGs without communication delays, respectively. Most previous conclusions hold for these extreme cases as well. It might be noteworthy, however, that the *PPS* and *LBMC* heuristics display different behaviors relative to the number of processors: the quality of solution degrades when this number decreases. Their general performance is so poor, however, that this observation cannot be considered definitive. Work is required to improve these heuristics but this is beyond the scope of the current paper.

An important observation is that, on average, the best solution (Column 3) does not improve when more processors are added. An analysis of the detailed results indicates that some improvement is achieved for sparse-task graphs, while the degradation in solution quality is observed for dense ones. It appears that dense graphs do not benefit from adding processors to multiprocessor systems due to the time spent waiting for predecessor tasks to complete their execution and transferring data between related tasks scheduled

Table 5: Results for Set 1 for completely interconnected systems and significant communication delays

		Av.	Percentage of deviation					
n	p	Best.	CPES	LPTES	DC	LPTDC	PPS	LBMC
50	4	595.90	0.15	4.38	19.57	17.35	26.35	52.02
50	8	596.67	0.00	0.00	19.42	17.19	20.57	48.96
100	4	1235.43	0.06	4.73	22.29	20.92	23.30	56.82
100	8	1233.80	0.06	4.60	22.45	16.71	17.63	51.60
200	4	2481.00	0.00	4.44	19.64	20.38	22.31	57.09
200	8	4279.93	0.00	4.15	19.69	20.49	16.91	52.68
300	4	3667.10	0.00	4.71	19.81	21.20	23.44	60.77
300	8	3660.57	0.00	4.64	20.02	21.42	17.70	68.73
400	4	4956.97	0.00	4.37	22.56	23.39	20.92	58.25
400	8	4951.30	0.00	3.97	22.63	23.53	15.22	53.34
500	4	6187.80	0.00	3.97	20.11	21.56	20.98	59.09
500	8	6181.90	0.00	3.85	20.22	21.68	15.87	54.24

on different processors. Moreover, the *ES*-based heuristics, which often perform best, do not aim to use all processors. Consequently, in most cases, they produce exactly the same schedule even though the number of available processors increases. The same conclusions hold for the problem instances without communication delays.

The detailed results for the 700 random test instances with known optimal solutions (Set 2) are presented in [7]. In this section, we summarize these results and present a limited number of illustrative examples using the *CPES* heuristic.

We first analyze the influence of the edge (communication) density on the solution quality. Since the results for each p are similar, we display in Figures 3 and 4 the results for $p = 8$ only. Each figure displays 10 curves, one for each n, showing the change in the deviation from the optimal solution when ρ increases. Two multiprocessor architectures with fixed number of processors p were considered: Connected according to a given interconnection structure (mesh or hypercube), and completely connected. We also distinguished between cases with and without communication delays. In the latter case, the interconnection network does not play any role. Consequently, three cases exist only (and three graphs are displayed) for each p: Particular interconnection network with and without communication delays (Figure 3), and complete interconnection network with communication delays (Figure 4). An exception is the case $p = 2$ since the particular and the complete connections are the same and only two cases have to be analyzed.

As illustrated in Figures 3 and 4, the deviation from the optimal solution generally

Table 6: Results for Set 1 for systems with no communication delays

		Av.	Percentage of deviation					
n	p	Best.	CPES	LPTES	DC	LPTDC	PPS	LBMC
50	2	582.37	0.00	2.29	13.05	13.09	18.53	24.27
50	4	538.00	0.00	0.74	17.16	13.74	10.54	19.83
50	8	538.03	0.00	0.00	17.14	13.41	1.86	8.48
100	2	1218.73	0.00	2.88	13.64	13.84	18.51	27.41
100	4	1113.70	0.00	0.92	17.51	14.81	10.57	22.36
100	8	1113.83	0.00	0.00	17.46	15.00	2.11	10.73
200	2	2438.70	0.01	3.15	10.59	12.82	18.67	28.16
200	4	2231.10	0.00	1.16	13.28	13.28	12.06	22.88
200	8	2229.27	0.00	0.00	13.38	13.17	4.37	12.76
300	2	3597.10	0.00	2.81	10.52	14.14	19.23	28.74
300	4	3279.30	0.00	0.80	14.34	14.48	11.93	25.11
300	8	2373.47	0.00	0.00	14.55	14.70	4.17	14.14
400	2	4852.64	0.01	2.80	15.27	16.67	17.92	28.58
400	4	4450.80	0.00	0.86	18.32	18.14	11.04	23.56
400	8	4445.17	0.00	0.00	18.46	17.96	3.41	12.83
500	2	6045.80	0.00	3.03	12.53	15.44	18.68	29.02
500	4	5545.50	0.00	0.86	15.93	16.57	11.32	24.42
500	8	5539.00	0.00	0.00	16.04	16.70	3.99	14.30

grows with ρ when communication times are significant. This emphasizes the fact that the problem instances of the Set 2, due to their special structure used to provide known optimal solutions, behave quite differently form the completely random test problems of the Set 1. The figures also illustrate the impact of the type of processor connection on the quality of solution of scheduling heuristics on these problem instances. For completely connected systems, the impact of increasing ρ appears relatively small and that of n appear negligible. The performance seems to degrade somewhat for large number of processors, however. For systems without communication times, deviations are very small in all cases. The observations are quite different for systems that are not completely connected: deviations from optimum can be quite large, an order of magnitude larger than for completely connected systems. Moreover, the deviations seem to increase with n. Finally, the results show that sparse task graphs in Set 2 are easy to schedule with *CPES*, which points to the shortcomings of the problem instances in [20], which were all scheduled optimally using this heuristic.

The second main analysis concernes the influence of the number of processors on the scheduling results. It is natural to expect to be easier to schedule on systems with a smaller number of processors than on larger multiprocessor systems. Data dependencies

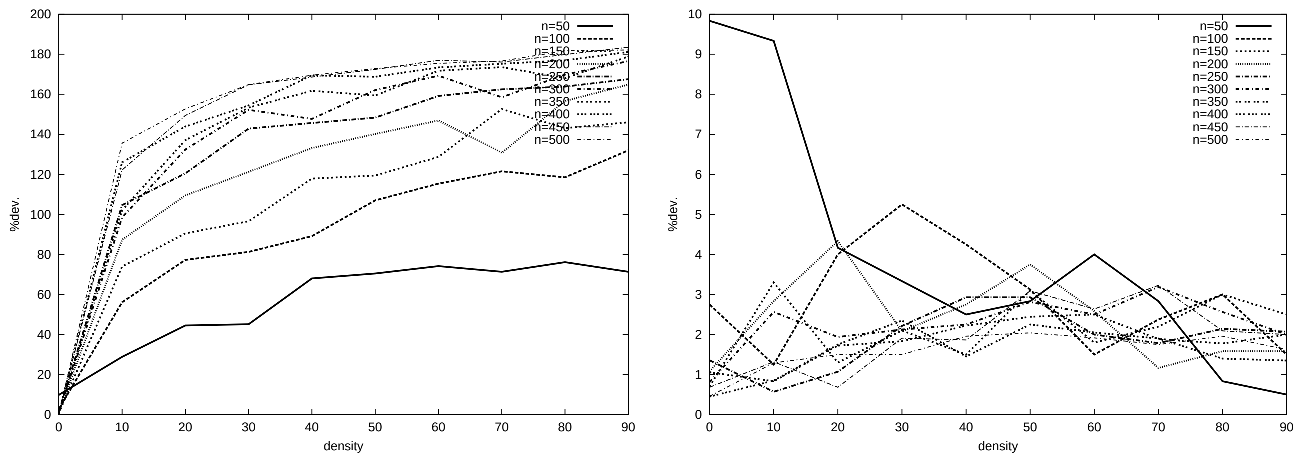

Figure 3: Solution deviations for 8-processor hypercube, with and without communication delays

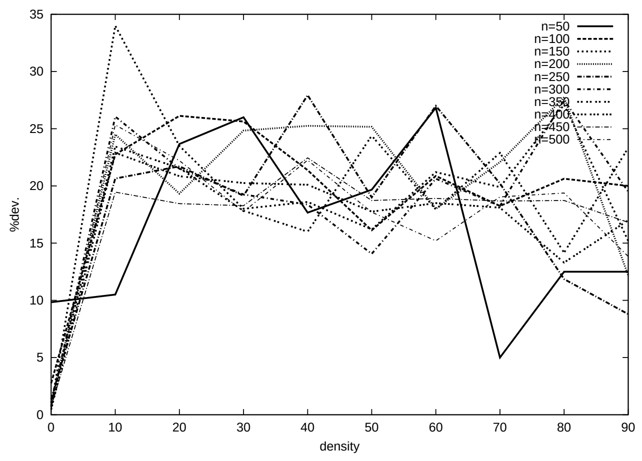

Figure 4: Solution deviations for completely connected 8 processors with communication delays

can indeed prevent the efficient exploitation of a large number of available processors. Sometimes, the number of processors can be even too large, resulting in very significant time spent on data transfers, as we already noticed while scheduling examples from Set 1.

Figures 5 and 6 illustrate the deviations in the schedule lengths obtained by *CPES*, for varying numbers of processors and task graphs with fixed density (at $\rho = 50\%$). The results support the previous hypothesis: when communication times are significant, the deviation from the optimal solution grows with the number of processors. It is interesting to note that, when communications can be neglected, the curves are very close (maximum deviation is 7%) and that the deviation is not increasing with n. The trend may also be observed for completely connected systems, but it is weaker (deviations increase at a slower rate). On the other hand, better task allocations may be obtained

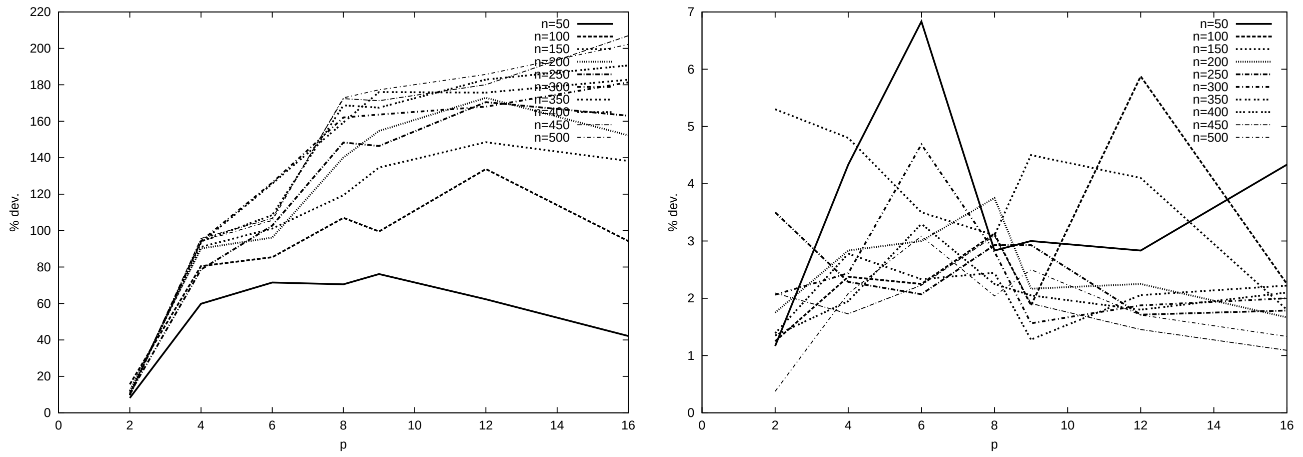

Figure 5: Solution deviations for task graphs with $\rho = 50$, with and without communication delays, scheduled onto incomplete networks

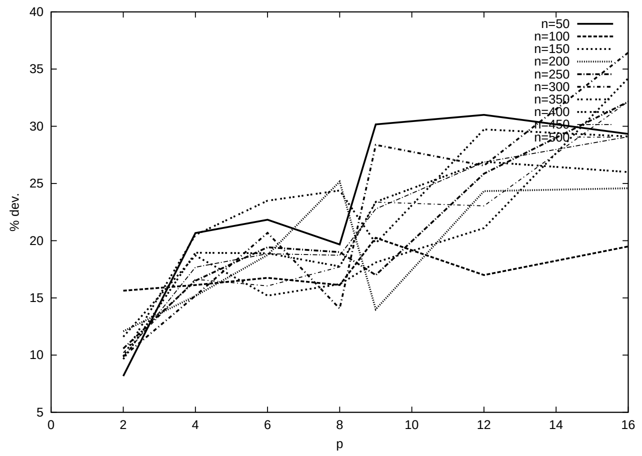

Figure 6: Solution deviations for task graphs with $\rho = 50$ with communication delays scheduled onto complete network of processors

when communication times are irrelevant. These observations yield a second explanation for the performance of *CPES* that found the optimal schedules for all problem instances from [20]: the single value of p given for each size n is sufficiently low to make the problems "easy".

It thus appears that it is easier to schedule sparse task graphs with *CPES* than denser ones. Moreover, better results may be expected when the number of processors is relatively small. We believe such hard or easy problem instances exist for all scheduling heuristics and that the set of problem instances we propose is sufficiently large to include challenging problems for most cases. This claim is supported by our care to vary the parameters related to the characteristics of both the task graphs and the multiprocessor systems, as well as by the results of the *LBMC*, *PPS*, and *DC* heuristics on the same set of task graphs (full results in [7]). The results show that *PPS* and *DC* encounter the same difficulties as *CPES* in scheduling dense task graphs. Moreover, the *DC* method does not distinguish between different types of processor connections. *LBMC*, as expected given its communication-minimizing heuristic rule, seems to have difficulty to address sparse task graphs (except for independent tasks where it acts like the *LPT* method). We notice a decrease in the solution deviations when the density of task graphs increases. For all heuristics, the deviations tend to increase with the number of processors, except for *LBMC* and *PPS* that seem to handle more easily the 4-dimensional hypercube ($p = 16$) than the 12-processor mesh when communications are significant.

The complete set of test instances with known optimal solution is quite large and it may not be necessary to use all the examples in all cases. According to our analysis, at least one of the two parameters p and ρ should be varied, however. This can reduce the size of the set by an order of magnitude (either 70 or 100 examples need to be evaluated, depending on which parameter is varied, p or ρ). Based on the experiments with the six constructive heuristics, we concluded that p and ρ are equivalent, in the sense of the insight their variation provides to the user. It is up to the user to decide which one to vary. We believe this conclusion holds for other methods as well.

6 Conclusion

In this paper, we propose two new sets of benchmark-problem instances for scheduling dependent tasks onto different multiprocessor architectures, not necessarily completely connected, taking into account communication delays (*MSPCD*). The first set includes completely random instances and may be used for any multiprocessor architecture. It is theoretically sound and general. The second set includes problem instances with known optimal solutions for specific processor interconnection networks. The generators are also described and made available.

Several representative heuristics were used to evaluate the sets of problems instances in the literature as well as the ones we propose. We analyzed the dependency of the deviation of the heuristic solution from the best known or the optimal one on several parameters: task graph density, communication delay, number of processors and connections between them. These parameters play a significant role in the quality of heuristic solutions and should be varied when scheduling heuristics are tested to ensure a fair evaluation. We noticed that communication delays are significantly degrading the performance of the scheduling heuristics. Regarding the multiprocessor architecture, it is of course easier to schedule onto completely connected processors but this is not always possible. Therefore, the type of processor interconnection should also be varied. For the other two parameters, number of processors and task graph density, at least one of them should be varied in order to objectively evaluate the efficiency of a particular scheduling heuristic. The sets of problem instances we propose provide this variation.

The *MSPCD* is a difficult and important problem class. The test-problem instances we propose bridge a gap in the literature by providing a theoretically sound and experimentally comprehensive framework for the fair evaluation of heuristics for the *MSPCD.*

Acknowledgments

This work was partially supported by NSF of Serbia, grant no. 1583, under project "Mathematical optimization models and methods with applications". Funding for this work has also been provided by the Natural Sciences and Engineering Council of Canada, through its Research Grant program, and by the Fonds F.C.A.R. of the Province of Québec.

References

[1] I. Ahmad and Y.-K. Kwok. Optimal and near-optimal allocation of precedence-constrained tasks to parallel processors: Defying the high complexity using effective search technique. In *Proceedings 1998 International Conference on Parallel Processing*, pages 424–431, 1998.

[2] J. Blazewicz, M. Drozdowski, and K. Ecker. Management of resources in parallel systems. In J. Blazewicz, K. Ecker, B. Plateau, D. Trystran, eds., *Handbook on Parallel and Distributed Processing*, pages 263–341. Springer, 2000.

[3] B. Chen. A note on lpt scheduling. *Operations Research Letters* 14:139–142, 1993.

[4] Jr. E. G. Coffman and R. L. Graham. Optimal scheduling for two processor systems. *Acta Informatica* 1:200–213, 1972.

[5] P. E. Coll, C. C. Ribeiro, C. C. de Sousa. Test instances for scheduling unrelated processors under precedence constraints. http://www-di.inf.puc-rio.br/ celso/grupo/readme.ps, 2002.

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

[7] T. Davidović and T. G. Crainic. New benchmarks for static task scheduling on homogeneous multiprocessor systems with communication delays. Publication CRT-2003-04, Centre de Recherche sur les Transports, Université de Montréal, 2003.

[8] T. Davidović, P. Hansen, and N. Mladenović. Scheduling by VNS: Experimental analysis. In *Proceedings Yugoslav Symposium on Operations Research, SYM-OP-IS 2001*, S. Minić, S. Borović, N. Petrović, eds., pages 319–322, Beograd, 2001.

[9] T. Davidović, P. Hansen, and N. Mladenović. Variable neighborhood search for multiprocessor scheduling problem with communication delays. In *Proc. MIC'2001, 4^th Metaheuristic International Conference*, J. P. de Sousa, ed., pages 737–741, Porto, Portugal, 2001.

[10] T. Davidović, N. Maculan, and N. Mladenović. Mathematical programming formulation for the multiprocessor scheduling problem with communication delays. In *Proc. Yugoslav Symposium on Operations Research*, N. Mladenović, Dj. Dugošija, eds., pages 331–334, Herceg–Novi, 2003.

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

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness.* W. H. Freeman and Company, 1979.

[13] N. G. Hall and M. E. Posner. Generating experimental data for computational testing with machine scheduling applications. *Operations Research* 49:854–865, 2001.

[14] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research* 9(6):841–848, 1961.

[15] A. A. Khan, C. L. McCreary, and M. S. Jones. A comparison of multiprocessor scheduling heuristics. In H. J. Siegel, editor, *Proceedings of the 8th International Symposium on Parallel Processing*, pages 243–250, Cancún, Mexico, IEEE Computer Society, 1994.

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

[17] Y.-K. Kwok and I. Ahmad. Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proceedings 7^{th} IEEE Symposium of Parallel and Distributed Processing (SPDP'95)*, pages 36–43, Dallas, Texas, USA, 1995.

[18] Y.-K. Kwok and I. Ahmad. Dynamic critical path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 7(5):506–521, 1996.

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

[20] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *J. Parallel and Distributed Computing* 59(3):381–422, 1999.

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

[22] S. Manoharan and P. Thanisch. Assigning dependency graphs onto processor networks. *Parallel Computing* 17:63–73, 1991.

[23] M. Pinedo. *Scheduling Theory, Algorithms and Systems.* 2dn edition, Prentice Hall, 2002.

[24] A. K. Sarje and G. Sagar. Heuristic model for task allocation in distributed computer systems. *IEE Proceedings-E* 138(5):313–318, 1991.

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

[26] T. Tobita and H. Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling* 5(5):379–394, 2002.

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

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

Appendix 1 - Description of Task Graph Files

The zip file `Bench.zip`, which can be downloaded from the following web address `http://www.mi.sanu.ac.yu/~tanjad/`, contains 180 completely random task graphs and the `readme.txt` file. File names are `t<n>_<r>_<i>.td`, where `<n>` should be substituted with the number of tasks in the corresponding graphs, and `<r>` with the edge density, while `i` is the index of the graph with the same `n` and `r` values (there are 6 graphs for each (n, r) pair). In each file, data are written in the following format:

- The first row contains the number of tasks n;
- The next n rows contain the data for task $i = 1, \dots, n$: index i, duration t_i, number of successors n_{succ}, the list of n_{succ} pairs s_j c_{ij} representing the index of the j^{th} successor and corresponding communication amount.

We report best known solutions in the file NonoptBestResults.dat (obtained by applying GA and VNS meta-heuristics). The structure of this file is given in the following table, where $D = C$ stands for complete interconnected processor networks, while $D = H$ indicates a hypercube multiprocessor architecture. 30 test examples are given for each n.

n	$p = 2$		$p = 4$			$p = 8$		
	$ccr = 0$	$ccr = 1$	$ccr = 0$	$ccr = 1$		$ccr = 0$	$ccr = 1$	
				$D = C$	$D = H$		$D = C$	$D = H$
50	SL	SL	SL	SL	SL	SL	SL	SL
.								
.								
.								
50	SL	SL	SL	SL	SL	SL	SL	SL
100	SL	SL	SL	SL	SL	SL	SL	SL
.								
.								
.								
100	SL	SL	SL	SL	SL	SL	SL	SL
.								
.								
.								

The zip file `Bench_opt.zip`, which can also be found at the same web address, contains the task graph examples with known optimal solutions and the corresponding `readme.txt` file. File names are `ogra<n>_<r>_<p>.td`, where n and r have the same meaning as in the previous case, while `<p>` should be substituted with the number of processors for which the optimal solution is given.

Appendix 2 - Complete Scheduling Results

The complete results for the sets *RGBOS* and *RGPOS* proposed in [20] are given in Tables 1 and 2. The third set proposed by Kwok and Ahmad [20] contains 250 task graphs. Since the authors did not provide any scheduling results, we run the 6 scheduling algorithms we selected on a subset and provide the corresponding scheduling results. The selected subset contains 50 task graphs with different number of tasks and degree of parallelism and the same value for the parameter $CCR = 1.0$. The results are reported in the file `RGNOS.RES,` which can be downloaded from the following web address `http://www.mi.sanu.ac.yu/~tanjad/`. The file is in LaTeX-like table format. The number of tasks appears in the first column and the schedule length in the third. The results are sorted according to the scheduling heuristic whose abbreviation is in the separate row

Table 7: Scheduling results for Tobita and Kasahara test examples [20]

			Percentage of deviation			
n	p	Av. Opt.	CPES	LPTES	DC	LPTDC
50	2	214.39	0.55	3.12	12.57	16.08
50	4	135.10	0.90	6.53	60.13	49.72
50	8	113.76	0.25	3.08	90.11	77.68
50	16	112.52	0.00	0.05	92.21	76.64
100	2	402.85	0.32	3.37	10.45	15.51
100	4	228.11	0.60	7.47	71.69	56.96
100	8	175.45	0.32	4.23	123.37	102.71
100	16	171.24	0.02	3.21	128.87	107.70
300	2	1206.14	0.11	2.00	62.21	11.16
300	4	619.63	0.38	7.23	85.13	63.85
300	8	398.31	0.32	7.08	187.90	153.82
300	16	361.51	0.22	1.23	217.21	179.66
500	2	1998.54	0.07	1.57	5.26	10.01
500	4	1021.56	0.26	5.64	86.48	62.01
500	8	617.99	0.31	7.21	208.24	167.32
500	16	521.53	0.04	2.18	265.25	216.76

indicating the beginning of the corresponding data. The value for p equals 2.

Tobita and Kasahara proposed a large number of test examples, but communication delays and multiprocessor architecture were not considered [26]. The input files can be found on `http://www.kasahara.elec.waseda.ac.jp/schedule`. There are 180 task graphs for each fixed value of the number of tasks n. Even though these problem instances are not directly relevant for the *MSPCD,* we applied the selected scheduling heuristics to the task graphs containing up to 500 tasks. Table 7 displays the average, over 180 instances, of the optimal schedule length value and the deviations from that value of the results obtained by each of the four heuristics (the deviations for the other heuristics range from 10% to over 200%). It is interesting to note that the deviations of the *ES*–based methods decrease when the number of processors increases, while the *PPS*, *LBMC*, and *DC*–based heuristics show the opposite trend. The result is quite natural since the first two heuristic aim to use all processors available, which yields higher levels of communications. For the *DC* heuristic, a large number of processors proves problematic when the tasks display high communication requirements.

Full scheduling results can be found at `http://www.mi.sanu.ac.yu/~tanjad/`. The files are also in LaTeX-like format. The name of each file is composed of the abbreviation of the heuristic, the name of the first author, followed by the number of tasks and the extension `.tex`. The results are written in the same format as for the previous case, and are sorted according to the number of processors in the target multiprocessor system. This means that the first 180 rows contain results of scheduling to $p = 2$ processors, the next line contains corresponding average values, then the structure is repeated for $p = 4$, $p = 8$, and $p = 16$.

We also provide the complete heuristic results of six selected scheduling heuristics applied to the Set 1 problem instances proposed in this paper. The format and file-name scheme used previously also applies here. The name of each file consists of the abbreviation of the method, followed by `nonopt`, the number of processors, and the indication of connection topology and communication issues. For example, `CPES.nonopt.p4.comm1.compl.tex` denotes the results of applying CPES scheduling heuristic to the Set 1 examples when the multiprocessor system contains 4 completely connected processors and communication time is significant. The corresponding resulting files can be found on the same web address.}