

Towards the Optimal Solution of the Multiprocessor Scheduling Problem with Communication Delays

Tatjana Davidović

Mathematical Institute, Serbian Academy of Sciences, Belgrade, Serbia, tanjad@mi.sanu.ac.yu

Leo Liberti

LIX Ecole Polytechnique, F-91128 Palaiseau, France, liberti@lix.polytechnique.fr

Nelson Maculan

COPPE – Systems Engineering, Federal University of Rio de Janeiro, Brazil, maculan@cos.ufrj.br

Nenad Mladenović

School of Information Systems, Computing and Mathematics, Brunel University, UB83PH Uxbridge UK,
nenad.mladenovic@brunel.ac.uk

We consider the Multiprocessor Scheduling Problem with Communication Delays, where the delay is proportional to both the amount of exchanged data between pairs of dependent tasks and the distance between processors in the multiprocessor architecture. Although scheduling problems are usually solved by means of heuristics due to their large sizes, we propose methods to identify optimal solutions of small and medium-scale instances. A set of instances with known optima is a useful benchmarking tool for new heuristic algorithms. We propose two new Mixed-Integer Bilinear Programming formulations, we linearize them in two different ways, and test them with CPLEX 8.1. To decrease the time needed by CPLEX for finding the optimal solution, we use Variable Neighborhood Search heuristic to obtain a good approximation for the initial solution.

Keywords: Multiprocessors, Task Scheduling, Communication Delays, Linear Programming, CPLEX.

1 Introduction

In this paper we propose formulation-based and combinatorial methods to find optimal solutions to the Multiprocessor Scheduling Problem with Communication Delays (MSPCD). This consists in finding a minimum makespan schedule to run a set of tasks with fixed lengths L_i on an arbitrary network of homogeneous processors. The task precedence is modelled by a directed acyclic graph $G = (V, A)$ where V is the set of tasks and an arc (i, j) exists if task i has to be completed before task j can start. If i and j are executed on different processors $h, k \in P$, they incur a communication cost penalty γ_{ij}^{hk} dependent on the distance d_{hk} between the processors, and on the amount of data c_{ij} exchanged between the corresponding tasks ($\gamma_{ij}^{hk} = \Gamma c_{ij} d_{hk}$, where Γ is a known constant).

Since this problem is a super-class of the classic scheduling problem, it is NP-hard. Its natural application field is in compiler design [13] and robotics [14]. The size of real-life instances is almost always too large to be solved to optimality, so heuristic algorithms are used to find good solutions. When designing new heuristic methods, it is necessary to validate them on benchmark instances with known optimal solutions. The motivation of this study (and purpose of this paper) is to provide a tool to accomplish this task, i.e. finding optimal solutions of the MSPCD. Of course, our methods are prohibitively computationally expensive to be applied to real-life MSPCD instances, so our numerical experiments were carried out on a set of small- and medium-sized instances.

In the literature one can find mathematical formulations for different variants of scheduling problems. For example, in [12] the formulation for scheduling independent tasks is given. Scheduling of dependent tasks without communication is modelled in several different ways [1, 10]. In [9]

the authors considered the problem of scheduling dependent tasks onto heterogeneous multiprocessor architecture, but neglected communication time. To the best of our knowledge, up to now no mathematical programming model involving communications was developed and solved to optimality. Moreover, in existing models targeting similar problems, the processor topology is assumed to be a complete graph.

The main contributions of this paper are: (a) an adaptation of the classic scheduling formulation with ordering variables to the MSPCD, and a compact linearization of the resulting bilinear program; (b) a new formulation based on rectangle packing, which considerably reduces the number of variables; (c) comparative computational experiments on CPLEX 8.1 solving the two formulations with the impact of good, heuristic-based initial solution for the search process.

The rest of this paper is organized as follows. The two formulations are presented in Section 2. In Section 3 we discuss the comparative computational results. Section 4 concludes the paper.

2 Problem formulation

In this section we present two new formulations for the MSPCD. One is an adaptation of the “classic” scheduling model with variables expressing both assignment and ordering, and the other is derived from the idea of identifying tasks with rectangles and packing rectangles into a larger rectangle representing the total makespan and processor set. The parameters for both models are: the set of task lengths L_i ($i \in V$), the task precedence graph $G = (V, A)$, the costs c_{ij} on the arcs $(i, j) \in A$, the processor distance matrix (d_{kl}) for $k, l \in P$. We let $\delta^-(j)$ be the set of precedents of task i , that is $\delta^-(j) = \{i \in V \mid (i, j) \in A\}$.

2.1 Classic formulation

The classic scheduling formulation employs a set of binary variables which control both assignment of task to processor and position in the order of tasks executed by the given processor, and a set of continuous variables indicating the starting time for each task. Thus, we let:

$$\forall i, s \in V, k \in P \quad y_{ik}^s = \begin{cases} 1 & \text{task } i \text{ is the } s\text{-th task on processor } k \\ 0 & \text{otherwise,} \end{cases}$$

and $t_i \in \mathbf{R}_+$ be the starting time of task i for all $i \in V$.

The MSPCD can be formulated as follows:

$$\min_{x, t} \quad \max_{i \in V} \{t_i + L_i\} \quad (1)$$

$$\forall i \in V \quad \sum_{k=1}^p \sum_{s=1}^n y_{ik}^s = 1 \quad (2)$$

$$\forall k \in P \quad \sum_{i=1}^n y_{ik}^1 \leq 1 \quad (3)$$

$$\forall k \in P, s \in V \setminus \{1\} \quad \sum_{i=1}^n y_{ik}^s \leq \sum_{i=1}^n y_{ik}^{s-1} \quad (4)$$

$$\forall j \in V, i \in \delta^-(j) \quad t_j \geq t_i + L_i + \sum_{h=1}^p \sum_{s=1}^n \sum_{k=1}^p \sum_{r=1}^n \gamma_{ij}^{hk} y_{ih}^s y_{jk}^r \quad (5)$$

$$\forall k \in P, s \in V \setminus \{n\}, i, j \in V \quad t_j \geq t_i + L_i - M \left(2 - \left(y_{ik}^s + \sum_{r=s+1}^n y_{jk}^r \right) \right) \quad (6)$$

$$\forall i, s \in V, k \in P \quad y_{ik}^s \in \{0, 1\} \quad (7)$$

$$\forall i \in V \quad t_i \geq 0, \quad (8)$$

where $M \gg 0$ is a sufficiently large penalty coefficient and γ_{ij}^{hk} represent the delay between tasks i and j as mentioned in Section 1.

Equations (2) ensure that each task is assigned to exactly one processor. Inequalities (3)-(4) state that each processor can not be simultaneously used by more than one task. (3) means that at most one task will be the first one at k , while (4) ensures that if some task is the s^{th} one ($s \geq 2$) scheduled to processor k then there must be another task assigned as $(s - 1)$ -th to the same processor. Inequalities (5) express precedence constraints together with communication time required for tasks assigned to different processors. Constraints (6) define the sequence of the starting times for the set of tasks assigned to the same processor. They express the fact that task j must start at least L_i time units after the beginning of task i whenever j is executed after i on the same processor k ; the M parameter must be large enough so that constraint (6) is active only if i and j are executed on the same processor k and $r > s$.

2.1.1 Linearization of the classic formulation

The mathematical formulation of MSPCD given by (1)-(8) contains linear terms in the continuous t variables and linear and bilinear terms in the binary y variables. It therefore belongs to the class of mixed integer bilinear programs. It is possible to linearize this model by introducing a new set of (continuous) variables $\zeta_{ijhk}^{sr} \in [0, 1]$ which replace the bilinear terms $y_{ih}^s y_{jk}^r$ in Eq. (5) [4, 5].

The linearizing variables ζ_{ijhk}^{sr} have to satisfy the following linearization constraints:

$$y_{ih}^s \geq \zeta_{ijhk}^{sr} \quad (9)$$

$$y_{jk}^r \geq \zeta_{ijhk}^{sr} \quad (10)$$

$$y_{ih}^s + y_{jk}^r - 1 \leq \zeta_{ijhk}^{sr} \quad (11)$$

$\forall i, j, s, r \in V, \forall h, k \in P$, which guarantee that $\zeta_{ijhk}^{sr} = y_{ih}^s y_{jk}^r$. We call this reformulation the *usual linearization*.

The number of variables and constraints in the linearized model is $O(|V|^4|P|^2)$. Solving the formulation with a Branch-and-Bound method, we need to solve a linear relaxation at each node. Thus, it is important to obtain formulations with a small number of constraints. Since the constraint size is due to the linearization constraints (9)-(11), we propose a different linearization technique, called *compact linearization*, which reduces the size of the problem. This consists in multiplying each assignment constraints (2) by y_{jk}^r ; the resulting bilinear terms are successively linearized by substituting them with the appropriate ζ variable: thus, we obtain valid linear relationships between the ζ and the y variables [8]. The compact linearization also assumes that $\zeta_{ijhk}^{sr} = \zeta_{jihk}^{rs}$, which holds by commutativity of product. Thus, the compact linearization for the MSPCD is as follows:

$$\forall i, j, r \in V, l \in P \quad \sum_{h=1}^p \sum_{s=1}^n \zeta_{ijhk}^{sr} = y_{jk}^r \quad (12)$$

$$\forall i, j, s, r \in V, h, k \in P \quad \zeta_{ijhk}^{sr} = \zeta_{jihk}^{rs} \quad (13)$$

Notice that although there are still $O(|V|^4|P|^2)$ constraints (13), these can be used to eliminate half of the linearizing variables and deleted from the formulation (any decent MILP pre-solver will actually perform this reformulation automatically). Hence, there are only $O(|V|^3|P|)$ constraints in the compact linearization.

2.1.2 Bounds to the objective function

Simply feeding the classic formulation to a MILP solver was found to be inefficient. To speed up the search, we artificially set lower and upper bounds to the objective function value.

Load Balancing (LB) and the Critical Path Method (CPM) are two fast alternatives to compute cheap lower bounds. The optimal solution cannot have a shorter execution time than the ideal load balancing case, i.e. when there is no idle time intervals and all the processors complete the execution exactly at the same time. In other words, if we let $T_{\max} = \max_{j \leq n} \{t_j + L_j\}$ denote the objective function, we have $T_{\max} \geq \frac{1}{p} \sum_{i=1}^n L_i = T_{LB}$. Furthermore, the length of final schedule can not be smaller than the length of the critical path, the longest path connecting a node with no predecessors to a node without successors. Let us denote by $\delta^+(j)$ the set of immediate successors of task j , i.e. $\delta^+(j) = \{i \in V : (j, i) \in A\}$. By using CPM the corresponding lower bound T_{CP} can be defined as $T_{CP} = \max_{j \leq n} CP(j)$ where

$$CP(j) = \begin{cases} L_j, & \text{if } \delta^+(j) = \emptyset, \\ L_j + \max_{j' \in \delta^+(j)} CP(j'), & \text{otherwise,} \end{cases}$$

Let \underline{T} be the greatest of the bounds obtained by LB and CPM; a constraint of the form $T_{\max} \geq \underline{T}$ is then also added to the formulation. Notice that this lower bound is valid throughout the whole solution process and does not depend on the current Branch-and-Bound node being solved.

Several efficient heuristics and meta-heuristics can provide upper bounds. These heuristics are seldom applicable to a Branch-and-Bound (B&B) algorithm because at any given (B&B) iteration, some of the variables are fixed – and it is usually difficult to force the graph-based heuristics to constrain the parts of the graph structure which correspond to the fixed variables. At the first iteration, however, no variables are fixed, which makes it possible to apply some efficient graph-based heuristics as a kind of pre-processing step to the whole (B&B) run. In our tests, we used Variable Neighborhood Search (VNS) [3] to compute tight upper bounds \overline{T} to the objective function value, and a constraint $T_{\max} \leq \overline{T}$ was added to the formulation prior to starting the solution process.

2.2 Packing formulation

The idea on which this model is based is to liken task i to a rectangle of length L_i and height 1, and to pack all the rectangles representing the tasks into a larger rectangle of height $|P|$ and length W , where W is the total makespan to be minimized [6].

For each task $i \in V$ let $t_i \in \mathbf{R}$ be the starting execution time (or alternatively, the horizontal coordinate of the bottom left corner of the rectangle representing task i) and $p_i \in \mathbf{N}$ be the ID of the processor where task i is to be executed (alternatively, the vertical coordinate of the bottom left corner of the rectangle representing task i). Let W be the total makespan (alternatively, the length of the larger rectangle where we want to pack the task-rectangles). Let x_{ik} be 1 if task i is assigned to processor k , and zero otherwise. In order to enforce non-overlapping constraints, we define two sets of binary variables:

$$\begin{aligned} \forall i, j \in V \sigma_{ij} &= \begin{cases} 1 & \text{task } i \text{ finishes before task } j \text{ starts} \\ 0 & \text{otherwise} \end{cases} \\ \forall i, j \in V \epsilon_{ij} &= \begin{cases} 1 & \text{the processor index of task } i \text{ is strictly less than that of task } j \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

The formulation is as follows:

$$\min_{t, p, \sigma, \epsilon} W \tag{14}$$

$$\forall i \in V \quad t_i + L_i \leq W \quad (15)$$

$$\forall i \neq j \in V \quad t_j - t_i - L_i - (\sigma_{ij} - 1)W_{\max} \geq 0 \quad (16)$$

$$\forall i \neq j \in V \quad p_j - p_i - 1 - (\epsilon_{ij} - 1)|P| \geq 0 \quad (17)$$

$$\forall i \neq j \in V \quad \sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1 \quad (18)$$

$$\forall i \neq j \in V \quad \sigma_{ij} + \sigma_{ji} \leq 1 \quad (19)$$

$$\forall i \neq j \in V \quad \epsilon_{ij} + \epsilon_{ji} \leq 1 \quad (20)$$

$$\forall j \in V : i \in \delta^-(j) \quad \sigma_{ij} = 1 \quad (21)$$

$$\forall j \in V : i \in \delta^-(j) \quad t_i + L_i + \sum_{h,k \in P} \gamma_{ij}^{hk} x_{ih} x_{jk} \leq t_j \quad (22)$$

$$\forall i \in V \quad \sum_{k \in P} k x_{ik} = p_i \quad (23)$$

$$\forall i \in V \quad \sum_{k \in P} x_{ik} = 1 \quad (24)$$

$$W \geq 0 \quad (25)$$

$$\forall i \in V \quad t_i \geq 0 \quad (26)$$

$$\forall i \in V \quad p_i \in \{1, \dots, |P|\} \quad (27)$$

$$\forall i \in V, k \in P \quad x_{ik} \in \{0, 1\} \quad (28)$$

$$\forall i, j \in V \quad \sigma_{ij}, \epsilon_{ij} \in \{0, 1\}, \quad (29)$$

$$(30)$$

where W_{\max} is an upper bound on the makespan W , e.g.

$$W_{\max} = \sum_{i \in V} L_i + \sum_{i,j \in V} c_{ij} \max\{d_{hk} \mid h, k \in P\}.$$

We minimize the makespan in (14) and (15), we define the time order on the tasks in terms of the σ variables in (16), and similarly we define the CPU ID order on the tasks in terms of the ϵ variables in (17). By (18), the rectangles must have at least one relative positioning on the plane. By (19) a task cannot both be before and after another task; similarly, by (20) a task cannot be placed both on a higher and on a lower CPU ID than another task. Constraints (21) enforce the task precedences; constraints (22) model the communication delays. Constraints (23) link the assignment variables x with the CPU ID variables p , and (24) are the assignment constraints.

Formulation (14)-(29) is also a bilinear MINLP where the bilinearities arise because of the communication delay constraints. Observe, however, that this formulation has variables with only two indices, as opposed to the formulation of Section 2.1 whose variables have three indices. Since linearization squares the size of the variable set, it appears clear that the packing formulation size is much smaller than the classic one.

2.2.1 Linearization of the packing formulation

We linearize formulation (14)-(29) by introducing variables $0 \leq z \leq 1$ as follows:

$$\forall j \in V, i \in \delta^-(j), h, k \in P \quad (z_{ij}^{hk} = x_{ih} x_{jk}).$$

Integrality on the linearizing variables z follows from the usual linearization constraints:

$$\forall j \in V, i \in \delta^-(j), h, k \in P \quad (x_{ih} \geq z_{ij}^{hk} \wedge x_{jk} \geq z_{ij}^{hk} \wedge x_{ih} + x_{jk} - 1 \leq z_{ij}^{hk}) \quad (31)$$

The above linearization constraints are equivalent to the following compact linearization:

$$\forall i \neq j \in V, k \in P \left(\sum_{h \in P} z_{ij}^{hk} = x_{jk} \right). \quad (32)$$

$$\forall i \neq j \in V, h, k \in P \left(z_{ij}^{hk} = z_{ji}^{kh} \right). \quad (33)$$

Although the set of compact linearization constraints (33)-(32) is smaller than the set of usual linearization constraints (31), the difference is not as dramatic as for the classic formulation ($O(|V|^2|P|)$ against $O(|V|^2|P|^2)$). Experimentally, we found that the compact linearization is beneficial when the precedence graph is dense. For sparse graphs the usual linearization sometimes yields shorter solution times. This is in accordance with the observation that a more precise evaluation of the magnitude order of the usual linearization constraint size is $O(|A||P|^2)$ rather than $O(|V|^2|P|^2)$, and that $|A|$ decreases with respect to $|V|^2$ as the sparsity of the graph increases.

3 Computational results

In this section we describe and discuss the computational experiments performed to test the ideas presented in this paper. All our computations were carried out on an Intel PIV 2.66GHz CPU with 1GB RAM running Linux. We used CPLEX v. 8.1 [7].

3.1 Instances

Several small size test instances known from the literature (like [11]) were used in our experiments. In addition, we tested a few examples with known optimal solutions (named with the prefix `ogra`) generated as in [2]. We scheduled small-sizes instances on: $p = 2$ and $p = 3$, completely connected processors. For `ogra_` examples the optimal solution is obtained when the tasks are well packed (as in ideal schedule) in the order defined by the task indices. For these task graphs it is always the case that $T_{max} = T_{LB} = T_{CP}$. This means that they have a special structure which makes them very hard instances for heuristic methods. In other words, in the cases when the task graph density is small, i.e. the number of mutually independent tasks is large, it is hard to find the task ordering which yields the optimal schedule.

Medium-size instances (with 30 and 40 tasks) are generated randomly, as it was proposed in [2]. These instances names are generated with the prefix `t` followed by the number of tasks, density and index distinguishing graphs with the same characteristics. They are scheduled on a ring of $p = 4$ processors.

3.2 Resulting tables

Here we present some selected computational results. In Table 1 comparative results for both formulations are given for small test instances. First four columns contain the task graph characteristics (name, number of processors, number of tasks, edge densities). The length of the optimal schedule is given in the fifth column, while the last two columns contain the time required by CPLEX to solve classical and packing formulation respectively. The time is represented by `<hours>:<minutes>:<seconds>.<hundreds of seconds>` with zero values in front omitted.

As we can see, with all improvements (artificial lower bounds, VNS heuristic initial solution) we were not able to solve to the optimality graphs with more than 10 tasks using classical formulation. On the other hand, with packing formulation 20 tasks represent no significant difficulty.

Medium-size examples seem to be difficult even for packing formulation. Results given in the Table 2 show that more than 24 hours are needed for most of them to be solved to the optimality.

Table 1: Results for small task graph examples

example	p	n	ρ	T_{max}	Classic	Packing
ogra_1	3	9	13.89	20	2:14:51	0.41
ogra_2	3	9	27.78	20	34:31	0.22
ogra_3	3	9	41.67	20	16:34	0.13
mt91	2	10	31.11	53	46:05	0.08
ogra_4	3	10	13.33	25	15:42:27	30.72
ogra_5	3	10	26.67	25	1:13:14	0.20
ogra_6	3	10	40.00	25	1:12:31	0.18
gra12	3	12	16.67	135	—	20:48.92
ogra_7	4	12	20	50	—	0.67
t20_10	4	20	8.95	110	—	45:43:51.52
t20_25	4	20	23.68	151	—	29:29.86
t20_50	4	20	53.16	221	—	17.06
t20_75	4	20	76.32	241	—	51:56.79
t20_90	4	20	80.52	242	—	11.72

Namely, for these examples we set time limit of 24 hours for the CPLEX execution. In the fourth column of Table 2 we put the required execution time to obtain the optimal solution (if it has been reached) or the given time limit (if the optimal solution remind unfound). The last columns contains the gap between the final solution and the obtained lower bound.

Table 2: Results for medium-size task graph examples

example	p	T_{max}^*	Time	Gap
t30_50_2	4	470	24:00:00	7.12%
t30_60_1	4	467	06:44:23	0.00%
t30_60_2	4	451	24:00:00	5.31%
t30_75_1	4	544	23:11:21	0.00%
t40_10_1	4	233	24:00:00	2.15%
t40_10_2	4	245	24:00:00	25.31%
t40_25_1	4	270	01:24:08	0.00%
t40_25_2	4	369	24:00:00	24.39%
t40_30_1	4	457	24:00:00	7.98%

The average CPU time improvement of the packing formulation over the best variant of the classic formulation is experimentally proved to be about 5000 times. Therefore, we believe that the packing formulation represents a valuable tool for solving small MSPCD instances to optimality.

4 Conclusion

In this paper two linear programming formulations for scheduling dependent tasks onto homogeneous multiprocessor architecture of an arbitrary structure with taking into account communication delays is proposed. The models are used within the commercial software for solving Mixed-Integer

Linear Programs CPLEX. To reduce the execution time required by CPLEX to solve test examples to the optimality, we added heuristic limitations to the upper bound and the lower bound and introduce reduction constraints to the original model.

References

- [1] E. H. Bowman (1959), The schedule-sequencing problem, *Operations Research* **7**(5), 621 – 624.
- [2] T. Davidović and T. G. Crainic (2006), Benchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systems, *Computers & OR* **33**(8), 2155 – 2177.
- [3] T. Davidović, P. Hansen, and N. Mladenović (2005), Permutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delays, *Asia-pacific Journal of Operational Research* **22**(3), 297 – 326.
- [4] R. Fortet (1960), Applications de l’algèbre de boole en recherche opérationnelle, *Revue Française de Recherche Opérationnelle* **4**, 17 – 26.
- [5] P.L. Hammer and S. Rudeanu (1968), *Boolean Methods in Operations Research and Related Areas*, Springer, Berlin.
- [6] Y. Guan and R.K. Cheung (2004), The berth allocation problem: models and solution methods, *OR Spectrum* **26**(1), 75 – 92.
- [7] ILOG (2002), *ILOG CPLEX 8.0 User’s Manual*, ILOG S.A., Gentilly, France.
- [8] L. Liberti (2005), Linearity embedded in nonconvex programs, *Journal of Global Optimization* **33**(2), 157 – 196.
- [9] N. Maculan, C. C. Ribeiro, S.C.S. Porto, and C. C. de Souza (1999), A new formulation for scheduling unrelated processors under precedence constraints, *RAIRO Recherche Opérationnelle* **33**, 87 – 90.
- [10] A. S. Manne (1960), On the job-shop scheduling problem, *Operations Research* **8**(2), 219 – 223.
- [11] S. Manoharan and P. Thanisch (1991), Assigning dependency graphs onto processor networks, *Parallel Computing* **17**, 63 – 73.
- [12] M. Queyranne and A. Schulz (1994), Polyhedral approaches to machine scheduling, *Technical report, TUB:1994-408*, Technical University of Berlin.
- [13] G. C. Sih and E. A. Lee (1993), A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, *IEEE Trans. Parallel and Distributed Systems* **4**(2), 175 – 187.
- [14] T. Tobita and H. Kasahara (2002), A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling* **5**(5), 379 – 394.