Mathematical Programming-Based Approach to Scheduling of Communicating Tasks

Tatjana Davidović[1], Leo Liberti[2], Nelson Maculan[3], Nenad Mladenović[1,4]

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

[2] *DEI, Politecnico di Milano, P.zza L. da Vinci 32, 20133 Milano, Italy*
email: liberti@elet.polimi.it

[3] *COPPE-Systems Engineering, Federal University of Rio de Janeiro P.O. Box 68511, 21941-972 Rio de Janeiro, Brazil*
email: maculan@cos.ufrj.br

[4] *GERAD, 3000 ch. de la Cote-Sainte-Catherine, Montréal, H3T 2A7, Canada*

December 15, 2004

Abstract

We present a MILP mathematical programming formulation for static scheduling of dependent tasks onto homogeneous multiprocessor system of an arbitrary architecture with communication delays. We reduce the number of constraints by applying a Reduction Constraint reformulation to the model. We solve several small-scale instances of the reformulated problem by using CPLEX 8.1. Upper bounds are computed with the Variable Neighborhood Search meta-heuristic applied directly to the graph-based formulation of the problem, whereas lower bounds are obtained by solving linear relaxations of the MILP formulation, further tightened by using load balancing and critical path method arguments.

Résumé

Nous présentons une formulation sous la forme d'un modèle de programmation linéaire en nombres entiers (PLNE) pour le problème d'ordonnancement avec des tâches dépendantes dans un système de multiprocesseurs homogènes associés à une architecture arbitraire en présence de délais de communications. Pour diminuer le nombre de contraintes nous utilisons une reformulation du modèle à l'aide d'une procédure de Réduction de Contraintes. Nous avons résolu plusieurs exemples de petite taille du modèle reformulé avec CPLEX 8.1. Les bornes supérieures ont été calculées par des techniques VNS appliquées directement à une formulation basée sur la théorie des graphes. Les bornes inférieures ont été obtenues en résolvant des relaxations linéaires de la formulation PLNE et améliorées par des méthodes du chemin critique et par des techniques de balancemment de charges.

Key words: Static scheduling, homogeneous multiprocessors, communication delays, reduction constraints, VNS.

1 Introduction

Parallel computing has been a very attractive research field for more than forty years [8]. One of the main problems is scheduling modules (tasks) to processors, i.e. definition of distribution among processors and execution order of tasks. There is a bulk of papers considering different variants of scheduling problems; a recent survey can be found in [1]. This problem is known to be NP-hard in the majority of cases, although there are several special cases that can be solved in polynomial time. Therefore, heuristic methods for finding good solutions are essential. Such suboptimal solutions can be used as upper bounds for an exact solution method based on Branch-and-Bound.

There are several possible ways to formulate scheduling problems: using graph representation for the program and/or multiprocessor architecture [3, 10, 26], via sets of instructions [20], or as a mathematical programming problem. In the literature one can find mathematical formulations for different variants of scheduling problems. For example, in [22] the formulation for scheduling of independent tasks is given. Scheduling of dependent tasks without communication is modelled in several different ways [2, 18, 27]. In [17] the authors considered the problem of scheduling dependent tasks onto completely connected heterogeneous multiprocessor architecture, but with negligible communication time. All these models do not involve data communication among processors. Moreover, it has always been assumed that the processor grid is completely interconnected. In this paper, we are interested in scheduling dependent tasks with communication delays onto a homogeneous, arbitrarily connected multiprocessor architecture. This variant of the scheduling problem has been referred to as Multiprocessor Scheduling Problem With Communication Delays (MSPCD). In this paper we propose a novel mixed-integer bilinear programming formulation for the MSPCD.

Traditionally, scheduling problems are solved by purpose-built heuristics; attempting to solve the MSPCD via a general purpose solver like CPLEX [9] is not usually a viable alternative. In this paper we suggest that, at least for small-scale instance, the MSPCD can actually be solved to optimality by using CPLEX 8.1. Upper bounds are computed by using Variable Neighborhood Search (VNS — see Section 4.2); lower bounds are obtained by solving a continuous relaxation of the problem which has been suitably reformulated by using Reduction Constraints [14, 12, 15], which are a subset of RLT constraints [24] (see Section 4.1). Further tightening of the lower bound is performed with considerations based on load balancing and critical path.

The paper is organized as follows. In the next section we give a description of the MSPCD problem based on graph theory. Section 3 contains the mathematical programming formulation of the problem, and shows an exact reformulation containing less constraints. Section 4 discusses the methods used to derive tight lower and upper bounds to be used in the Branch-and-Bound algorithm in CPLEX 8.1. Computational results are described in Section 5.

2 Problem description

The Multiprocessor Scheduling Problem With Communication Delays (MSPCD) is defined as follows: tasks (or modules) have to be executed on several processors; we have to find where and when each task will be executed, such that the total completion time is minimum. The duration of each task is known as well as the precedence relations among tasks, i.e. what tasks should be completed before some other could begin. In addition, if dependent tasks are executed on different processors, the data transferring times (or communication delays) are also considered as part of the set of input parameters.

The set of tasks to be scheduled is represented by a Directed Acyclic Graph (DAG) [3, 10, 26] defined by a quadruplet $\mathcal{G} = (M,\ E,\ C,\ \Lambda)$ where $M = \{1, \ldots, n\}$ denotes the set of tasks (modules); $E = \{e_{ij} \mid i, j \in M\}$ represents the set of communication edges; $C = \{c_{ij} \mid e_{ij} \in E\}$ denotes the set of edge communication costs; and $\Lambda = \{L_1, \ldots, L_n\}$ represents the set of task computation times (execution times, lengths). The communication cost $c_{ij} \in C$ denotes the amount of data transferred between tasks i

and j if they are executed on different processors. If both tasks are scheduled to the same processor the communication cost equals zero. The set E defines precedence relation between tasks. A task cannot be executed unless all of its predecessors have completed their execution and all relevant data is available. Task preemption and redundant executions are not allowed.

The multiprocessor architecture $\mathcal{A} = \{1, 2, \ldots, p\}$ is assumed to contain p identical processors with their own local memories which communicate by exchanging messages through bidirectional links of the same capacity. This architecture is modelled by a $p \times p$ *distance matrix* [3, 7]. The (k, l)-th element of the distance matrix $D = (d_{kl})$ is equal to the minimum distance between the nodes k and l. Here, the minimum distance is calculated as the number of links along the shortest path between two nodes. It is obvious that distance matrix is symmetric with zero diagonal elements.

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

$$\gamma_{ij}^{kl} = c_{ij} \cdot d_{kl} \cdot ccr, \tag{1}$$

where it is assumed that task i will be executed on processor k, task j on processor l and ccr represents the Communication-to-Computation-Ratio which is defined as the ratio between time for transferring the unit amount of data and the time spent for performing single computational operation. This parameter is used to describe the characteristics of multiprocessor system. In message passing systems ccr usually has a large value because communication links are very slow. For shared–memory multiprocessors the communication is faster since it consists of writing data from main (electronic) memory of one processor into global (also fast) memory and then into main memory of another processor. If the tasks are scheduled to the same processor, i.e. $k = l$, the amount of communication equals zero since $d_{kk} = 0$.

3 Mathematical formulation

In this section we give a mathematical programming formulation of the MSPCD. Let us denote the set of immediate predecessors of task j by $Pred(j)$, i.e. $Pred(j) = \{i \in M \mid e_{ij} \in E\}$.

Let

$$y_{jk}^{s} = \begin{cases} 1, & \text{if task } j \text{ is the } s\text{-th task executed on processor } k, \\ 0, & \text{otherwise,} \end{cases}$$

$\forall j \in M$, $k \in \mathcal{A}$. Variables t_j denote the starting time of task j for all $j \in M$.

The MSPCD can be formulated as follows:

$$\min_{y,t} \max_{j \leq n} \{t_j + L_j\} \tag{2}$$

subject to:

$$\sum_{k=1}^{p} \sum_{s=1}^{n} y_{jk}^{s} = 1 \quad \forall j \leq n \tag{3}$$

$$\sum_{j=1}^{n} y_{jk}^{1} \leq 1 \quad \forall k \leq p \tag{4}$$

$$\sum_{j=1}^{n} y_{jk}^{s} \leq \sum_{j=1}^{n} y_{jk}^{s-1} \quad \forall k \leq p \quad \forall s \in \{2, \ldots, n\} \tag{5}$$

$$\begin{aligned} t_j \quad \geq \quad & t_i + L_i + \sum_{k=1}^{p} \sum_{s=1}^{n} \sum_{l=1}^{p} \sum_{r=1}^{n} \gamma_{ij}^{kl} y_{ik}^{s} y_{jl}^{r} \\ & \forall i \in \mathrm{Pred}(j), \forall j \leq n \end{aligned} \tag{6}$$

$$\begin{aligned} t_j \quad \geq \quad & t_i + L_i - \alpha \left[2 - \left(y_{ik}^{s} + \sum_{r=s+1}^{n} y_{jk}^{r} \right) \right] \\ & \forall k \leq p, \forall s \leq n-1, \forall i, j \leq n \end{aligned} \tag{7}$$

$$y_{jk}^{s} \in \{0, 1\}, \quad \forall j, s \leq n \quad \forall k \leq p \tag{8}$$

$$t_j \geq 0 \quad \forall j \leq n, \tag{9}$$

where $\alpha \gg 0$ is a sufficiently large penalty coefficient and γ_{ij}^{kl} represent the amount of communication between tasks i and j as defined in (1).

Equations (3) ensure that each task is assigned to exactly one processor. Inequalities (4)-(5) state that each processor can not be simultaneously used by more than one task. (4) means that at most one task will be the first one at k, while (5) 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 (6) express precedence constraints together with communication time required for tasks assigned to different processors. Constraints (7) 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. If tasks i and j are not assigned to the same processor and they are mutually independent, the previous reasoning does not hold: even being r^{th} ($r > s$) task on some other processor l, j may start before i completes its execution. The last term of inequalities (7), i.e. $\alpha \left[2 - \left(y_{ik}^{s} + \sum_{r=s+1}^{n} y_{jk}^{r} \right) \right]$, is added to cover that case.

The mathematical formulation of MSPCD given by (2)-(9) contains bilinear terms in the y variables, and therefore belongs to the class of mixed integer bilinear programs. Variables y_{ik}^{s} are of 0-1 type and variables t_i are continuous. It is possible to linearize this model by introducing a new set of (continuous) variables $z_{ijkl}^{sr} \in [0, 1]$ which replace the bilinear terms $y_{ik}^{s} y_{jl}^{r}$ in Eq. (6).

Variables z_{ijkl}^{sr} have to satisfy the following linearization constraints:

$$y_{ik}^{s} \quad \geq \quad z_{ijkl}^{sr} \tag{10}$$

$$y_{jl}^{r} \quad \geq \quad z_{ijkl}^{sr} \tag{11}$$

$$y_{ik}^{s} + y_{jl}^{r} - 1 \quad \leq \quad z_{ijkl}^{sr} \tag{12}$$

$\forall i, j, s, r \leq n, \forall k, l \leq p$, which guarantee that $z_{ijkl}^{sr} = y_{ik}^{s} y_{jl}^{r}$. The following constraints:

$$z_{ijkl}^{sr} \quad = \quad z_{jilk}^{rs} \tag{13}$$

$$z_{iikk}^{ss} \quad = \quad y_{ik}^{s}, \tag{14}$$

based on the observations that $y_{ik}^{s} y_{jl}^{r} = y_{jl}^{r} y_{ik}^{s}$ (commutativity of product) and $(y_{ik}^{s})^2 = y_{ik}^{s}$ (a squared binary variable has the same value as the variable itself), are also valid constraints. In particular, (13) makes it possible to reduce the number of z variables by about half. The number of variables in the original model is $O(n^6)$. Constraints (14) can be added to the formulation.

The number of linearization constraints (10)-(12) is rather large: $O(n^6)$. These linearization constraints were experimentally observed to slow down the solution process considerably, even when the y

variables were relaxed to continuous. It was shown in [13] that assignment constraints like (3) can be multiplied by each y^r_{jl} variables (and successively linearized by substituting each resulting bilinear term with the appropriate z variable) to obtain *reduction constraints* which, together with constraints (13), turn out to be equivalent to the linearization constraints (10)-(12). The reduction constraint system for the MSPCD is as follows:

$$\sum_{k=1}^{p}\sum_{s=1}^{n} z^{sr}_{ijkl} = y^r_{jl} \quad \forall i,j,r \leq n, l \leq p. \tag{15}$$

Notice that there are only $O(n^4)$ reduction constraints. Next, we show that a reformulation of the problem containing the reduction constraint system instead of the usual linearization constraints (10)-(12) is exact.

3.1 Proposition
Constraints (3), (13), (15) imply the linearization constraints (10)-(12).

Proof. By (13) and (15), we have

$$\forall i,j,s,r \leq n \; \forall k,l \leq p \; (z^{sr}_{ijkl} \leq y^s_{ik} \wedge z^{sr}_{ijkl} \leq y^r_{jl}). \tag{16}$$

By (16), for any set J of index pairs (f,t) with $f \leq p, t \leq n$, we have $\sum_{(f,t)\in J} z^{tr}_{ijfl} \leq \sum_{(f,t)\in J} y^t_{if}$. Pick $k \leq p, s \leq n$ and consider the set $J = M \times M \backslash \{(k,s)\}$. For each $i,j,r \leq n$ and $l \leq p$, we have:

$$\begin{aligned}
\sum_{(f,t)\neq(k,s)} y^t_{if} &\geq \sum_{(f,t)\neq(k,s)} z^{tr}_{ijfl} && \Rightarrow \text{(add and subtract } z^{sr}_{ijkl}) \\
\sum_{(f,t)\neq(k,s)} y^t_{if} &\geq \sum_{f=1}^{p}\sum_{t=1}^{n} z^{tr}_{ijfl} - z^{sr}_{ijkl} && \Rightarrow \text{(substitute } y^r_{jl} \text{ by (15))} \\
z^{sr}_{ijkl} &\geq y^r_{jl} - \sum_{(f,t)\neq(k,s)} y^t_{if} && \Rightarrow \text{(add and subtract } y^s_{ik}) \\
z^{sr}_{ijkl} &\geq y^s_{ik} + y^r_{jl} - \sum_{f=1}^{p}\sum_{t=1}^{n} y^t_{if} && \Rightarrow \text{(substitute 1 by (3))} \\
z^{sr}_{ijkl} &\geq y^s_{ik} + y^r_{jl} - 1. && (17)
\end{aligned}$$

Constraints (16) and (17) are exactly the linearization constraints (10)-(12), as claimed. □

Our computational results show that this reformulation is much faster to solve than the original formulation.

4 Bounds

In this section we discuss the methods used to find tight lower and upper bounds $\underline{T}, \overline{T}$ to the objective function value throughout the Branch-and-Bound solution process, as simply using CPLEX built-in lower and upper bounding procedures was found to be inefficient.

4.1 Lower bound

A continuous relaxation of the above MILP formulation can be used to find a valid lower bound to the objective function value at each step of the Branch-and-Bound algorithm in CPLEX. This bound can be further tightened in two ways: Load Balancing (LB) and the Critical Path Method (CPM). 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 that the length of the critical path, the longest path connecting a node with no predecessors to a node without successors. Let us denote by $Succ(j)$ the set of immediate successors of task j, i.e. $Succ(j) = \{j' : e_{jj'} \in E\}$. By using CPM the corresponding lower bound T_{CP} can be defined as $T_{CP} = \max_{j \leq n} CP(j)$ where

$$CP(j) = \begin{cases} L_j, & \text{if } Succ(j) = \emptyset, \\ L_j + \max\limits_{j' \in Succ(j)} CP(j'), & \text{otherwise,} \end{cases}$$

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.

4.2 Upper bound

CPLEX is supplied with a general purpose heuristic that finds sub-optimal solutions for the problem at hand. For any given Branch-and-Bound region, some of the variables are fixed. For MILPs which model a combinatorial problems, more efficient heuristics are usually available based on the graph structure of the problem. In particular, several efficient heuristics and meta-heuristics exist for the MSPCD. These heuristics are seldom applicable to a Branch-and-Bound algorithm because at any given Branch-and-Bound 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 the efficient graph-based heuristics as a kind of pre-processing step to the whole Branch-and-Bound run.

In our tests, we used VNS [21] 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 CPLEX. We employed the VNS based method proposed in [6] which is very efficient in solving the MSPCD.

5 Numerical experiments

In this section we discuss the numerical results obtained by solving the proposed model with CPLEX 8.1. We describe our test examples in the next subsection, while the comparative numerical results are presented in subsection 5.3.

5.1 Description of test examples

Several small size test instances known from the literature [3, 11, 19, 23, 25] were used in our experiments. In addition, we tested a few examples with known optimal solutions generated as in [5] (which proved to be hard instances for meta-heuristic methods). We selected examples with different characteristics in order to examine the influence of task graph parameters on the efficiency of model-based solution methods. The relevant parameters to describe an instance of the task graph are for example:

- n- the number of tasks;
- ρ - the density of task graph edges;
- f_p - level of parallelism, i.e. average value of mutually independent tasks;

- f_c - the ratio between the required computation and communication time; and so on.

The value for f_p is calculated based on the number of processors, number of levels within task graph and the average number of tasks per level. This definition is different from the one given in [11] and it is more realistic since it depend on various parameters. Our f_c is equivalent to the CCR parameter of [11] and is calculated as follows

$$f_c = \frac{\sum_j L_j}{\sum_i \sum_j c_{ij}}.$$

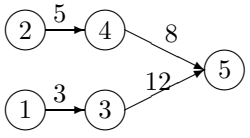
Moreover, there are two parameters describing the target multiprocessor architecture, namely the number of processors p and the distance matrix D describing the processor interconnection network.

According to the small size of test instances, we select the following values for these parameters

$p = 2$ with $D = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$; $p = 3$ with $D = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$; and $p = 4$ with $D = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$.

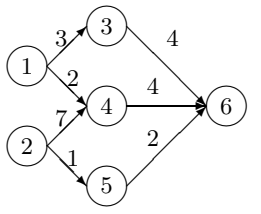
Task graphs used in this paper are illustrated on the figures below and are selected as the representative examples for different characteristics. For example, the *counter* is task graph containing 5 tasks that was used in [3] to illustrate the need of combining different heuristics during the scheduling process. Its CP based schedule onto 2-processor system happen to have longer execution time then the sequential one. *Test* is multistage graph with a good balance between parallelism and dependencies between tasks.

The task graph density was also shown to be an important parameter [4] and therefore we selected several examples with the same number of tasks and different densities. The parameters describing each particular task graph example are listed in the table containing scheduling results given in the next subsection.



i	1	2	3	4	5
L_i	2	3	5	8	4

a)

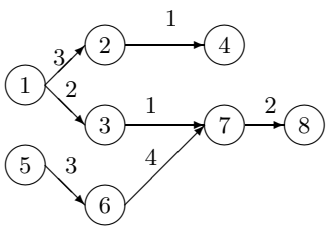


i	1	2	3	4	5	6
L_i	5	12	8	3	3	7

b)

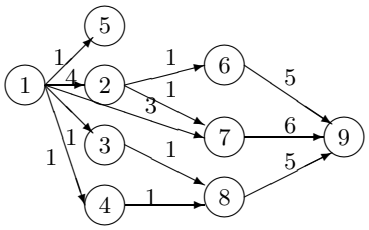
Figure 1: a) *counter* - Task graph example from [3]; b) *test* - Task graph with 6 nodes

Task graphs with known optimal solutions generated with the procedure proposed in [5] are all named with the prefix `ogra`. The optimal solution value 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.



i	1	2	3	4	5	6	7	8
L_i	5	6	2	3	6	4	4	2

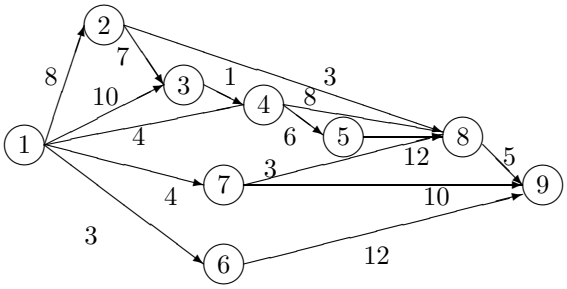
a)



i	1	2	3	4	5	6	7	8	9
L_i	2	3	3	4	5	4	4	4	1

b)

Figure 2: a) *sih91* - Task graph example from [25]; b) *kwok* - Benchmark task graph from [11]



i	1	2	3	4	5	6	7	8	9
L_i	60	30	15	40	15	30	35	50	40

Figure 3: *ss91* - The task graph example from the [23]

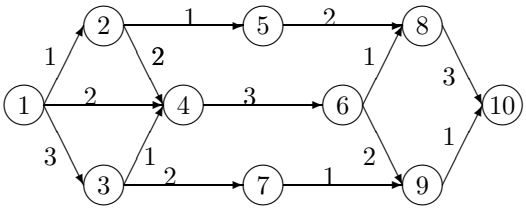
5.2 Bounds

Together with the optimal solutions, we also computed bounds to the objective function values. The upper bounds were obtained by running a custom-built implementation of the VNS metaheuristic applied to this problem (see Section 4.2). We computed three classes of lower bounds: $\underline{T}$ is the maximum between Load Balancing and CPM applied to the problem, as described in Section 4.1. LB_1 is the lower bound obtained by solving a linear relaxation of the linearized model from Section 3 using the CPLEX LP solver. LB_2 is the lower bound obtained by globally solving the continuous nonconvex problem obtained from the model in Section 3 by simply relaxing the binary variables to lie in $[0, 1]$. This bound was computed by running a VNS solver for continuous global optimization [16] on the problem instances.

5.3 Computational results

In Table 1 we present the computation times required by CPLEX 8.1 to find the optimal solutions of selected examples for different improvements and modifications of proposed model. All our computational results were obtained on a Pentium IV 2.66GHz processor with 1GB RAM running Linux.

The first column of Table 1 contains the names of the test examples. The second one contains the number of processors p in the target multiprocessor architecture. The next six columns contain parameters that characterize each particular task graph: number of tasks n, task graph density ρ, level of parallelism f_p, ratio between computation and communication time f_c. The computation times of



i	1	2	3	4	5	6	7	8	9	10
L_i	6	8	7	10	8	12	9	6	8	7

Figure 4: *mt91* - Example from [19]

different versions of the MILP model for MSPCD are given in the next four columns. Mod. 1 denotes the basic model with the lower bound given by Load Balancing (LB) and Critical Path Method (CPM) constraint $T_{max} \geq \underline{T} = \max\{T_{LB}, T_{CP}\}$. Mod. 2 is like Mod. 1 with symmetry and square constraints (13) and (14). Mod. 3 is obtained from Mod. 2 by replacing the linearization constraints (10)-(12) with the reduction constraint system (15). Mod. 4 is Mod. 3 where the upper bound to the root node of the Branch-and-Bound process has been obtained by heuristically solving the problem with VNS. The running time taken by VNS to find the good initial solution is estimated to be around 1% of the time needed by Mod. 3, which means that it can be neglected compared to the CPLEX 8.1 execution time required by Mod. 4. The last five columns contain: the length of optimal schedule T_{max}, the length of upper-bounding heuristic solution T_{VNS} obtained with VNS, and the three lower bounds to T_{max} ($\underline{T}$, LB_1 and LB_2) discussed in Section 5.2.

Note that parameter α from the inequality (7) was set to 2000 for all the examples. The "—" symbol is used in Table 1 to denote cases when CPLEX 8.1 was interrupted due to excessive CPU time requirements.

Table 1: Comparison results for different variants of MSPCD model

Instance	p	n	ρ	f_p	f_c	**Mod. 1**	**Mod. 2**	**Mod. 3**	**Mod. 4**	T_{max}	T_{VNS}	$\underline{T}$	LB_1	LB_2
counter	2	5	40.00	0.83	0.78	0:13	0:03	0:01	0:00	20	23	11	15	15
test	2	6	46.67	1.00	1.65	1:14	0:15	0:05	0:04	25	25	19	22	22
sih91	2	8	25.00	1.00	2.00	50:12	10:27	4:41	0:55	16	16	16	16	16
kwok	3	9	33.33	0.75	1.00	62:13:11	105:44:43	32:58:05	27:47:23	15	16	10	11	11
ss91	2	9	41.67	0.64	3.28	2:18:04	55:11	1:03:43	1:27:50	250	250	157.5	250	250
ogra_1	3	9	13.89	1.50	2.50	14:05:34	13:16:47	5:11:05	2:14:51	20	20	20	13	13
ogra_2	3	9	27.78	1.00	1.46	40:57:50	9:59:16	1:59:56	34:31	20	20	20	20	20
ogra_3	3	9	41.67	0.75	1.05	32:22:54	7:01:19	3:51:18	16:34	20	20	20	20	20
ogra_4	3	10	13.33	1.11	3.00	—	55:29:40	25:07:32	15:42:27	25	25	25	19	19
ogra_5	3	10	26.67	1.11	1.23	—	97:03:57	26:06:48	1:13:14	25	25	25	25	25
ogra_6	3	10	40.00	1.11	0.86	—	—	31:51:57	1:12:31	25	25	25	25	25
mt91	2	10	31.11	1.00	3.24	17:13:44	33:03:06	6:05:46	46:05	53	53	40.5	51	51

It appears clear, from our computational results, that the importance of the formulation is paramount for improving the solution speed taken by the Branch-and-Bound algorithm. Usually, this is taken to mean that a tightening reformulation, i.e. a formulation with added valid cuts which improve the lower bound, is bound to be better than the original formulation. In this case, the reduction constraints were shown to be an exact reformulation for the usual linearization constraints. However, the reformulation is not tighter in the sense of improved lower bounds, but rather because there are $O(n^4)$ reduction constraints instead of the $O(n^6)$ linearization constraints. The upshot of this is that each LP relaxation is solved in a much shorter time. In some cases, the reduction constraints reformulation (Mod. 3 in the table) was the only way to actually obtain the optimal solution, as the other methods took inordinately

long times (more than 120h of CPU time). This leads us to think that in some Branch-and-Bound nodes (certainly not the root one, and possibly quite deep in the node tree) the lower bounds provided by the reduction constraints reformulation are actually tighter than those given by the LP relaxation of the original problem; but we have no proof of this as yet.

It is also clear that an optimality approach through generic MILP methods (like Branch-and-Bound) to the solution of the MSPCD is necessarily limited to small-scale examples. This is certainly a stringent practical limit for real-life cases. However, when assessing the performance of a new heuristic method, it is customary to run it on instances with known optima. Our methods can successfully be used to generate a test set of small instances with known optima, which can then be used for comparison with heuristic methods.

6 Conclusion

In this paper, we proposed a bilinear MILP formulation for the multiprocessor scheduling problem with communication delays, where the processor grid has arbitrary topology. The formulation is linearized in a very efficient manner by using reduction constraints. We solve small-scale instances of the problem to optimality using CPLEX, where upper bounds are provided by VNS and lower bounds by solving a continuous relaxation of the problem, further tightened using load balancing and critical path arguments. It appears that the largest improvements in terms of computation time are given by the reduction constraints reformualtion, which reduces the number of linearization constraints in the problem.

References

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

[2] E. H. Bowman. The schedule-sequencing problem. *Oper. Res.*, 7(5):621–624, 1959.

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

[4] T. Davidović and T. G. Crainic. Benchmark problem instances for static task scheduling of task graphs with communication delays on homogeneous multiprocessor systems. *Centre de Recherche sur les Transports, CRT-2004-15, (accepted for Comput. & OR).*

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

[6] T. Davidović, P. Hansen, and N. Mladenović. Permutation based genetic, tabu and variable neighborhood search heuristics for multiprocessor scheduling with communication delays. *GERAD Tech. Report, G-2004-19, (accepted for publication in APJOR).*

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

[8] T. C. Hu. Parallel sequencing and assembly line problems. *Oper. Res*, 9(6):841–848, Nov. 1961.

[9] ILOG. *ILOG CPLEX 8.1 User's Manual*. ILOG S.A., Gentilly, France, 2002.

[10] 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.

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

[12] L. Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization (accepted for publication)*, 2003.

[13] L. Liberti. Automatic reformulation of bilinear minlps. *DEI, Politecnico di Milano, Technical report n. 2004.24*, July 2004.

[14] L. Liberti. Reduction constraints for the global optimization of nlps. *International Transactions in Operations Research*, 11(1):34–41, 2004.

[15] L. Liberti and C.C. Pantelides. An exact reformulation algorithm for large nonconvex nlps involving bilinear terms. *Journal of Global Optimization (submitted)*, 2004.

[16] L. Liberti Writing global optimization software. In L. Liberti and N. Maculan, *Global Optimization: from Theory to Implementation*, Nonconvex Optimization and Its Applications series, Kluwer, Dordrecht (to appear).

[17] N. Maculan, C. C. Ribeiro, S.C.S. Porto, and C. C. de Souza. A new formulation for scheduling unrelated processors under precedence constraints. *RAIRO Recherche Operationelle*, 33:87–90, 1999.

[18] A. S. Manne. On the job–shop scheduling problem. *Oper. Res.*, 8(2):219–223, 1960.

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

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

[21] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. & OR*, 24(11):1097–1100, 1997.

[22] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling. Technical report, TUB:1994–408, Technical University of Berlin, 1994.

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

[24] H.D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2:379–410, 1992.

[25] G. C. Sih. *Multiprocessor Scheduling to Account for Interprocessor Communication*. PhD thesis, University of California, Berkeley, 1991.

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

[27] H. M. Wagner. An integer linear-programming model for machine scheduling. *Nav. Res. Log. Quart.*, 6(2):131–140, 1959.