ILP formulations for Optimal Task Scheduling with Communication Delays on Parallel Systems

Sarad Venugopalan and Oliver Sinnen
Department of Electrical and Computer Engineering, The University of Auckland, New Zealand
sven251@aucklanduni.ac.nz, o.sinnen@auckland.ac.nz

***Abstract*—To fully benefit from a multiprocessor system, the tasks of a program are to be carefully assigned and scheduled on the processors of the system such that the overall execution time is minimal. The associated task scheduling problem with communication delays, $P|prec, c_{ij}|C_{max}$, is a well known NP-hard problem. We propose a novel Mixed Integer Linear Programming (MILP) solution to this scheduling problem, despite the fact that scheduling problems are often difficult to handle by MILP solvers. The proposed MILP solution uses problem specific knowledge to eliminate the need to linearise the bi-linear equations arising out of communication delays. Further, the size of the proposed formulation in terms of variables is independent of the number of processors. We analyse and discuss the influence of the different MILP components in respect to characteristics of the task graph such as structure and communication to computation ratio. The proposed MILP formulation is experimentally compared with previous MILP formulations used to solve this scheduling problem. The proposed formulation displays a drastic improvement in performance, which allows to solve larger problems optimally. We also observe strengths and weaknesses of the formulation related to the input characteristics.**

I. Introduction

For the performance and efficiency of a parallel program, the scheduling of its (sub)tasks is crucial. Unfortunately, scheduling is a fundamentally hard problem (an NP-hard optimisation problem [1]), as the time needed to solve it optimally grows exponentially with the number of tasks (unless $P = NP$). Existing scheduling algorithms are mostly heuristics as they try to produce good rather than optimal schedules, e.g. [2], [3], [4], [5], [6], [7], [8], [9], [10]. Having optimal schedules can make a fundamental difference, e.g. for time critical systems such as flight control, industrial automation, automotive applications, telecommunication systems, consumer electronics, robotics and multimedia systems. Multiprocessor systems are also popular in small portable devices such as cellphones or navigators to large systems such a industrial robots or aircraft. An optimal schedule may also be used as a benchmark to enable the precise evaluation of scheduling heuristics. Moreover, once an optimal schedule is found, it may be reused when a parallel program is rerun. Due to today's widespread use of parallel systems, an efficient parallelisation is fundamental to take advantage of the computational power available. It is hence of enormous practical significance to be able to schedule small and medium sized task graphs optimally on parallel processors. The objective of this work is to present a fast Mixed Integer Linear Programming (MILP) formulation for the classic problem of scheduling task graphs on parallel systems with communication delay, which is $P|prec, c_{ij}|C_{max}$ in the $\alpha|\beta|\gamma$ notation [11], [12]. Many heuristics have been proposed for task scheduling on parallel systems [13]. While they often provide good results and tend towards the optimal schedule there is no guarantee that the solutions are optimal, especially for task graphs with high communication costs [14],[15]. A number of approximation algorithms have been proposed for the scheduling problem [16],[17]. For the here addressed scheduling problem, $P|prec, c_{ij}|C_{max}$, no α-approximation is known [18]. The only known guaranteed approximation algorithm in [19] has an approximation factor depending on communication costs of the longest path in the schedule.

Given the NP-hardness, finding an optimal solution requires an exhaustive search of the entire solution space. For scheduling, this solution space is spawned by all possible processor assignments combined with all possible task orderings. Clearly this search space grows exponentially with the number of tasks, thus it becomes impractical already for very small task graphs. Hence, few attempts have been made to solve $P|prec, c_{ij}|C_{max}$ optimally. With the increase in processor power in computers, it is now feasible to find optimal solutions to larger instances of the scheduling problem. The A* algorithm is one such popular optimal search algorithm used to solve this scheduling problem [20],[21]. It begins the search with an empty solution space and is then incrementally grown. A* employs a best-first search technique [22],[23] and is guided by a problem specific cost function. The main drawback of A* is that it keeps all the nodes in memory and it usually runs out of memory long before it runs out of time making it unusable for medium and large sized problem instances.

In this paper, a MILP formulation is proposed for the task scheduling problem on parallel systems with communication delays. It uses an overlap approach [24] to set variables and constraints to ensure that no two tasks executing on the same processor overlap in time. The proposed formulation eliminates the need for the linearisation of bi-linear equations arising out of communication delays. Further, the number of variables is not a function of the number of processors, which is beneficial for the complexity of the formulation. The different components of the MILP formulation are analysed and their significance discussed. This helps to gain insights into the relevance of task graph characteristics for the efficient formulations of MILPs. An extensive experimental evaluation consisting of over 7400 schedules is carried out

and compared with other existing MILP formulations that solve this scheduling problem. The proposed formulation displays a drastic improvement in performance. The proposed formulation is found to outperform other MILPs especially when the Communication to Computation Ratio (CCR) of the task graph is high. It is also seen from the experiments that a larger number of processors do not necessarily mean a slow down in the runtime of the MILP.

The rest of the paper is organised as follows: Section II discusses the general use of MILP formulations for scheduling problems. Section III then describes the task scheduling model. Section IV discusses bi-linear forms arising out of communication delays and its linearisation for the studied scheduling problem. Section V discusses the proposed mathematical formulation, its relaxation and reduction using MILP. The constraint complexity of the proposed formulation is compared with other known formulations. Section VI details the experimental results wherein the runtime of the proposed formulation is compared with other known formulations in literature.

II. Mixed Integer Linear Programming

MILP may be used to solve optimisation problems, including scheduling problems. The MILP formulations can be broadly classified as discrete time and continuous time approaches [24],[25]. The discrete time approach introduces a new variable for each instant of time on each processor [26]. The number of time variables introduced in this approach explode when diverse execution times are present in the formulation. The continuous time approach, on the other hand, can handle diverse execution times, but its efficiency depends on how well the constraints and variables are formulated. The continuous time approach is further subdivided into three lines - sequencing, slots and overlaps. In sequencing, the formulation involves invoking new variables to determine if one task is executed after another task on the same processor [27],[28]. The number of constraints required to enforce the schedule requirements on each processor are known to grow quickly. In slots, each task is assigned to a space-time vacancy on a processor. The slot defines an order of tasks running on a processor [29],[30]. The start time and end time of tasks entering the slot are not fixed a priori. Since the exact number of slots required on each processor is not known a priori, a conservative number of slots (the number of tasks) has to be reserved and it suffers from a variable blow-up if the number of tasks to be scheduled is large. In overlap, variables are defined to prevent overlap of tasks scheduled on the same processor. Unlike other approaches, the number of variables and constraints in the formulation scales well as the number of tasks to be scheduled increases [24],[31],[32].

III. Task scheduling model

The tasks that are to be scheduled are represented as a weighted directed acyclic graph. The nodes in the graph represent tasks while directed edges represent data precedence relationships. Precedence relationships (if any) have to be respected at all times. The node cost is the time required for the task to complete its execution on a processor and the edge cost is the communication time between two tasks on different processors. If two tasks with data dependences are mapped onto the same processor, the communication between them is implemented by data sharing in local memory and no communication delay is incurred. The model assumes a fully connected network of homogeneous multiprocessors $P = \{1, \ldots, |P|\}$ with identical communication links. Each processor may execute several tasks, but each task has to be assigned to exactly one processor, in which it is entirely executed without pre-emption. Further, no multitasking or parallelism is permitted within a task. The execution time for each task on each processor and the data transfer times (or communication delays) between tasks with data dependence are given in advance as part of the task graph.

Formally, the tasks to be scheduled are represented by a directed acyclic graph (DAG) defined by a 4-tuple G=(V, E, C, L) where V denotes the set of tasks and E represents the set of edges. Each edge $(i, j) \in E$ defines a precedence relation between the tasks $i, j \in V$. A task cannot be executed unless all of its predecessors (parents) have completed their execution and all relevant data is available. The set $C = \{\gamma_{ij} : (i, j) \in E\}$ denotes the set of edge communication times. If tasks i and j are executed on different processors $h, k \in P, h \neq k$, they incur a communication time penalty γ_{ij}. If both tasks are scheduled to the same processor the communication time is zero. For a graph with $|V| = n$ tasks, the set $L = \{L_1 \ldots, L_n\}$ represents the task computation times (execution time length). Let $\delta^-(j)$ be the set of precedents of task i, that is $\delta^-(j) = \{i \in V | (i, j) \in E, j \in V\}$. The variables t_i and p_i are the main variables that describe a schedule for the problem to be solved. The start time of task i is t_i and the processor on which task i executes is p_i. The objective of this task scheduling problem is to allocate and schedule the tasks onto the processors such that the overall completion time W (makespan) is minimised [31],[33].

IV. Bi-linear reductions

Communication between tasks executing on different processors results in bi-linear constraints and needs to be linearised. A fast linearisation is crucial in developing an efficient MILP formulation for the task scheduling problem. A commonly used linearisation of the bi-linear forms arising out of communication delays for the task scheduling problem called the usual linearisation and compact linearisation are discussed in [34],[35]. The MILP formulations in [31] use the linearisation in [34] to solve the task scheduling problem and is categorised under the overlap approach discussed in Section II.

We are now looking at that linearisation. Let t_i be the start time of task i and t_j the start time of task j. Define the following variable for the MILP of the scheduling problem

$$x_{ih} = \begin{cases} 1 & \text{task } i \text{ runs on processor } h \in P \\ 0 & \text{otherwise} \end{cases}$$

When using this variable in the formulation, the precedence constraint created by an edge between two tasks i and j incurring a communication cost, is then

$$\forall j \in V : i \in \delta^-(j) \quad t_j \geq t_i + L_i + \sum_{h,k \in P, h \neq k} \gamma_{ij}(x_{ih}.x_{jk}) \tag{1}$$

So task j can only start after task i has finished $(t_i + L_i)$ plus the communication time. Remember, the communication cost is only incurred if the tasks are on different processors, otherwise it is zero. The task scheduling model presented in Section III assumes a fully connected network with identical communication links. Hence, the communication time between any two tasks i and j running on different processors is given by γ_{ij}. By definition, x_{ih} and x_{jk} are Boolean variables and their multiplication needs to be linearised.

The linearisation in [31] uses two different approaches. The linearisation variable z_{ij}^{hk}, where task i runs on processor h and task j runs on processor k, is defined as

$$\forall j \in V : i \in \delta^-(j), h, k \in P \qquad z_{ij}^{hk} = x_{ih}.x_{jk} \tag{2}$$

Using this definition, the multiplication of the Boolean variables $x_{ih}.x_{jk}$ of (1) is replaced by the linearisation variable z_{ij}^{hk} resulting in

$$\forall j \in V : i \in \delta^-(j) \quad t_j \geq t_i + L_i + \sum_{h,k \in P, h \neq k} \gamma_{ij}.z_{ij}^{hk} \tag{3}$$

By (3), the number of constraints produced is $|E|$ and the number of variables per constraint in terms of the processor combinations over z_{ij}^{hk} is $O(|P|^2)$.

The first linearisation method called **PACKING-USUAL** makes use of (3) and needs the additional (4)-(6) for the complete linearisation.

$$\forall j \in V, i \in \delta^-(j), h, k \in P \qquad x_{ih} \geq z_{ij}^{hk} \tag{4}$$
$$\forall j \in V, i \in \delta^-(j), h, k \in P \qquad x_{jk} \geq z_{ij}^{hk} \tag{5}$$
$$\forall j \in V, i \in \delta^-(j), h, k \in P \qquad z_{ij}^{hk} \geq x_{ih} + x_{jk} - 1 \tag{6}$$

(4) - (6) are used to simulate the logic of a Boolean multiplication using linear inequalities. By (4) - (6), the number of constraints produced is $|E||P|^2$ and the number of variables per constraint is $O(1)$. Hence, the total complexity of the PACKING-USUAL linearisation in terms of number of constraints is $O(|E||P|^2)$.

The second linearisation method called **PACKING-COMPACT** uses (3) plus (7) - (8), instead of (4)-(6).

$$\forall i \neq j \in V, k \in P \qquad \sum_{h \in P} z_{ij}^{hk} = x_{jk} \tag{7}$$
$$\forall i \neq j \in V, h, k \in P \qquad z_{ij}^{hk} = z_{ji}^{kh} \tag{8}$$

(7) is obtained by multiplying both sides of the equality (9) with x_{jk}; $\forall i \neq j \in V$, $k \in P$. (9) implies that any given task can run on exactly one processor.

$$\forall i \in V \qquad \sum_{h \in P} x_{ih} = 1 \tag{9}$$

(8) indicates that the multiplication $z_{ij}^{hk} = x_{ih}.x_{jk}$ is commutative. By (7), the number of constraints generated is $O(|V|^2|P|)$ and the number of variables per constraint is $O(|P|)$. So, the total complexity of PACKING-COMPACT in terms of number of constraints is $O(|E| + |V|^2|P|) = O(|V|^2|P|)$.

V. Proposed formulation

The literature surveyed indicates that amongst all MILP formulations, an overlap approach is best suited to tackle the task scheduling problem. All the MILP formulations discussed in this section are based on the overlap approach. A new MILP formulation for scheduling (SHD-BASIC), its relaxation (SHD-RELAXED) and reduction (SHD-REDUCED) are proposed and compared with the PACKING formulation in [31] as well as ILP-RBL and ILP-TC in [32]. The formulation in [31] utilise overlap variables adopted from [36] along with a technique for the linearisation discussed in the previous section. The formulation in [31] is improved in [32] by reworking the boolean logic for dependent tasks and by defining a partial order using a transitivity clause.

The first contribution of this work is to use problem specific knowledge to eliminate the bi-linear forms [31] arising out of communication delays. The proposed formulation eliminates the use of the z variable (in Section IV) for the linearisation of the bi-linear forms. This frees up to $|V|^2|P|^2$ z variables and its associated constraint complexity in the MILP formulation and speeds up the runtime of the solver. The second contribution is to run all variable indices in the proposed MILP formulation independent of the number of processors. As a result, the constraint complexity of the proposed MILP reduces to $O(|V|^2)$.

Next, we propose the basic formulation SHD-BASIC in Section V-A, its relaxation SHD-RELAXED in Section V-B and its reduction SHD-REDUCED in Section V-C. SHD-RELAXED and SHD-REDUCED formulations are compared with PACKING formulation, ILP-RBL and ILP-TC in Section V-D to bring out the advantages in using the proposed formulation.

A. *BASIC formulation (*SHD-BASIC*)*

The MILP is formulated as a min-max problem that involves minimising the maximum task finish time. The variables t and p gives the allocation and schedule of tasks on processors. The σ and ϵ variables model the relative position of tasks respectively along time and on the processors. They together ensures that tasks running on the same processor do not overlap in time. The constraints relate these variables to allow a fast ILP without the need for linearisation. Let W be the total makespan and $|P|$ the number of processors available. For each task $i \in V$ let $t_i \in \mathbf{R}$ be the start execution time and $p_i \in \mathbf{N}$ be the ID of the processor on which task i is to

be executed. In order to enforce non-overlapping constraints, define two sets of binary variables

$$\forall i,j \in V \quad \sigma_{ij} = \begin{cases} 1 & \text{task } i \text{ finishes before task } j \text{ starts} \\ 0 & \text{otherwise} \end{cases}$$

$$\forall i,j \in V \quad \epsilon_{ij} = \begin{cases} 1 & \text{PI of task } i \text{ is less than of task } j \\ 0 & \text{otherwise} \end{cases}$$

where PI is the Processor Index.

Based on these two types of binary variables the SHD-BASIC formulation is proposed next, followed by a detailed explanation of the role of each constraint. $(A01)$ is the objective, (A02)-$(A10)$ are the main constraints and $(A11)$-$(A15)$ are the bounds.

$$min \quad W \quad \text{(A01)}$$
$$\forall i \in V \quad t_i + L_i \leq W \quad \text{(A02)}$$
$$\forall i \neq j \in V \quad \sigma_{ij} + \sigma_{ji} \leq 1 \quad \text{(A03)}$$
$$\forall i \neq j \in V \quad \epsilon_{ij} + \epsilon_{ji} \leq 1 \quad \text{(A04)}$$
$$\forall i \neq j \in V \quad \sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1 \quad \text{(A05)}$$
$$\forall i \neq j \in V \quad p_j - p_i - 1 - (\epsilon_{ij} - 1)|P| \geq 0 \quad \text{(A06)}$$
$$\forall i \neq j \in V \quad p_j - p_i - \epsilon_{ij}|P| \leq 0 \quad \text{(A07)}$$
$$\forall i \neq j \in V \quad t_i + L_i + (\sigma_{ij} - 1)W_{max} \leq t_j \quad \text{(A08)}$$
$$\forall j \in V : i \in \delta^-(j) \quad t_i + L_i + \gamma_{ij}(\epsilon_{ij} + \epsilon_{ji}) \leq t_j \quad \text{(A09)}$$
$$\forall j \in V : i \in \delta^-(j) \quad \sigma_{ij} = 1 \quad \text{(A10)}$$
$$\forall i,j \in V \quad \sigma_{ij}, \epsilon_{ij} \in \{0,1\} \quad \text{(A11)}$$
$$\forall i \in V \quad p_i \in \{1,..,|P|\} \quad \text{(A12)}$$
$$\forall i \in V \quad t_i \geq 0 \quad \text{(A13)}$$
$$W \geq 0 \quad \text{(A14)}$$
$$\sum_{i \in V} L_i + \sum_{i,j \in V} \gamma_{ij} - W_{max} = 0 \quad \text{(A15)}$$

The constraints can be roughly categorised in the min-max objective ($A01$-$A02$), overlap constraints ($A03$-$A05$), processor constraints ($A06$-A07) and precedence constraints (A08-A10). To discuss the completeness and correctness of the formulation let us start by considering the case where there are no precedence constraints and hence no communication delays between tasks. Then, the minimisation objective (A01), constraints (A02-A06), (A08), (A11-A15) are sufficient to give a valid formulation. For each task, $(A02)$ specifies that the sum of the task start time and its execution time (hence the task's finish time) is less than or equal W. By $(A03)$, the sum of σ_{ij} and σ_{ji} is at most 1 and $(A04)$ enforces the same for the sum of ϵ_{ij} and ϵ_{ji}. When tasks are assigned to processors, the tasks that run on the same processor are to be ordered by defining their start time. Let us consider the set V_x of tasks that are assigned to some processor x (I.e. all tasks $i \in V$ such that $p_i = x$). Constraint (A06) enforces that for any two tasks $i,j \in V_x$, $\epsilon_{ij} = \epsilon_{ji} = 0$. Consequently, the non-overlapping constraint (A05) will imply an order on these two tasks by raising exactly one of the two variables σ_{ij} or σ_{ji} to 1 (by constraint A03). This set of variables σ_{ij} for all pairs of tasks $i,j \in V_x$ will give a total order of the tasks on processor x (transitivity being enforced by constraints (A05) and (A08)). Then constraint (A08) will affect valid starting times to the tasks while constraint (A01) will define the objective to be minimised.

For this independent tasks case, an ordering on a pair of tasks executing on different processors is not required as there are no precedence constraints and do not require ϵ_{ij} or ϵ_{ji} to be set to 1 whenever $p_i \neq p_j$. To model the precedence and communication delays, constraints (A07), (A09) and (A10) are added to the independent tasks case. Constraint (A10) pre-sets an ordering for all tasks that are connected by an edge and constraint (A08) affects the start times according to this order. Constraint (A09) will ensure that the communication delays are taken into account, but to meet this aim, exactly one of ϵ_{ij} or ϵ_{ji} must be equal to 1 when $p_i \neq p_j$. This is guaranteed by the addition of constraints (A07). Note that (A06) and (A07) are both required to enforce that exactly one of ϵ_{ij} or ϵ_{ji} is set to 1.

Both σ_{ij} and ϵ_{ij} are Boolean variables due to $(A11)$. $(A12)$ gives the bound on processor allocation for each task $i \in V$. By $(A13)$, all tasks have a start time greater than or equal to zero. By $(A14)$, the makespan W is greater than or equal to zero. By $(A15)$, W_{max} gives an upper bound on W and is defined as the sum of all task execution times and edge communication times. This is a worst case value which ensures that $(A08)$ is correct when $\sigma_{ij} = 0$. Note that there is no strict requirement that σ values have to correspond to a transitive closure for the task graph. E.g. If $\sigma_{13} = 1$ and $\sigma_{35} = 1$, then the value of σ_{15} is not relevant with respect to time ordering.

*B. RELAXED formulation (*SHD-RELAXED*)*

The RELAXED formulation is introduced to speed up the BASIC formulation. In the BASIC formulation (SHD-BASIC), (A07) is created for all edge pairs $i \neq j$. In the RELAXED formulation, these constraints are defined only for tasks with a direct edge between them. I.e. (A07A) and (A07B) are used instead of (A07). The formulation using (A07A) and (A07B) instead of (A07) is named SHD-RELAXED.

$$\forall j \in V : i \in \delta^-(j) \quad p_j - p_i - \epsilon_{ij}|P| \leq 0 \quad \text{(A07A)}$$
$$\forall j \in V : i \in \delta^-(j) \quad p_i - p_j - \epsilon_{ji}|P| \leq 0 \quad \text{(A07B)}$$

From section V-A, it is seen that constraint (A07) is included to model communication delays. However, since communication delay exists only between tasks that have an edge between them, constraints (A07) are needed only for such pairs of tasks.

*C. REDUCED formulation (*SHD-REDUCED*)*

The REDUCED formulation is introduced to remove redundant constraints that appear in SHD-RELAXED. The logic in $(A03)$ is redundant in $(A08)$ and the logic in $(A04)$ is redundant in $(A06)$. For $(A08)$: if $\sigma_{ij} = 1$, then $t_j \geq t_i + L_i$ and if $\sigma_{ji} = 1$, then $t_i \geq t_j + L_j$. If $\sigma_{ji} = \sigma_{ji} = 1$, then $t_j \geq t_i + L_i$ and $t_i \geq t_j + L_j$ should be simultaneously valid constraints. This is true only if $t_i = t_j$ and $L_i = L_j = 0$. If non-zero task execution times are considered, then by $(A08)$, at most one of the σ variable may be set to 1.

For $(A06)$: if $\epsilon_{ij} = 1$, then $p_j \geq p_i + 1$ and if $\epsilon_{ji} = 1$, then $p_i \geq p_j + 1$. If $\epsilon_{ij} = \epsilon_{ji} = 1$, then $p_j \geq p_i + 1$ and $p_i \geq p_j + 1$ should be simultaneously valid constraints. However, both the constraints cannot be simultaneously true as it is not possible to place a task on a higher and lower processor index at the same time. Hence, $(A06)$ may set at most one of the ϵ variables to 1. This allows $(A03)$ and $(A04)$ to be eliminated from SHD-RELAXED. The reduced formulation eliminating $(A03)$ and $(A04)$ from SHD-RELAXED is named SHD-REDUCED and the performance of SHD-REDUCED is compared with SHD-RELAXED. Note that $(A03)$ and $(A04)$ are redundant in the formulation but not for their linear relaxation.

D. Comparison of SHD-RELAXED *and* SHD-REDUCED *with* PACKING *formulation,* ILP-RBL *and* ILP-TC

SHD-RELAXED and its reduction SHD-REDUCED are used instead of SHD-BASIC in the following analysis. SHD-RELAXED generates $O(|E|)$ constraints using (A07A)-(A07B) whereas SHD-BASIC generates $O(|V|^2)$ constraints using (A07). The PACKING formulation in [31] uses linearisation variables to linearise the bi-linear inequalities arising from communicating edges. These linearisation variables are eliminated in ILP-RBL [32] by reworking the Boolean logic of the communicating edges. ILP-TC [32] reworks the linearisation of the bi-linear forms in the PACKING formulation using a transitivity clause in a manner that aids the elimination of over-defined inequalities in ILP-RBL.

For uniformity across comparisons, it is noted that the task scheduling model for the PACKING formulation unlike SHD-BASIC and its variants do not mandate a fully connected processor network. Table I compares the variable and constraint complexities of the formulations tested.

When ILP-RBL is used to schedule a large number of tasks on a small number of processors, the contribution of $|P|^2$ towards the constraint complexity of ILP-RBL diminishes.

The proposed formulation SHD-RELAXED has a constraint complexity of $O(|V|^2)$ as all variable indices are free of the number of processors. The bound on the number of processors available for scheduling is given by $(A12)$. SHD-RELAXED is also free of the linearisation variable z, making it faster than the PACKING formulation. SHD-RELAXED and ILP-RBL have a similar constraint complexity over a small number of processors but is much faster than ILP-RBL over a larger number of processors. SHD-RELAXED also runs faster than ILP-TC which have a higher constraint complexity, as observed from Table I. The reduced formulation SHD-REDUCED has fewer inequalities than SHD-RELAXED through the elimination of $(A03)$ and $(A04)$. However, as noted in Section V-C, these inequalities are redundant only for the formulation but not their linear relaxation. The experiments carried out in Section VI confirm that the complexity comparisons in this section are in agreement with the experimental results.

VI. Experimental results

The main goals of this section are: (a) performance comparisons of the proposed MILP formulation SHD-RELAXED with both linearisation of the PACKING formulation in [31] (PACKING-USUAL and PACKING-COMPACT), the MILP formulations in [32], namely ILP-RBL and ILP-TC and the reduced formulation SHD-REDUCED from Section V-C (b) to analyse the behaviour of the MILP formulations with respect to graph structures (c) study the effect of Communication cost to Computation cost Ratio (CCR) on MILP formulation runtime.

The computations are carried out using CPLEX 11.0.0 [37] on an Intel Core i3 processor 330M, 2.13 GHZ CPU and 2 GB RAM running with no parallel mode and on a single thread on Windows 7.

All experiments are run for the task scheduling model discussed in Section III. I.e. a fully connected processor network with identical bandwidth capacity is assumed. The input graphs used for experiments in Section VI-A are from [21] and the input graphs used for benchmarking in Section VI-B are from [31],[38]. The following two definitions of graph densities are used:

$$\Omega = |E|/|V| \tag{10}$$

$$\omega = (|E|/\gamma)100 \tag{11}$$

with the maximum possible number of edges in the graph as $\gamma = |V|(|V| - 1)/2$. The two density equations (10) and (11) arise due the difference in density definitions of the task graph databases used. It is also worth noting that very high densities are not realistic for most real software applications.

Two types of performance comparison experiments are carried out: a 1 minute timeout in Section VI-A and 12 hour timeout in Section VI-B. The 1 minute timeout experiments carried out in Section VI-A1 are to get many results to learn about the performance behaviour and on which input characteristics it depends. Section VI-A2 compares the relation between characteristics of the task graph structure with respect to the MILP formulations. Section VI-A3 studies the effect of CCR on MILP formulations. The 12 hour timeout experiments in Section VI-B are longer experiments for a direct comparison between runtime of the MILP formulations, especially in regards to previous work. Larger input graphs are tackled in these experiments. A Java implementation of all the MILP formulations compared and some of the optimal results that it returned can be found in the Green Banana (GB) scheduler suite [39]. They use the Graph eXchange Language (GXL) format [40] to represent input task graphs.

A. MILP comparisons with 1 minute timeout

A database of 207 task graphs, summarised in Table II, comprising of 10, 21 and 30 tasks of the following structures: fork, join, fork-join, in-tree, out-tree, series-parallel, pipeline, random, stencil and independent tasks is chosen [21]. The densities for the graphs in the database conform to the definition in (10). The experiments are carried out for 2, 4, 8 and 16 processors. A one minute timeout is set for each graph.

1) Overall performance evaluation: In this section, the overall performance of the graphs in the database are compared for the MILP formulations discussed. The number of graphs

Table I: Comparison between formulations tested

	PACKING-USUAL	PACKING-COMPACT	ILP-RBL	ILP-TC	BASIC	RELAXED	REDUCED
VARIABLES	$\|E\|.\|P\|^2$,z variables	$\|V\|^2.\|P\|^2$,z variables	free of z	free of z	free of z	free of z	free of z
CONSTRAINTS	$O(\|V\|^2+\|E\|\|P\|^2)$	$O(\|V\|^2.\|P\|)$	$O(\|V\|^2+\|E\|\|P\|^2)$	$O(\|V\|^3)$	$O(\|V\|^2)$	$O(\|V\|^2)$	$O(\|V\|^2)$

Table II: Detailed Structure of the 207 Graph Database

Graph Structure	$n = 10$	$n = 21$	$n = 30$	Total
Fork-Join	4	4	4	12
Fork	4	4	4	12
Independent	1	1	1	3
InTree	8	8	8	24
Join	4	4	4	12
OutTree	8	8	8	24
Pipeline	4	4	4	12
Random	16	16	16	48
Series-Parallel	16	16	16	48
Stencil	4	4	4	12

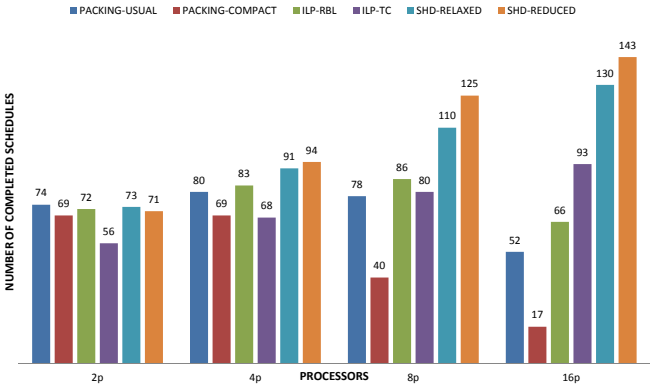


Figure 1: Completed schedules of different formulations over number of processors

for which an optimal schedule was returned by CPLEX within one minute for the 207 graph set is tallied and plotted in Figure 1. The comparison criteria used here is completed schedules within one minute.

It is seen from Figure 1 that SHD-RELAXED and SHD-REDUCED outperform all the other ILP's on 4, 8 and 16 processors. PACKING-USUAL marginally outperforms SHD-RELAXED and SHD-REDUCED over 2 processors, with 74 optimal solutions found in the 207 graph dataset. However, the performance of PACKING-USUAL and PACKING-COMPACT quickly degrades over 4, 8 and 16 processors in comparison to SHD-RELAXED and SHD-REDUCED. ILP-TC has a steady performance improvement over 2, 4, 8 and 16 processors unlike PACKING-USUAL, PACKING-COMPACT and ILP-RBL. This performance increase is attributed to the constraint complexity of ILP-TC that is independent of the number of processors. It is found that an increase in the number of processors does not necessarily imply a decrease in performance. Other than ILP-TC, only SHD-RELAXED and SHD-REDUCED have a steady performance increase over 2, 4, 8 and 16 processors. SHD-RELAXED and SHD-REDUCED also have a constraint complexity independent of the number of processors. Unlike ILP-TC with a constraint complexity of $O(|V|^3)$, the constraint complexity of SHD-RELAXED and SHD-REDUCED is $O(|V|^2)$. This improved constraint complexity matches with the experimental comparisons as it is observed that SHD-RELAXED and SHD-REDUCED outperforms ILP-TC.

2) Structure based performance evaluation: In this section, we analyse the performance of the MILP formulations over the graph structures given in Table II. Figure 2, Figure 3, Figure 4 and Figure 5 depict the previous results separated by the individual graph structures over 2, 4, 8 and 16 processors for the formulations SHD-RELAXED, SHD-REDUCED, PACKING-USUAL, PACKING-COMPACT, ILP-RBL and ILP-TC.

From the 2 processor experiments in Figure 2, it is seen that SHD-RELAXED and SHD-REDUCED display a relatively similar performance with respect to PACKING-USUAL or PACKING-COMPACT except for PIPELINE and RANDOM graphs where PACKING-USUAL exhibits a better performance. ILP-RBL is also seen to perform well in comparison with the other formulations (as expected to be the case over a smaller number of processors). For the 4 processor experiments in Figure 3, it is observed that either SHD-RELAXED or SHD-REDUCED displays a similar or better performance in relation to the other formulations and for the 8 and 16 processor experiments (in Figure 4 and Figure 5 respectively), their performance gains over other formulations are observed to be prominent.

The Fork-Join graphs have the best performance with SHD-RELAXED and ILP-TC whereas the Forks perform best on SHD-REDUCED. Independent graphs have a similar performance with SHD-RELAXED, SHD-REDUCED, ILP-RBL, ILP-TC and PACKING-USUAL. InTree graphs on 2, 4 processors are seen to best perform with SHD-RELAXED, PACKING-USUAL and on 8, 16 processors are seen to best perform with SHD-REDUCED. OutTree graphs are seen to perform best with SHD-REDUCED whereas Pipeline graphs are seen to perform best with SHD-RELAXED (except on 2 processors where PACKING-USAL and ILP-RBL are seen to perform better). Random graphs are seen to perform best on SHD-REDUCED (except on 2 processors were PACKING-USUAL and ILP-RBL perform better). Series-Parallel and Stencil graphs are both seen to perform best over SHD-REDUCED. Overall, it is observed that SHD-RELAXED or SHD-REDUCED have the best performance for each graph structure.

Table III gives the percentage of individual graph structures that passed the 1 minute timeout test averaged over 2, 4, 8 and 16 processors for SHD-RELAXED and SHD-REDUCED in Figure 2 to Figure 5. It is seen that the graph structures that perform the best over these two MILP formulations are Random and Pipeline and the worst are Join, Fork and Independent. These graphs have a very similar structure because

Table III: Performance Statistics on Individual Graph Structures over SHD-RELAXED and SHD-REDUCED

RELAXED	% Passed	REDUCED	% Passed
Random	62.5	Random	70
Pipeline	62.5	Pipeline	60.25
Stencil	54	Series-Parallel	56
Series-Parallel	52.25	OutTree	47.75
InTree	44.75	InTree	46.75
OutTree	42.5	Stencil	39.5
Fork-Join	35.25	Fork-Join	33.25
Join	27	Join	29.25
Independent	25	Fork	27
Fork	22.75	Independent	25

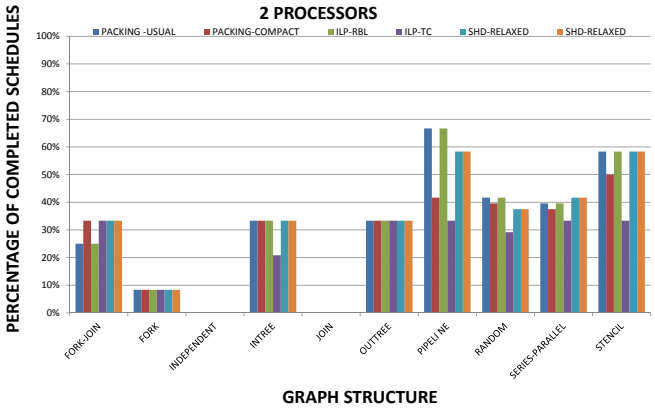


Figure 2: Completion percentage of formulations over graph structures 2 processors

a Fork or Join is a set of Independent tasks with an edge from or to another task. Independent tasks are in general harder to schedule since they have no precedence constraints and all possible task combinations on an allocated processor have to be tried out. Stencil graphs are seen to perform better over SHD-RELAXED as compared to SHD-REDUCED. The other graph structures are found to have a similar performance with SHD-RELAXED and SHD-REDUCED.

3) Effect of CCR on MILP formulation runtime: In this section, the effect of CCR on the performance of MILP

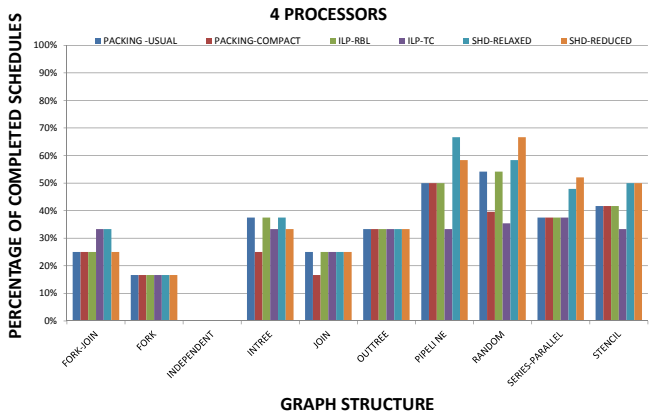


Figure 3: Completion percentage of formulations over graph structures on 4 processors

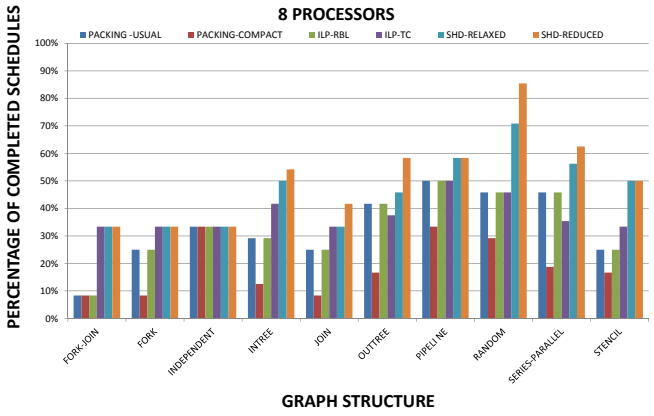


Figure 4: Completion percentage of different formulations over graph structures on 8 processors

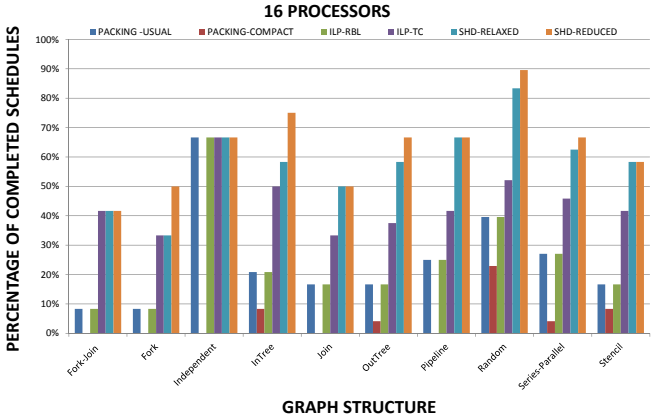
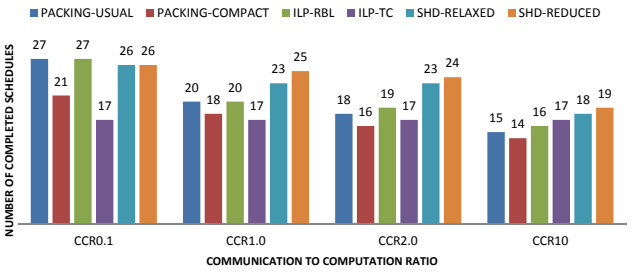


Figure 5: Completion percentage of formulations over graph structures on 16 processors

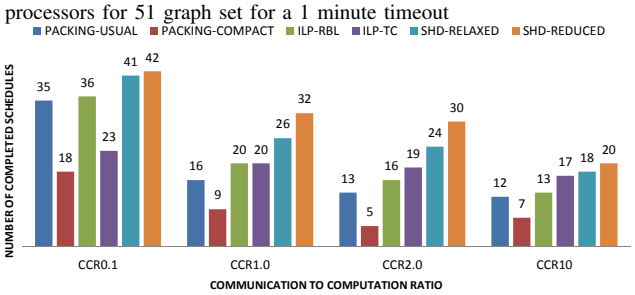
formulation is studied. Of the 207 graphs in the database, 51 each are of CCR 0.1, CCR 1.0, CCR 2.0 and CCR 10. The independent graphs in the database have no communication edges and are not considered for the analysis. The number of graphs for which an optimal schedule was returned by CPLEX within one minute is tallied and plotted in Figure 6 for 4 and 8 processors.

It is observed that an increase in CCR results in a decrease in the performance of all the MILP formulations in the experiment irrespective of the number of processors they are scheduled on. A possible explanation for this behaviour is that higher CCR values increase the schedule length variance between different schedules. A single task allocated to the wrong processor can imply a strong penalty on the schedule length due to large remote communication. The LP relaxation of the ILP formulations used by the solver can then be further away from the optimal solution of the ILP. This results in longer runtimes of the branch-and-bound part of the solver.

This CCR dependent behaviour is very interesting as it shows that the ILP runtimes do not only related to the size of the input problem in terms of number of tasks, edges and processors, but also depends on the weight values.



(a) Completed schedules of different formulations over CCR on 4 processors for 51 graph set for a 1 minute timeout



(b) Completed schedules of different formulations over CCR on 8 processors for 51 graph set for a 1 minute timeout

Figure 6: CCR Comparisons on 4 and 8 Processors Set for a 1 minute Timeout

B. MILP comparisons set for a 12 hour timeout

The 12 hour timeout experiments are for a direct comparison between runtimes of the MILP formulations. The input graphs used for benchmarking in this section are from [31],[38]. If the CPLEX solver is unable to find an optimal solution within 12 hours, the program is terminated and the results tabulated. The h:m:s notation is the standard hours:minutes:seconds taken by the MILP formulation to find an optimal solution. The graphs with a name starting with 't' were generated randomly and suffixed with the number of tasks in that graph followed by its edge density and the index used to distinguish graphs when they have the same n and ω values. The graphs with a name starting with 'ogra' are suffixed with the number of tasks followed by its edge density. These edge densities conform to the percentage definition of density in (11). According to [31], the optimal solution for 'ogra' graphs are obtained when the tasks are well packed (as in ideal schedule). This has similar characteristics with respect to a number of mutually independent tasks (that can be well packed as there are no communication delays), for which it is hard to find the task ordering which yields the optimal schedule. In Table IV, the first column gives the name of the graph, column n records the number of tasks in the graph. The symbols Ω and ω conform to the density definition in (10) and (11) respectively. Column p refers to the number of processors. The rest of the columns record the solution time for the MILP formulations being compared. Where the comparison table refers to the MILP formulation as **PACKING**, it implies that both PACKING-USUAL and PACKING-COMPACT in [31] are used for the experiments and the shorter of the two MILP runtimes is recorded. When the CPLEX solver is unable to find an optimal solution within 12 hours, the program is terminated and the gap returned by CPLEX is recorded. Let the feasible schedule length returned when the program terminates be FSL and the optimal schedule length (to be found) be OSL. The gap (in percentage) is a guarantee that the the difference between the FSL and OSL is at most FSL times the gap. I.e. $FSL-OSL \leq FSL \cdot gap$. If a solution is found within 12 hours, the gap is 0% and not recorded. If no feasible solution could be found within 12 hours, the gap is infinite and recorded as inf. For example, if the schedule length returned by the program when it terminates at the end of 12 hours is 200 units and the gap is 10%, then the optimal schedule length is guaranteed to be greater than or equal to 180 units.

Table IV compares the proposed formulations SHD-BASIC, SHD-RELAXED and SHD-REDUCED with PACKING, ILP-RBL and ILP-TC over 2, 4, 8 and 16 processors. The MILP formulation that yields the fastest result (or the least gap) is highlighted. It is seen that for most cases SHD-RELAXED or SHD-REDUCED runs faster than SHD-BASIC. Despite 'ogra' graphs having a special structure making it harder to solve optimally, it is seen that the proposed formulations have a significantly improved performance and outdo other formulations when more processors are available for scheduling. For the random 't' graphs, ILP-RBL runs fast over a smaller number of processors. ILP-TC for most cases displays an improved runtime with an increase in number of processors, but are found to run slower with an increase in the number of tasks in the graph.

It is seen from Table IV that 10 out of 16 column entries for the PACKING formulation timeout at 12 hours, indicating a much longer runtime over the proposed formulations. For most instances, SHD-RELAXED and SHD-REDUCED are many orders of magnitude faster than the PACKING formulation and ILP-TC. In fact, for many instances the runtime of the proposed formulations are seen to decrease when more processors are available for scheduling.

It is observed from Table IV that SHD-REDUCED has the overall best performance in terms of the number of fastest results found over all the MILP formulations compared. Absolute stability cannot be expected, this being the nature of ILP's. From the experiments carried out, it is seen that SHD-RELAXED and SHD-REDUCED are often an order of magnitude faster than the other formulations and hence a significant improvement of previous work. The experiments indicate that ILP performance is structure dependent and their performance degrades with high CCR.

VII. Conclusion

A MILP formulation for the task scheduling problem with communication delay was proposed. Each part of the formulation was motivated and discussed in detail, explaining its role and importance. A major feature of this formulation is the reduction of the number of variables and constraints by the effective linearisation of the bi-linear equation arising out of communication delays. Further, all variable indices in the MILP formulation are independent of the number of processors. As a result, the constraint complexity of the proposed MILP formulation was reduced to $O(|V|^2)$, which is significantly less than previous formulation as analysed

Table IV: 12 hour Timeout Comparisons on ILP Formulations

Graph	n	Ω	ω	p	PACKING	ILP-RBL	ILP-TC	BASIC	RELAXED	REDUCED
ogra20_55	20	5.2	55	2	*12h 32.09%*	*12h 26.15%*	*12h 35.40%*	***12h 25.78%***	*12h 30.01%*	*12h 26.58%*
				4	8m:49s	18s	21m:9s	**14s**	20s	27s
				8	35m:56s	6m:11s	11m:32s	22s	11s	**6s**
				16	*12h 3.92%*	1h:38m:5s	11m:46s	12s	15s	**10s**
t30_56_1	30	8.1	56	2	9s	**2s**	46m:26s	16s	5s	22s
				4	7m:32s	**17s**	5h:17m:51s	32s	21s	23s
				8	7h:22m:19s	*12h 0.21%*	2h:6m:54s	7m:38s	**48s**	2m:6s
				16	*12h 8.94%*	*12h 4.07%*	1h:11s	33s	19s	**15s**
t40_30_1	40	5.8	30	2	6m	**46s**	*12h 22.06%*	3h:49m:40s	3m:29s	5m:2s
				4	*12h 7.18%*	29m:33s	*12h 16.32%*	2h:50m:14s	10m:40s	**4m:25s**
				8	*12h 9.83%*	*12h 5.07%*	*12h 11.56%*	9h:36m:14s	23m:24s	**3m:34s**
				16	*12h 23.66%*	*12h 9.43%*	*12h 23.12%*	2h:12m:11	7m:49s	**4m:3s**
Ogra50_53	50	12.9	53	2	*12h 46.33%*	***12h 46.15%***	*12h inf*	*12h 48.50%*	*12h 48.27%*	*12h 48.22%*
				4	*12h 19.78%*	***12h 5.26%***	*12h inf*	*12h 12.94%*	*12h 15.76%*	*12h 12.02%*
				8	*12h inf*	*12h 2.46%*	*12h inf*	2h:0m:1s	33m:34s	**17m:17s**
				16	*12h inf*	*12h 5.58%*	*12h inf*	1h:23m:4s	**58m:2s**	2h:16m:48s

in detail. An optimisation of the proposed SHD-RELAXED formulation was developed through the investigation of redundant terms in its formulation. The proposed formulations were experimentally compared with several existing MILP formulations. The experimental results indicate that the proposed formulation (SHD-RELAXED) and its reduction (SHD-REDUCED) provide a drastic improvement in runtime over other MILP formulations. The analysis of the behaviour of the MILP formulations with respect to graph structures indicate that some structures perform better than the others. It was seen that an increase in CCR deteriorates the performance of all the MILP formulations and to conclude that MILP formulations do not scale well for high CCR graphs. It was also seen that a larger number of processors is not necessarily bad for the performance of the MILP formulation.

ACKNOWLEDGEMENT

We gratefully acknowledge that this work is supported by the Marsden Fund Council from Government funding, Grant 9073-3624767, administered by the Royal Society of New Zealand. We are thankful to the anonymous reviewers whose feedback helped in greatly improving the readability and quality of the paper.

REFERENCES

[1] V. Sarkar, *Partitioning and scheduling parallel programs for multiprocessors*. MIT press, 1989.

[2] W. Löwe and W. Zimmermann, "Scheduling Iterative Programs onto LogP-Machine," vol. 1685, pp. 332–339, 1999. Euro Par 99 Parallel Processing, Springer, Lecture Notes in Computer Science.

[3] A. Palmer and O. Sinnen, "Scheduling Algorithm Based on Force Directed Clustering," pp. 311–318, dec. 2008. Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on.

[4] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, pp. 653–670, 2005.

[5] A. Radulescu and A. van Gemund, "Low-cost task scheduling for distributed-memory machines," *Parallel and Distributed Systems, IEEE Transactions on*, vol. vol 13, pp. 648–658, jun 2002.

[6] O. Sinnen, *Task Scheduling for Parallel Systems (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2007.

[7] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Computing*, vol. 19, no. 12, pp. 1321–1344, 1993.

[8] A. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: comparative studies and performance issues," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, pp. 795–812, aug 1999.

[9] E. G. Coffman Jr. and R. L. Graham, "Optimal scheduling for two-processor systems," *Acta Informat.*, vol. 1, pp. 200–213, 1972.

[10] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Comput.*, vol. 18, no. 2, pp. 244–257, 1989.

[11] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," vol. 5, pp. 287–326, 1979.

[12] B. Veltman, B. J. Lageweg, and J. K. Lenstra, "Multiprocessor Scheduling with Communication Delays," vol. 16, no. 2-3, pp. 173–182, 1990.

[13] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.

[14] O. Sinnen and L. Sousa, "Experimental Evaluation of Task Scheduling Accuracy: Implications for the Scheduling Model," *IEICE Transactions on Information and Systems*, vol. E86-D, pp. 1620–1627, Sept. 2003.

[15] O. Sinnen and L. Sousa, "Scheduling Task Graphs on Arbitrary Processor Architectures Considering Contention," in *High Performance*

Computing and Networking (HPCN'01), vol. 2110 of *Lecture Notes in Computer Science*, pp. 373–382, Springer-Verlag, 2001.
[16] P. Crescenzi and V. Kann, "Approximation on the Web: A Compendium of NP Optimization Problems," in *RANDOM*, pp. 111–118, 1997.
[17] S. Fujita and M. Yamashita, "Approximation Algorithms for Multiprocessor Scheduling Problem," *IEICE Transactions on Information and Systems*, vol. 83, pp. 503–509, 2000.
[18] M. Drozdowski, *Scheduling for Parallel Processing*. Springer Publishing Company, Incorporated, 1st ed., 2009.
[19] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times," *SIAM J. Comput.*, vol. 18, no. 2, pp. 244–257, 1989.
[20] Y.-K. Kwok and I. Ahmad, "On multiprocessor task scheduling using efficient state space search approaches," *Journal of Parallel and Distributed Computing*, vol. 65, no. 12, pp. 1515–1532, 2005.
[21] A. Z. Semar Shahul and O. Sinnen, "Scheduling task graphs optimally with A*," *Journal of Supercomputing*, vol. 51, pp. 310–332, Mar. 2010.
[22] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *J. ACM*, vol. 32, pp. 505–536, July 1985.
[23] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice hall, 2010.
[24] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, "Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling," Tech. Rep. UCB/EECS-2006-166, EECS Department, University of California, Berkeley, Dec 2006.
[25] C. A. Floudas and X. Lin, "Mixed Integer Linear Programming in Process Scheduling: Modeling, Algorithms, and Applications," *Annals of Operations Research*, vol. 139, no. 1, pp. 131–162, 2005.
[26] "Chapter 3 Parallel machines - Linear programming and enumerative algorithms," *Annals of Operations Research*, vol. 7, pp. 99–132, 1986.
[27] P. E. Coll, C. C. Ribeiro, and C. C. de Souza, "Multiprocessor scheduling under precedence constraints: Polyhedral results," *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 770–801, 2006. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.
[28] A. Bender, "MILP Based Task Mapping for Heterogeneous Multiprocessor Systems," in *in Proceedings European Design Automation Conference*, pp. 190–197, IEEE, 1996.
[29] T. Davidović, L. Liberti, N. Maculan, and N. Mladenović, "Mathematical programming-based approach to scheduling of communicating tasks," tech. rep., 2004.
[30] N. Maculan, S. C. S. Porto, C. C. Ribeiro, C. C. de Souza, R. Cid, and C. Souza, "A New Formulation for Scheduling Unrelated Processors under Precedence Constraints," *RAIRO Operations Research*, vol. 33, pp. 87–91, 1997.
[31] T. Davidović, L. Liberti, N. Maculan, and N. Mladenovic, "Towards the Optimal solution of the Multiprocessor Scheduling Problem with Communication Delays.," in *3rd Multidisciplinary International Conference on Scheduling: Theory and Application.*, pp. 128–135, 2007.
[32] S. Venugopalan and O. Sinnen, "Optimal Linear Programming Solutions for Multiprocessor Scheduling with Communication Delays," in *Algorithms and Architectures for Parallel Processing* (Y. Xiang, I. Stojmenovic, B. Apduhan, G. Wang, K. Nakano, and A. Zomaya, eds.), vol. 7439 of *Lecture Notes in Computer Science*, pp. 129–138, Springer Berlin Heidelberg, 2012.
[33] C. C. Price and U. W. Pooch, "Search techniques for a nonlinear multiprocessor scheduling problem," *Naval Research Logistics Quarterly*, vol. 29, no. 2, pp. 213–233, 1982.
[34] L. Liberti, "Compact linearization for binary quadratic problems," *4OR*, vol. 5, pp. 231–245, 2007.
[35] L. Liberti, "Compact linearization for bilinear mixed-integer problems," tech. rep., 2005.
[36] Y. Guan and R. Cheung, "The berth allocation problem: models and solution methods," in *Container Terminals and Automated Transport Systems*, pp. 141–158, Springer Berlin Heidelberg, 2005.
[37] "ILOG CPLEX 11.0 User's Manual," ILOG S.A., Gentilly, France, 2007.
[38] T. Davidović and T. G. Crainic, "Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems," *Comput. Oper. Res.*, vol. 33, pp. 2155–2177, August 2006.
[39] S. Venugopalan and O. Sinnen., "Green banana task scheduler for multiprocessor systems.," URL (Case Sensitive): http:// homepages.engineering.auckland.ac.nz/ ˜parallel/ OptimalTaskScheduling/.
[40] R. Hull and A. Winter, "A short introduction to the gxl software exchange format," in *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pp. 299–301, 2000.

S arad Venugopalan received his Bachelor degree in Computer Science and Engineering from Siddaganga Institute of Technology, Karnataka, India and a Master (by Research) degree in Wireless Communication from Madras Institute of Technology, Anna University Chennai, India. He is a Marsden Fund scholar and pursuing his PhD at the University of Auckland, New Zealand working on Optimal Task Scheduling for Parallel Systems. His other research interests include computer algorithms and cryptology.

O liver Sinnen received his Diploma degree in electrical and computer engineering from RWTH Aachen University, Germany. He did his PhD studies at Instituto Superior Técnico (IST), Technical University of Lisbon, completed in 2003. Since 2004, he has been a (Senior) Lecturer with the Department of Electrical and Computer Engineering at the University of Auckland, New Zealand. He authored the book "Task Scheduling for Parallel Systems", published by Wiley. His research interests include parallel computing and programming, scheduling and reconfigurable computing.