Satisfiability Modulo Theory (SMT) Formulation for Optimal Scheduling of Task Graphs with Communication Delay

Avinash Malik[†], Cameron Walker[‡], Michael O'Sullivan[‡], Oliver Sinnen[†]

[†]*Department of Electrical and Computer Engineering*
[‡]*Department of Engineering Science*
University of Auckland,
Auckland, New Zealand

Abstract

In scheduling theory and practise for parallel computing, representing a program as a task graph with communication delays is a popular model, due to its general nature, its expressiveness and relative simplicity. Unfortunately, scheduling such a task graph on a set of processors in such a way that it achieves its shortest possible execution time ($P|pred, c_{ij}|C_{max}$ in $\alpha|\beta|\gamma$ notation) is a strong NP-hard optimization problem without any known guaranteed approximation algorithm. Hence, many heuristics have been researched and are used in practise. However, in many situations it is necessary to obtain optimal schedules, for example, in the case of time-critical systems or for the evaluation of heuristics. Recent years have seen some advances in optimal algorithms for this scheduling problem, based on smart exhaustive state-space search or MILP (Mixed Integer Linear Programming) formulations. This paper proposes a novel approach based on SMT (Satisfiability Modulo Theory). We propose an elegant SMT formulation of the scheduling problem that only needs one decision variable and is very compact and comprehensible in comparison to the state-of-the-art MILP formulations. This novel optimal scheduling approach is extensively evaluated in experiments with more than a thousand task graphs. We perform experimental comparison

Email address: `{avinash.malik, cameron.walker, michael.osullivan, o.sinnen}@auckland.ac.nz` (Avinash Malik[†], Cameron Walker[‡], Michael O'Sullivan[‡], Oliver Sinnen[†])

with the best known MILP formulations, with attempts to further improve them, and deeply analyse the behaviour of the different approaches with respect to size, structure, number of processors, etc. Our proposed SMT-based approach in general outperforms the MILP-formulations and still possesses great potential for further optimization, from which MILP formulations have benefited in the past.

1. Introduction

Task scheduling has long been recognized as a crucial part of parallel computing. In any parallel computation, tasks need to be mapped to the available processors and ordered for execution. In the formalisation of this process, a directed acyclic graph (DAG) – a task graph – represents the program to be executed on a parallel system. The nodes or vertices of this graph represent the (sub-)tasks and the edges between them reflect the communication (data transfer) or dependencies between them. The weighted DAG representation, a very universal and flexible model, has been widely used in literature and practice to create schedules on a given set of processors [1, 2]. It has also seen an increasing interest in High Performance Computing in dynamic runtime schedulers such as StarPU [3], KAAPI [4], StarSS [5], and PaRSEC [6]. Even the OpenMP standard now includes DAG scheduling constructs [7].

Unfortunately, the optimization problem of minimising the execution time of a program, modelled as a weighted DAG, on a set of processors is a well known NP-hard problem [8], for all but some very simplified cases, e.g. a fork or a join graph on an unlimited number of processors [9], or a tree with unit weights on two processors [10]. For that reason, many scheduling algorithms have been proposed in the past. The largest category of algorithms is based on list scheduling [11, 12, 13, 14]. Other algorithms are based on clustering, where a second phase maps the clusters onto the limited number of processors [15, 16, 17, 18]. Also meta-heuristics, such a genetic algorithms have been applied to task scheduling [19, 20]. In the general case there is no guaranteed constant approximation factor, only one based on the communication costs of the graph [21]. Some scheduling algorithms might even produce schedules that are longer than the sequential schedule, where all tasks are executed on the same processors [1]. This is especially true for large communication costs.

Given the importance of this task scheduling problem and the lack of guaranteed approximation algorithms, an attempt has been made to design algorithms and approaches to optimally solve this scheduling problem. An optimal solution can be crucial for critical systems, where the scheduled application is executed many times. A very important usage of optimal schedules is the evaluation of heuristic scheduling algorithms, given the lack of guaranteed optimality from heuristics. The quality of heuristic algorithms can be judged better when the optimal solutions are available for comparison. As a combinatorial optimization problem, the task scheduling problem can be tackled with various approaches. Searching through the exhaustive solution space is one method that has been proposed, using algorithms such as branch-and-bound or A* [22, 23, 24, 25, 26]. Another approach is using (Mixed) Integer Linear Programming (MILP) formulations with corresponding solvers. Despite the difficulty of solving scheduling problems with MILP formulations, there has been some significant progress recently [27, 28, 29]. As a natural generalization, constraint programming has been applied to solving the task scheduling problem [30]. Another recent approach is to use a SAT (Boolean Satisfiability Problem) formulation to solve the scheduling problem [31].

In this paper we propose an approach based on a Satisfiability Modulo Theory (SMT) formulation. The formulation is simple and elegant as it is mostly based on the constraints that naturally emerge from the scheduling model. We only employ one binary decision variable to determine whether a task is scheduled on a given processor or not. In terms of theory solver for the inequalities, we employ Quantifier Free Linear Arithmetic Logic (QF_LRA).

To evaluate the performance and usefulness of the proposed SMT formulation, we compare it with state-of-the-art MILP formulations of the same problem [29, 32, 33]. In an attempt to further improve these formulations, we propose new variants that address the typical symmetry issue of MILP formulations for scheduling problems.

The experimental evaluation is based on a large set of 120 task graphs, which are scheduled with all the SMT and MILP formulations onto different numbers of processors. These graphs cover different structures, sizes and weightings. As solvers for the SMT and MILP formulations we use Z3 [34] and Gurobi [35], respectively. We perform a thorough analysis of the results, uncovering performance behaviour of the formulations with respect to structure and other characteristics of the task graphs and the processor number. Two types of statistical classification are carried out, namely logistic regression and decision tree classification. While some of the obtained results are intuitive and expected (e.g. larger graphs are more difficult to schedule optimally), other results are unexpected and can lead to further improvement in the future. The evaluation shows that the SMT formulation outperforms the MILP based approaches, especially for a lower number of processors, solving more problems optimally within the given time limit. This holds a lot of promise for the novel SMT approach, given the potential of optimization, which MILP formulations have already enjoyed.

The remainder of this paper is organized as follows. Section 2 defines the scheduling model and problem. We propose our SMT formulation of the problem in Section 3 and revisit MILP formulations in Section 4. The experimental evaluation of all approaches is conducted and discussed in Section 5. We study related work in Section 6 and conclude the paper in Section 7.

2. Scheduling model

A task graph $G(V, E)$ is a directed acyclic graph (DAG), where $V = \{v_1, ..., v_n\}$ is the set of computation tasks, and $E \subseteq V \times V$ is the set of data dependency constraints (or communications) between tasks. A homogeneous multiprocessor platform $P = \{p_1, ..., p_m\}$ consists of m identical processors connected by a communication network. Each processor is connected to every other processor. Furthermore we also assume a dedicated communication sub-system so that computation and communication can be executed in parallel. Tasks are executed sequentially without preemption on a processor. Every task is able to run on any processor. Execution Time (ET) of task $v \in V$ is given by $t(v)$, and Communication Latency (CL) for edge $(v_i, v_j) \in E$ is given by $c(v_i, v_j)$. If two tasks, with data dependence, are mapped to the same processor, inter-task communication is implemented by data sharing in local memory, and no communication latency is incurred. If two tasks, with data dependence, are mapped to different processors, communication between them is implemented by data transfer through communication links, and communication latency corresponds to the CL (e.g., $c(v_i, v_j), (i, j) \in E$). We define $deg^-(v)$ ($deg^+(v)$) as being the in-degree (out-degree) of the vertex v, i.e. the number of entering (leaving) edges. In this work, we assume the ET values of tasks and CL values of intertask communications are known, and focus on optimizing the mapping and scheduling of a task graph under these assumptions.

Given a task graph $G(V, E)$ and a homogeneous multi-processor platform P, the optimization problem is to find a mapping $M : V \to P$ for each task in V to a processor in P and a schedule $S : V \to \mathbb{N}$ for the tasks assigned to processors, where each task v in V mapped to a processor in P is assigned a natural number indicating its starting time s for execution on P. This schedule (from now on the term schedule implies both the mapping and the start time assignment to tasks) must adhere to two constraints resulting from the above definitions. The *processor constraint*, meaning that for any two tasks on the same processor, one task must finish before the other one starts. And the *precedence constraint*, which enforces that for any edge $(v_i, v_j) \in E$, task v_j can only start after task v_i is completed (i.e. $s(v_j) \geq s(v_i)+t(v_i)$) plus the communication time, if the tasks are mapped to different processors (i.e. $s(v_j) \geq s(v_i) + t(v_i) + c(v_i, v_j)$). The goal of the optimization is to minimize the *makespan* $m(G)$. Makespan, also called schedule length, is defined as the time difference between the start time of the earliest task and the finish time of the latest one. Assuming that the earliest task's start time is 0, the makespan is given by $m(G) = max_{v \in V}[s(v) + t(v)]$. The problem of finding the schedule that minimizes the makespan is known to be NP-hard [8].

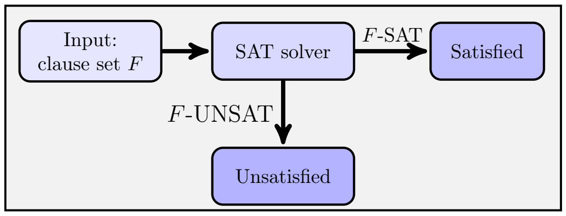
3. Satisfiability Modulo Theory (SMT) formulation

Satisfiability Modulo Theory (SMT) [36] is a technique for solving boolean constraint problems modulo theories. SMT has been used successfully in verification of hardware and software programs. Figure 1 shows the difference between a Satisfiability (SAT) solver and a SMT solver. A SAT solver is used to solve propositional formulas. These formulas consist of atomic propositions (or their logical negation), in the Boolean domain, combined using logical connectives such as $\vee$ (disjunction), and $\wedge$ (conjunction). A SAT solver uses the well known Davis-Putnam-Logemann-Loveland (DPLL) [37] solving technique for finding a *model*, i.e., an assignment for atomic propositions that satisfies the propositional formula, if one exists. A SMT solver extends a SAT solver with theory solvers in order to solve more expressive satisfiability problems such as those in the domain of bit-vectors, integers, etc. The general approach (see Figure 1b) of SMT solvers is as follows:

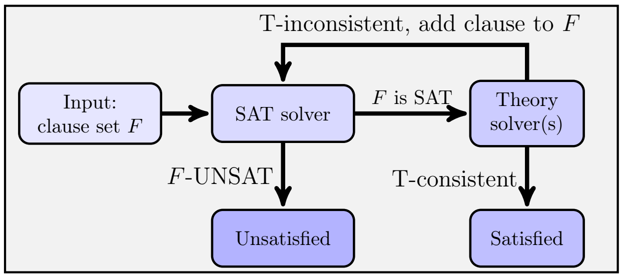
1. Step-1: Translate the non-propositional expressions into atomic propositions.
2. Step-2: Use the SAT solver to find an assignment for the atomic propositions, if one exists, that satisfies the propositional formula.

3. Step-3: Determine that the satisfying assignments are consistent with the theory.
4. Step-4(a): If consistent with the theory (denoted as T-consistent) then we have satisfied the SMT formulation.
5. Step-4(b): If inconsistent with the theory (denoted T-inconsistent) then we add a negation of the clause to the propositional formula and iterate from steps 1 to 4. This procedure is carried out until the SMT formula is satisfied or the whole search space is explored and no satisfying assignment exists.

Let us consider a pedagogical SMT formulation in Equation (1) to elucidate the working of the SMT solver. Applying Step-1 gives the propositional formula shown in Equation (2). In this translation, each of the linear inequalities are replaced by atomic propositions. One possible assignment for these atomic propositions that satisfies the propositional formula, given by the SAT solver, in shown in Equation (3). Next, in Step-4, we need to check if these assignments are consistent with the theory of linear inequalities – obtained using a theory solver (e.g., simplex algorithm in case of linear inequalities). For, Equation (3) to be true, variables x and y should be assigned values such that $x + y \leq 5$, $y \geq 5$ and $x > 5$ hold. There exists no assignments for



(a) A general SAT solver framework



(b) A general SMT solver framework

Figure 1: Comparison between SAT and SMT solvers

the variables x and y, where these three linear inequalities hold. Hence, the theory solver gives back the result as T-inconsistent, with atomic proposition R being negated and added to the propositional formula. The resultant propositional formula, and the consequent assignments to the propositions are shown in Equation (4). The theory solver is called again, and in this instance the linear inequalities $x + y \leq 5$, $y \geq 5$ and $x \geq 0$ can hold for the assignment $x \leftarrow 0$ and $y \leftarrow 5$, which makes the final result satisfiable. The final assignment to x and y (called the *model*) is shown in Equation (5).

$$(x + y \leq 5) \wedge (y \geq 5) \wedge ((x > 5) \vee (x \geq 0)) \tag{1}$$
$$P \wedge Q \wedge (R \vee S) \tag{2}$$
$$P \leftarrow \mathrm{T}, Q \leftarrow \mathrm{T}, R \leftarrow \mathrm{T} \tag{3}$$
$$P \wedge Q \wedge (\neg R \vee S); P \leftarrow \mathrm{T}, Q \leftarrow \mathrm{T}, S \leftarrow \mathrm{T} \tag{4}$$
$$x \leftarrow 0, y \leftarrow 5 \tag{5}$$

In this paper we provide the first ever SMT formulation for solving the task-scheduling problem. An SMT formulation consists of a number of constraints, that need to be satisfied by the SMT solver. To formulate all necessary constraints we employ the definitions of our scheduling problem in Section 2. We only need to introduce one set of binary decision variables $b(v, p)$, which is true if task v is scheduled on processor p and false otherwise. Formally, $\forall v \in V, \forall p \in P$:

$$b(v,p) = \begin{cases} 1 & \text{if } v \text{ is mapped to } p \\ 0 & \text{otherwise} \end{cases}$$

The SMT formulation is then defined as given in Figure 2. The implication operator is indicated as $\supset$. Logical implication ($A \supset B$) is equivalent to $\neg A \vee B$, in classical logic. For the task-scheduling problem, the constraints are as follows: constraint (A) guarantees that every task is allocated and executed *only* on a single processor.

Constraint (S) guarantees that all source vertices, i.e., vertices with no incoming edges have a start time of at least zero. Given an edge $(v_i, v_j) \in E$, constraints ($C1$) and ($C2$) together guarantee that task v_j can only start execution after execution of task v_i. Furthermore, constraint ($C1$) states that if these tasks are assigned to different processors, then some communication latency is incurred to transfer data from v_i to v_j. Constraint $(C2)$ on the other hand states that if these two tasks are assigned to the same processor then no communication costs are incurred.

$$
\begin{array}{lll}
objective & Min(m(G)) & \\
st: & & \\
(A) & \forall v \in V & \bigvee_{\forall p \in P} b(v,p) \\
(S) & \forall v \in V : deg^{-}(v) = 0 & s(v) \geq 0 \\
(C1) & \forall (v_i, v_j) \in E, \forall p_k, \in P & b(v_i, p_k) \wedge b(v_j, p_l) \supset s(v_j) \geq s(v_i) + t(v_i) + c(v_i, v_j) \\
 & p_l \in P, p_k \neq p_l & \\
(C2) & \forall (v_i, v_j) \in E, \forall p \in P & b(v_i, p) \wedge b(v_j, p) \supset s(v_j) \geq s(v_i) + t(v_i) \\
(R) & \forall v_i, v_j \in V, (v_i, v_j) \notin E^T, & b(v_i, p) \wedge b(v_j, p) \supset (s(v_i) \geq s(v_j) + t(v_j)) \vee (s(v_j) \geq s(v_i) + t(v_i)) \\
 & (v_j, v_i) \notin E^T, \forall p \in P & \\
(M) & & m(G) = max_{v \in V}[s(v) + t(v)] - min_{v \in V} s(v) \\
\text{where} & & \\
 & t(v) \in \mathbb{R}^{\geq 0} & \\
 & c(v_i, v_j) \in \mathbb{R}^{\geq 0} & \\
 & G^T = (V, E^T) & \text{is transitive closure of } G
\end{array}
\tag{6}
$$

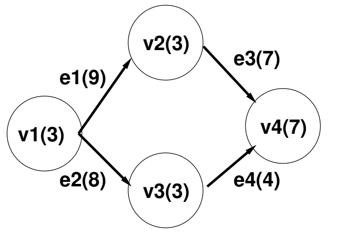
Figure 2: The Satisfiability Modulo Theory (SMT) formulation

Constraint (R) states that if two tasks $v_i \in V$ and $v_j \in V$ allocated to the same processor $p \in P$ and do not have a dependency edge between them, then either can be scheduled for execution before the other. Moreover, the task that is scheduled second is delayed by ET of the first task. Finally, constraint (M) gives the makespan of the task-graph. The SMT formulation in Figure 2 is solved iteratively to find the optimal solution to the task-scheduling problem. We use the *Quantifier Free Linear Arithmetic Logic* (QF_LRA) theory with Boolean constraints. We perform binary search for satisfaction between:

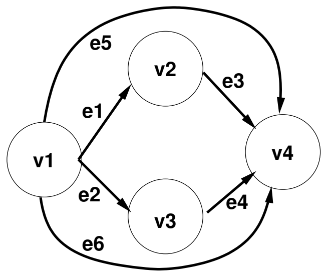
$$
\left[max_{\forall v \in V} t(v), \sum_{\forall v \in V} t(v) \right] \tag{7}
$$

In Equation (7), the lower bound in the closed interval gives the *makespan* of the task-graph, assuming a load balanced allocation of $|V|$ tasks onto $|V|$ processors, and ignoring communication costs. The upper bound in Equation (7), is obtained by assuming all $|V|$ tasks are allocated on a single processor.

An important point to note is that the SMT formulation only uses disjunction, *not* exclusive OR for allocating tasks onto processors in constraint (A). The SAT solver (a part of SMT solver) expects the problem to be in



(a) An example task graph with ET and CL in brackets



(b) Transitive closure of the example task graph in Figure 3a

Figure 3: An example task graph and its transitive closure

so called *Conjunctive Normal Form* (CNF). Translating a disjunction into the CNF form results in smaller number of terms than translating an exclusive OR into the CNF form. Hence, disjunction is preferable. Constraint (A) is meant to allocate a task onto a single processor, however, consider a given task v and (say) two processors $p1$ and $p2$, one possible solution, for constraint (A) is that both variables $b(v, p1)$ and $b(v, p2)$ are true for proposition $b(v, p1) \vee b(v, p2)$ to hold. Such cases do *not* affect the makespan and schedule due to the following observations:

1. If a task duplicated elongates the makespan, given that all computation times are positive, the minimization objective will always find a smaller makespan (and related schedule), by not duplicating the task.
2. If duplicated tasks do not affect the schedule, e.g., in cases of independent task graphs, they can be trivially removed from the final schedule by picking the earliest starting instance and breaking ties randomly.
3. It is not possible that the duplication of tasks permits the creation of schedules with a shorter makespan than the corresponding schedule without duplication (see previous point), because the (C) constraints are constructed for all duplicated parent-child pairs, hence duplication cannot remove communication costs. For example, a task w depends on a task v, which is allocated to $p1$ and $p2$. Then, enforced by the (C) constraints, the start time of w needs to be after the finish time of v on $p1$ plus communication costs (if remote) *and* after the finish time of v on $p2$ plus communication costs (if remote). In other words, it is never an advantage to duplicate tasks.

$$\begin{array}{rl}
(A) & b(v1,p1) \vee b(v1,p2) \\
 & \wedge b(v2,p1) \vee b(v2,p2) \\
 & \wedge b(v3,p1) \vee b(v3,p2) \\
 & \wedge b(v4,p1) \vee b(v4,p2) \\
\wedge & \\
(S) & s(v1) \geq 0 \\
\wedge & \\
(C1) & b(v1,p1) \wedge b(v2,p2) \supset s(v2) \geq s(v1) + 3 + 9 \\
 & \wedge b(v1,p2) \wedge b(v2,p1) \supset s(v2) \geq s(v1) + 3 + 9 \\
 & \wedge b(v2,p1) \wedge b(v4,p2) \supset s(v4) \geq s(v2) + 3 + 7 \\
 & \wedge b(v2,p2) \wedge b(v4,p1) \supset s(v4) \geq s(v2) + 3 + 7 \\
 & \wedge b(v1,p1) \wedge b(v3,p2) \supset s(v3) \geq s(v1) + 3 + 8 \\
 & \wedge b(v1,p2) \wedge b(v3,p1) \supset s(v3) \geq s(v1) + 3 + 8 \\
 & \wedge b(v3,p1) \wedge b(v4,p2) \supset s(v4) \geq s(v3) + 3 + 4 \\
 & \wedge b(v3,p2) \wedge b(v4,p1) \supset s(v4) \geq s(v3) + 3 + 4 \\
\wedge & \\
(C2) & b(v1,p1) \wedge b(v2,p1) \supset s(v2) \geq s(v1) + 3 \\
 & \wedge b(v1,p2) \wedge b(v2,p2) \supset s(v2) \geq s(v1) + 3 \\
 & \wedge b(v2,p1) \wedge b(v4,p1) \supset s(v4) \geq s(v2) + 3 \\
 & \wedge b(v2,p2) \wedge b(v4,p2) \supset s(v4) \geq s(v2) + 3 \\
 & \wedge b(v1,p1) \wedge b(v3,p1) \supset s(v3) \geq s(v1) + 3 \\
 & \wedge b(v1,p2) \wedge b(v3,p2) \supset s(v3) \geq s(v1) + 3 \\
 & \wedge b(v3,p1) \wedge b(v4,p1) \supset s(v4) \geq s(v3) + 3 \\
 & \wedge b(v3,p2) \wedge b(v4,p2) \supset s(v4) \geq s(v3) + 3 \\
\wedge & \\
(R) & b(v2,p1) \wedge b(v3,p1) \supset s(v2) \geq s(v3) + 3 \\
 & \quad \vee s(v3) \geq s(v2) + 3 \\
 & \wedge b(v2,p2) \wedge b(v3,p2) \supset s(v2) \geq s(v3) + 3 \\
 & \quad \vee s(v3) \geq s(v2) + 3 \\
\wedge & \\
(M) & m(G) = s(v4) + 7 - s(v1)
\end{array}$$

Figure 4: SMT formulation for task graph in Figure 3a

Let us consider a simple task graph as shown in Figure 3a to describe the SMT formulation from Figure 2 and its working. In the task graph in Figure 3a, the vertices are labeled from $v1$ through to $v4$, while the edges are labeled from $e1$ through to $e4$. The ET of vertices and the CL of the edges, in time units, are given in brackets.

The SMT formulation from Figure 2 applied to the task graph in Figure 3a, for a two processor system, is shown in Figure 4. The first thing to note is that the formulation applied to any task graph is free of quantifiers, (e.g., $\forall$ and $\exists$) with all the quantifiers unrolled. Constraint (A), in Figure 4, states that each vertex in the task graph can only be allocated to a single processor. If we consider vertex $v1$, then only $b(v1, p1)$ or $b(v1, p2)$ can be assigned true in the optimal solution. Should both be assigned true in a solution, which is possible by the constraints, then either that solution will be rejected later as non-optimal (a better solution where only one b value is true exists) or this duplication has no influence on the makespan and either task-to-processor allocation can be selected in the final schedule produced by the solver. In the task graph in Figure 3a only vertex $v1$ has in degree 0, and hence constraint (S) is applied to only vertex $v1$. The first two linear inequalities in constraints $(C1)$ and $(C2)$, in Figure 4, correspond to edge $e1$ in the task-graph. If the problem is feasible, only one of these linear inequalities can hold for any given solution because of constraint (A). These inequalities guarantee two properties for edge $e1$:

- Vertex $v2$ can only start processing, indicated by the postponement of the start time $(s(v2))$ of vertex $v2$, after vertex $v1$, due to the precedence constraint enforced by the edge $e1$.

- If vertices $v1$ and $v2$ are allocated to different processors, then according to the first two inequalities in $(C1)$ the start time of vertex $v2$ $(s(v2))$ is postponed by the ET of vertex $v1$, which is 3 times units. Furthermore, CL, of 9 time units, is incurred for transferring data from processor $p1$, which is executing $v1$, to $p2$, which is executing $v2$ (or vice-versa). On the other hand if $v1$ and $v2$ are assigned to the same processor (e.g., $p1$ in the first expression of $(C2)$) then, the start time of $v2$ is only delayed by the ET of $v1$.

Constraints $(C1)$ and $(C2)$ work in the same way for all the edges in the task-graph. Now consider the transitive closure of the task-graph shown in Figure 3b. As we can see from the transitive closure, there is no path (edge) from vertex $v2$ to $v3$ *and* vice-versa. If these two vertices are allocated to the same processor, we need to decide in what order we should execute them. Constraint (R), in Figure 4, solves this ordering dilemma. Constraint (R) states that if both these vertices are allocated onto processor $p1$ or $p2$, then either one can be scheduled first, and the other gets delayed by its

ET. Finally, constraint (M) gives the makespan for the task-graph. For this pedagogical example, we know that vertex $v1$ starts earliest and vertex $v4$ will start latest in the schedule, obvious from the precedence constraints enforced by the edges. Hence, the makespan of the schedule of the example task-graph is the difference between the end of computation of vertex $v4$ and the start time of the vertex $v1$. Once all these constraints are generated, for any task-graph, they are anded together to form the overall SMT formulation.

4. Mixed Integer Linear Programming (MILP) formulation and variants

In order to evaluate the performance of our proposed SMT based approach, we will compare it with the performance of the best previously proposed Mixed Integer Linear Programming (MILP) formulations [29, 33, 32]. Like SMT, MILP formulations are a general approach to solve combinatorial optimization problems. As such, a comparison with this widely used approach will position the performance of our proposed SMT approach. MILP solvers have been widely used for scheduling problems and we will reference and discuss related work in Section 6. The MILP formulations we consider fall into two categories: *packing* and *non-packing*. We revisit the packing MILP formulations [29] in this section and then propose some variants to address the symmetry issue common in scheduling problems. We do not modify the non-packing formulations [33, 32], so their formulations are omitted for brevity but their performance is evaluated in Section 5.

4.1. MILP Formulations

Venugopalan and Sinnen [29] proposed 3 MILP formulations (ILP-BASIC, ILP-RELAXED, and ILP-REDUCED) that are variations of each other. We present ILP-BASIC and then indicate where the other two formulations differ. Note that we have modified the notation from [29] to match our notation in Section 3, but we retain their equation numbering to aid the reader. For, the DAG $G(V, E)$, which has the set of computational tasks $V = \{v_1, \ldots, v_n\}$ and communication edges $E \subseteq V \times V$ with ET for the tasks given by $t(v), v \in V$ and the CL given by $c(v_i, v_j), e = (v_i, v_j) \in E$. The DAG is being scheduled on a homogeneous multiprocessor platform $P = \{p_1, \ldots, p_m\}$ of m identical processors. The decision variables are the processor indices

$$p(v) \in \{1, \ldots, m\}, \forall v \in V \tag{A12}$$

i.e., the index of the processor that v is scheduled on, the starting time variables

$$s(v) \geq 0, \forall v \in V \tag{A13}$$

the makespan variable

$$m(G) \geq 0 \tag{A14}$$

and two "overlap" variables

$$\sigma(v_i, v_j) = \begin{cases} 1 & \text{if } v_i \text{ finishes before } v_j \text{ starts} \\ 0 & \text{otherwise} \end{cases}, \forall v_i \in V, \forall v_j \in V \tag{A11a}$$

$$\epsilon(v_i, v_j) = \begin{cases} 1 & \text{if the processor index of } v_i \text{ is} \\ & \text{less than processor index of } v_j \\ 0 & \text{otherwise} \end{cases}, \forall v_i \in V, \forall v_j \in V \tag{A11b}$$

The objective of all the formulations is to minimise the makespan of the schedule, i.e.,

$$\min m(G) \tag{A01}$$

The makespan is the maximum of the starting times plus the execution times of the tasks

$$s(v) + t(v) \leq m(G), \forall v \in V \tag{A02}$$

If tasks are different either one starts before the other or they start at the same time

$$\sigma(v_i, v_j) + \sigma(v_j, v_i) \leq 1, v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \tag{A03}$$

If the tasks are different either one processor index is less than the other or they are on the same processor

$$\epsilon(v_i, v_j) + \epsilon(v_j, v_i) \leq 1, v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \tag{A04}$$

If the tasks are different then either one finishes before the other or they are on different processors

$$\sigma(v_i, v_j) + \sigma(v_j, v_i) + \epsilon(v_i, v_j) + \epsilon(v_j, v_i) \geq 1, v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \tag{A05}$$

If tasks are different then their processor indices and their ϵ variable are linked

$$p(v_j) - p(v_i) - 1 - m\,(\epsilon(v_i, v_j) - 1) \geq 0, v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \quad \text{(A06)}$$
$$p(v_j) - p(v_i) - m\,\epsilon(v_i, v_j) \leq 0, v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \quad \text{(A07)}$$

(Recall that $m = |P|$.) The maximum value of the makespan is calculated

$$\overline{m}(G) = \sum_{\forall v \in V} t(v) + \sum_{\forall (v_i, v_j) \in E} c(_i, v_j) \quad \text{(A15)}$$

and is used to constrain task starting times depending on processor indices

$$s(v_i) + t(v_i) + \overline{m}(G)(\sigma(v_i, v_j) - 1) \leq s(v_j), v_i \neq v_j, \forall v_i \in V, \forall v_j \in V \quad \text{(A08)}$$
$$s(v_i) + t(v_i) + c(v_i, v_j)(\epsilon(v_i, v_j) + \epsilon(v_j, v_i)) \leq s(v_j), (v_i, v_j) \in E \quad \text{(A09)}$$

If task j comes after task i in the DAG, then it must start after task i finishes

$$\sigma(v_i, v_j) = 1, \forall (v_i, v_j) \in E \quad \text{(A10)}$$

In ILP-RELAXED, Equation (A07) is relaxed. According to [29] it is included to model communication delays. As communication only occurs between $(v_i, v_j) \in E$, $\epsilon(v_i, v_j)$ only needs to be calculated for $(v_i, v_j) \in E$ so Equation (A07) is replaced with

$$p(v_j) - p(v_i) - m\,\epsilon(v_i, v_j) \leq 0, \forall (v_i, v_j) \in E \quad \text{(A07a)}$$
$$p(v_i) - p(v_j) - m\,\epsilon(v_j, v_i) \leq 0, \forall (v_i, v_j) \in E \quad \text{(A07b)}$$

In ILP-REDUCED, redundant constraints are removed from ILP-RELAXED. Equation (A03) is redundant because of Equation (A08) and Equation (A04) is redundant because of Equation (A06), so ILP-REDUCED is the ILP-RELAXED formulation without Equations (A03) and (A04).

4.2. Breaking Symmetry

We recognized that the MILP formulations (see Section 4.1) will suffer from solution symmetry, e.g., consider a problem with $V = \{v_1, v_2\}$, $E = \emptyset$, and $P = \{p_1, p_2\}$. Then a solution to the MILP formulation could schedule

v_1 on p_1 and v_2 on p_2 (to get a makespan $= \max\{t(v_1), t(v_2)\}$). However, a different solution with an identical makespan would be to schedule v_1 on p_2 and v_2 on p_1. There is symmetry in the scheduling of tasks to the processors because the processors in P are identical. If the processor indices are randomly shuffled, then a different solution to the MILP formulation with the same makespan will be created. To guarantee optimality of a solution, a MILP solver needs to explore every one of these symmetrical solutions, i.e., $m!$ solutions. This means that the MILP approaches will not perform well as m increases.

One simple, yet often effective, method to break these symmetries is to perturb the solutions' objective values so that there is: 1) enough of a difference in objective function so the that MILP solver will stop once it has identified 1 of the symmetrical solutions; and 2) not enough of a difference that the unperturbed optimal solution will be different. For the example with 2 tasks and 2 processors, consider the extra information that both tasks take 10ms to execute. Hence, the minimum makespan is 10 whether v_1 is scheduled on p_1 and v_2 is scheduled on $p2$ or vice versa. However, consider adding a perturbation to the objective function based on the processor index for each task, i.e.,

$$\min m(G) + \rho(i)p(v_i) \tag{A01+}$$

If, for our example, we set $\rho(i) = i, v_i \in V$, i.e., $\rho(1) = 1, \rho(2) = 2$ then the objective function value of scheduling v_1 to p_1, i.e., $p(v_1) = 1$, and v_2 to p_2, i.e., $p(v_2) = 2$, is $10 + 1 \times 1 + 2 \times 2 = 15$. We call this the direct solution. However, the objective function value of scheduling v_1 to p_2, i.e., $p(v_1) = 2$, and v_2 to p_1, i.e., $p(v_2) = 1$, is $10 + 1 \times 2 + 2 \times 1 = 14$. We call this the swap solution. Hence, the MILP solver will be directed to the swap solution and its preferable objective function value means the direct solution won't be explored. However, as MILP solvers iteratively remove fractionality until an integer solution is found, these solvers may spend some time exploring the fractional region near the direct solution before determining it is less preferable than the swap solution. Thus, perturbation may not significantly reduce the time MILP solvers spend looking at symmetrical solutions, but it is an approach worth investigating.

In the experiments described in Section 5, we define $\rho(i), v_i \in V$ as follows:

1. Calculate the maximum value contribution that the perturbations will

make to the objective function in terms of processor indices

$$\overline{\rho} = \sum_{v_i \in V} m\, i$$

2. Calculate a lower bound on the makespan if there was no communication between tasks, i.e., the tasks are independent

$$\underline{m}(G) = \frac{\sum_{v_i \in V} t(v_i)}{m}$$

This lower bound allows tasks to be split between processors, so it is unlikely to be realised unless the ETs of the task and the number of processors are particularly congenial, e.g., 4 tasks all with execution time of 4ms and 2 or 4 processors.

3. Calculate the increment for the perturbation

$$\varepsilon(\rho) = \frac{\underline{m}(G)}{10\overline{\rho}}$$

This ensures that the maximum contribution the perturbations will make to the objective function will be $\frac{1}{10}$ of the lower bound on the makespan (not enough to affect the optimal solution).

4. Define the perturbation for each node

$$\rho(i) = i\varepsilon(\rho), v_i \in V$$

By applying this perturbation "recipe" and replacing the makespan objective function (Equation (A01)) with the perturbed objective function in Equation (A01+), the antisymmetry MILP formulations ILP-BASIC-ASYM, ILP-RELAXED-ASYM, and ILP-REDUCED-ASYM were created from ILP-BASIC, ILP-RELAXED, and ILP-REDUCED respectively.

All 6 MILP formulations were investigated in the experiments described in the next section.

5. Experimental evaluation

The main goals of this section are: (a) performance comparisons of the proposed SMT formulation and the various MILP formulations as described in [29, 33, 32] and Section 4; (b) analysis of the behavior of the MILP formulations with respect to graph structures; (c) studying the effect of communication cost to computation cost ratio (CCR) on various formulation runtimes. The formulations are solved using Gurobi 5.6.3 [35] and Z3 4.4.2 [34], MILP and SMT solvers respectively, on an Intel Core i5 processor 3360M, 2.8 GHZ CPU and 8 GB RAM running with no parallel mode and on a single thread on Linux 4.5 x86-64. All experiments are run for the task scheduling model discussed in Section 2, i.e., a fully connected processor network with identical bandwidth capacity is assumed. The input graphs used for experiments in Section 5.1.1 are from [25] and the input graphs used for benchmarking in Section 6.2 are from [28, 38]. The following two definitions of graph densities are used:

$$\Omega = |E|/|V| \tag{8}$$

$$\omega = (|E|/\gamma) \times 100 \tag{9}$$

with the maximum possible number of edges in the graph as $\gamma = |V|(|V| - 1)/2$. The two density Equations (8) and (9) arise due to 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. We divide our experimental evaluation section into two distinct sections — Section 5.1 compares the proposed SMT formulation with *packing* MILP formulations and their variants as described in Section 4. Section 5.2 compares the SMT formulation with recent *non-packing* MILP formulations presented in [32, 33]. Two types of performance comparison experiments are carried out: (a) one minute timeout in Sections 5.1.1 and 5.2.1 and 12 hour timeout in Sections 5.1.2 and 5.2.2. The one minute timeout experiments carried out in order to get many results that inform the performance behaviour and the input characteristics on which it depends. Sections 5.1.1 and 5.2.1 compare the relation between characteristics of the task graph structure with respect to the SMT and MILP formulations. These sections also study the effect of CCR on SMT and packing MILP formulations. The 12 hour timeout experiments in Sections 5.1.2 and 5.2.2 are longer experiments for a direct comparison between runtime of different formulations. Large input graphs are tackled in these experiments. All input task graphs are represented in the *Graph eXchange Language* (GXL) [39] format.

Graph Structure	$n = 10$	$n = 21$	$n = 30$	Total
Fork-Join	30	30	30	90
Fork	30	30	30	90
Independent	10	10	10	30
InTree	30	30	30	90
Join	30	30	30	90
OutTree	30	30	30	90
Pipeline	30	30	30	90
Random	60	60	60	180
Series-Parallel	60	60	60	180
Stencil	30	30	30	90

Table 1: Detailed structure of the 1020 graph database

5.1. Comparison with packing MILP formulations

This section describes various comparisons between the SMT formulation and the packing MILP formulations described in Section 4 and [29].

5.1.1. Comparisons with one minute timeout

A database of 1020 task graphs, summarized in Table 1, comprising of 10, 21, and 30 tasks of the following structure: fork-join, fork, independent, intree, join, outtree, pipeline, random, series-parallel, and stencil are chosen from [25]. The densities of the graphs in the database conform to Equation (8). The experiments are carried out on 2, 4, 8, and 16 processor architecture. A one minute timeout is set for each graph.

Overall, we generated $4080 \times 6 = 24480$ results, one for each of the six packing MILP formulation variants, and 4080 SMT results. Given the large number of sample data points, we carried out statistical analysis to better understand the effect of graph characteristics on the MILP and SMT techniques. Our statistical analysis was a two label (1/0), optimal solution obtained, optimal solution *not* obtained, classification problem. We carried out two types of classifications: (1) logistic regression [40] and (2) decision tree classification. The results obtained from these two approaches are complimentary. Decision trees give us a big picture view, in that, they help us understand the *influence* of different graph characteristic on solvability of the problem. For example, the conclusion that the structure of the graph matters more than the number of tasks. Logistic regression helps us understand the *influence* of different sub-classes within a characteristic. For

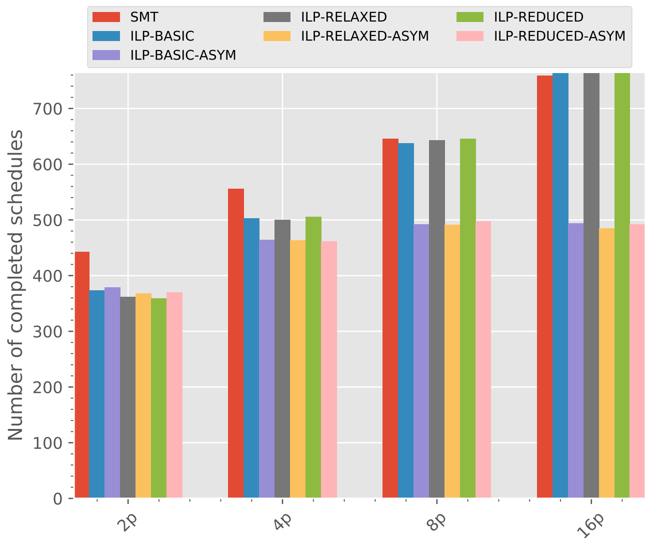


Figure 5: Completed schedules of different formulations over number of processors.

example, what type of structure is the hardest to solve. Note that we performed manual characteristic selection such that the statistical models have a prediction accuracy of 90%. We used the same characteristics (predictor variables) for logistic regression and decision trees. Furthermore, in the case of decision trees, we used the *gini* impurity to measure the quality of the split. At each node of the decision tree, the nodes were split according to the smallest gini measure and the trees were unrolled until the leaves were pure. The results of this analysis along with a traditional, graphical comparison of the approaches is given in the rest of the subsection.

Overall performance evaluation. The overall results are shown in Figure 5. SMT in general solves more graphs to optimality than the packing MILP formulations, when using two or four processors. As the number of processors increases to eight and 16 the two approaches perform comparably. SMT and MILP are able to solve larger numbers of graphs with increasing numbers of processors. SMT and MILP both use the simplex solver to solve the numerical equations and, with an increasing number of processors, there is more room for relaxation in simplex, which explains these results. In the case of MILP, symmetric solutions always outperform the anti-symmetric solutions. Finally, there is not much difference between the basic, relaxed, and

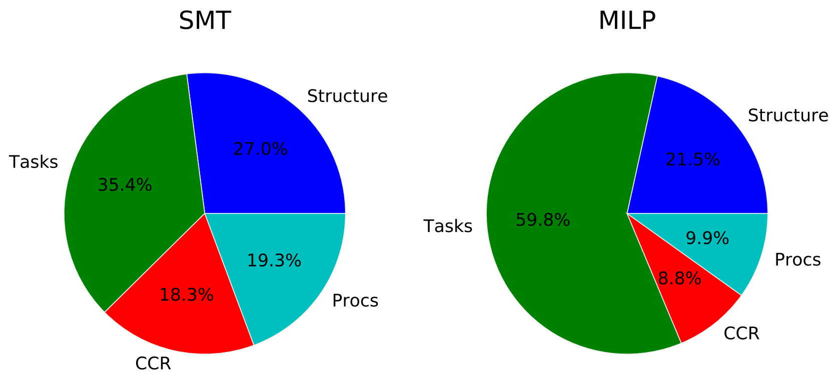


Figure 6: Influence of graph characteristics on solvability

reduced formulations.

The results for the statistical analysis via McNemar's test [41] which checks the (null) hypothesis that SMT and one of the packing MILP formulations are equally likely to obtain an optimal schedule on two, four, eight, or 16 processors is given in [42]. The results show that none of the anti-symmetric packing MILP formulations are equally likely to result in an optimal makespan compared to the SMT formulation. The symmetric packing MILP formulations on the other hand are as likely as the SMT formulation to produce an optimal schedule for the eight and 16 processors, but not equally likely to produce an optimal solution for the two and four processor systems compared to the SMT formulation. The influence scores for different graph characteristics on their solvability are shown in Figure 6. The number of tasks has the highest effect on solvability, followed by the type of the graph structure. In case of SMT; CCR value and number of processors plays a significant role, but not so for MILP. In fact, logistic regression results (Figure A.18 in Appendix A) inform us that SMT has a steeper (negative) gradient for increasing CCR values, compared to MILP, i.e., SMT is less likely to obtain an optimal schedule with larger CCR.

Structure based performance evaluation. Structure based performance evaluation comparing SMT and MILP techniques are shown in Figures 7-10. It is clear that fork/join, fork-join, and independent graph structures are the hardest to solve for both SMT and MILP. We attribute these results to the observation that our SMT and MILP formulations are ignorant about the fact that many tasks in independent, fork/join, and fork-join graphs do not

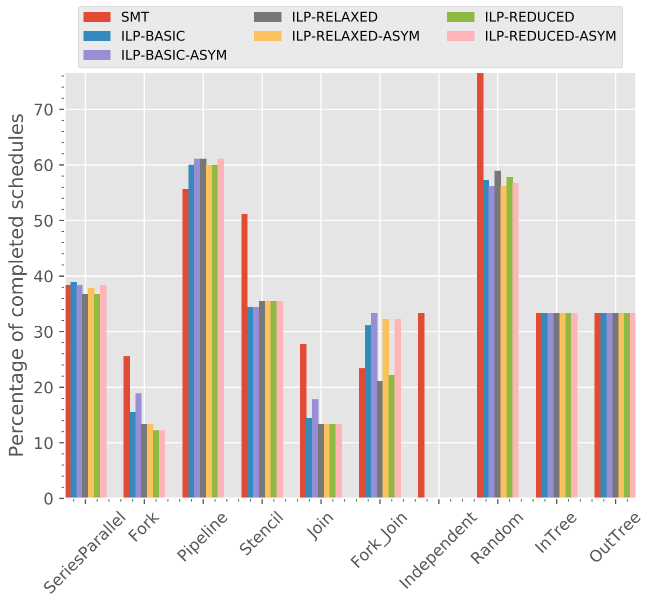


Figure 7: Completion percentage of formulations over graph structures on 2 processors

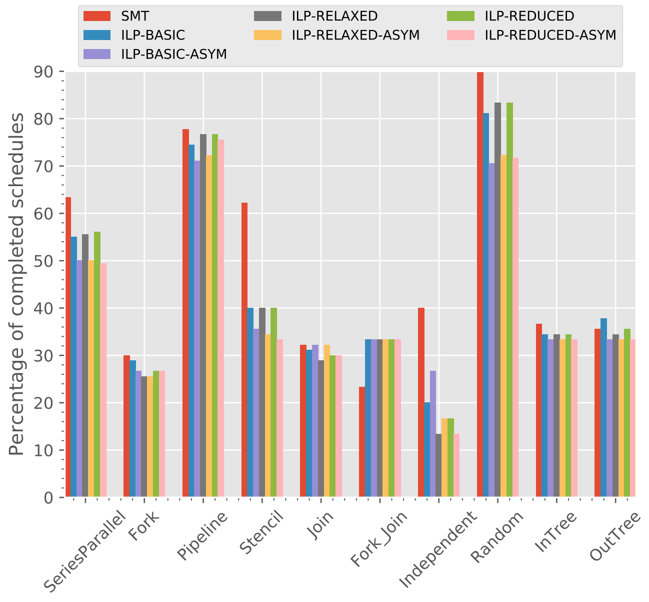


Figure 8: Completion percentage of formulations over graph structures on 4 processors

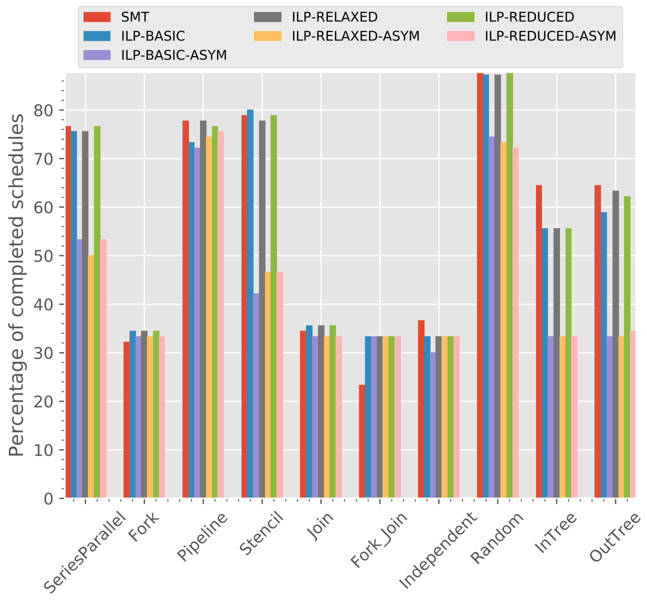


Figure 9: Completion percentage of formulations over graph structures on 8 processors

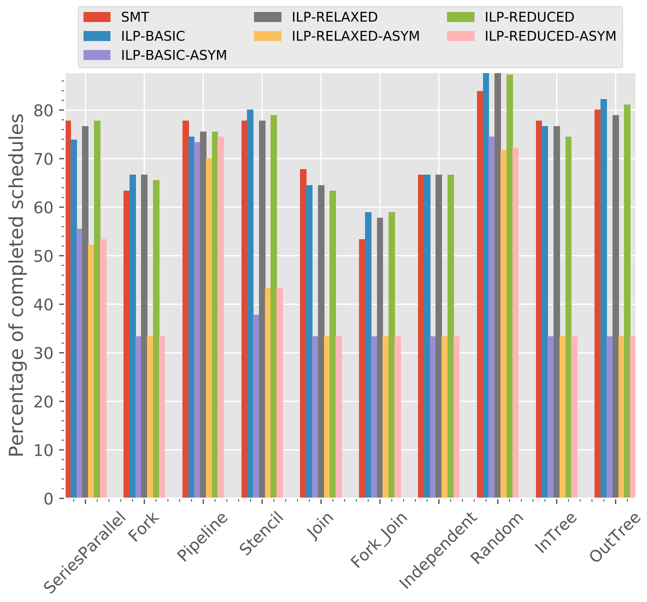


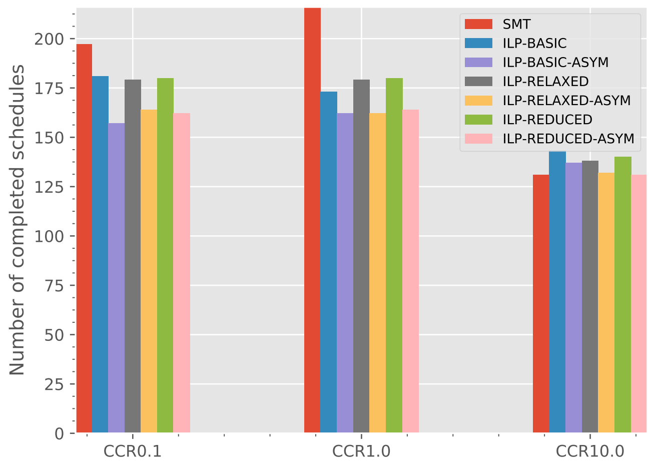
Figure 10: Completion percentage of formulations over graph structures on 16 processors

need to be ordered and in fact any order will suffice. Interestingly, these are the graphs where the ASYM approaches do the best for 2 processors When there are fewer processors, more tasks are allocated to the same processor and SMT/MILP need to sift through all possible orderings (a factorial operation) to make sure that the optimal solution is obtained. On the other hand, with increasing number of processors (especially when number of tasks is approx. equal to the number of processors), each task can be allocated to a separate processor and hence, search for the optimal ordering of tasks is reduced. Our statistical analysis simply reinforces the results shown in Figures 7-10.

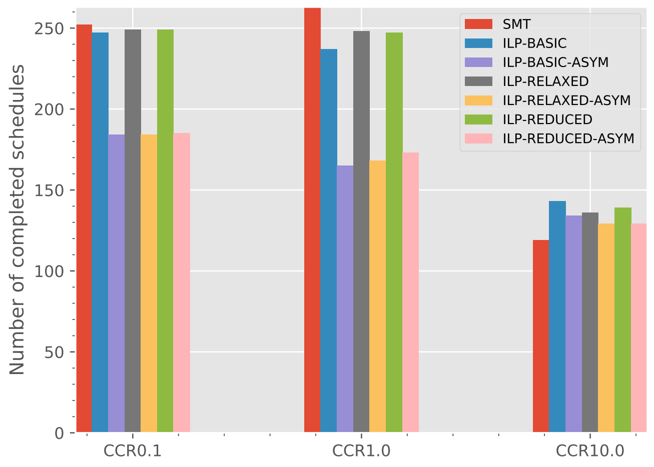
Effect of CCR on formulation runtime. In this section, the effect of CCR on the performance of SMT and MILP formulation is studied. In our database there are 330 graphs each of CCR 0.1, CCR 1.0, and CCR 10, respectively. The independent graphs in the database have no communication edges and are not considered for the analysis. The number of graphs for which an optimal schedule was returned by solvers within one minute is tallied and plotted in Figure 11 for 4 and 8 processors. We did not show the graphs of 2 and 16 processors, because they follow the same general trend.

The proposed SMT formulation performs better than the packing MILP formulations overall in terms of absolute number of optimal schedules obtained for CCR values of 0.1 and 1.0, but worse for a CCR value of 10.0. SMT and MILP formulations perform quite differently for varying CCR values. SMT formulation performs best on *medium* CCR value of 1.0, whereas it performs much worse for large CCR values. On the other hand all MILP formulations show a small deterioration in performance when CCR increase from 0.1 to 1.0 – not sure if this is true, but look that way – with a larger drop off as CCR increases from 1.0 to 10.0. However, at CCR of 10.0 the MILP approaches outperfrom SMT. This behaviour is because at low and medium CCR values, load balancing the task graphs is easier. Higher CCR values, on the other hand, 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.

This CCR dependent behavior is very interesting as it shows that the SMT and MILP runtimes do not only relate to the size of the input problem in terms of number of tasks, edges and processors, but also depends on the weight values. The main result from the above experiments is that SMT solvers (at least Z3 in our case) seem to be better than MILP solvers for load balancing problems. This result is reinforced by the fact that the



(a) Completed schedules of different formulations over CCR on 4 processors for 330 graph set for a 1 minute timeout



(b) Completed schedules of different formulations over CCR on 8 processors for the 330 graph set for a 1 minute timeout

Figure 11: CCR Comparisons on 4 and 8 Processors Set for a 1 minute Timeout

SMT formulation was able to obtain a far greater number of optimal results compared to any of the MILP formulation for the independent task graphs, which is a pure load balancing problem.

5.1.2. Comparisons with 12 hour timeout

The 12 hour timeout experiments are for a direct comparison between runtimes of the SMT and MILP formulations. The input graphs used for benchmarking in this section are from [28] and [38]. If the solver is unable to find an optimal solution within 12 hours, the program is terminated and the results tabulated. The m:s notation is the standard minutes:seconds taken by the 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 (Equations (8) and (9)). According to [28], the optimal solution for 'ogra' graphs are obtained when the tasks are well packed (as in an 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 that yields the fastest result (or the least gap) is highlighted.

SMT performs better than MILP formulations in six of the sixteen experiments, but we emphasise that each experiment consists of only one problem instance. Amongst the MILP formulations, it is seen that for most cases RELAXED or REDUCED formulations run faster than BASIC. The anti-symmetric formulations are slower in all cases. The most interesting observation is that SMT performs better when the number of processors is smaller, but REDUCED performs the best with larger number of processors. Despite 'ogra' graphs having a special structure making it harder to solve optimally, it is seen that the proposed SMT formulation has significantly improved performance and outdoes other formulations. For the random 't' graphs, we see the same the outcome; SMT performing better with smaller number of processors, but MILP formulations run slightly faster when more processors are available for scheduling. In our benchmarks there is no case where the packing MILP formulations provide an optimal solution but SMT fails to do so. Yet, there are cases, e.g., first row in Table 2, where SMT formulation gives an optimal solution, but all of the packing MILP formulations fail to

Table 2: 12 hour Timeout Comparisons on SMT vs. ILP Formulations

Graph	p	SMT	BASIC	BASIC-A	REL	REL-A	RED	RED-A
Ogra20_60, n=20 ω=60, Ω=5.7	2	**2s**	*12h 31.59%*	*12h 26.14%*	*12h 28%*	*12h 27.98%*	*12h 30%*	*12h 27.38%*
	4	**1s**	3s	18s	4s	9s	4s	7s
	8	4s	2s	6s	1.4s	3.5s	**1.1s**	4s
	16	15s	2s	7s	**1s**	8s	3s	3s
Ogra50_60, n=50 ω=60, Ω=14.7	2	***12h 0.04%***	*12h 48.67%*	*12h 48.13%*	*12h 45.79%*	*12h 48.13%*	*12h 48%*	*12h 46.3%*
	4	**21.7s**	11m:5s	46m:5s	17m:0s	189m:5s	28m:3s	106m:7s
	8	1m:0s	1m:9s	19m:6s	52s	7m:3s	**26s**	8m:9s
	16	3m:5s	50s	1m:7s	42s	3m:0s	**29s**	3m:4s
t30_60_1, n=30 ω=60, Ω=8.7	2	**2.48s**	7.7s	3.7s	3.9s	3.2s	4.7s	2.49s
	4	4.6s	5.7s	6s	3.5s	**2.8s**	3.6s	3.2s
	8	13.2s	10.7s	5.6s	4.6s	**3.5s**	4.7s	3.6s
	16	50.7s	5.9s	4.1s	4.2s	3.3s	3.7s	**3.2s**
t40_30_2, n=40 ω=30, Ω=5.85	2	**11.7s**	1m:2s	1m:3s	18.9s	1m	16.8s	54.4s
	4	**10.9s**	40.9s	44.2s	14.9s	13s	14s	13.3s
	8	18.5s	32.2s	22.8s	13.8s	13s	**11.9s**	15.4s
	16	1m:4s	1m:09s	56.5s	16.5s	20s	**13.5s**	15.3s

Legend REL: RELAXED, RED: REDUCED

do so. Overall, we posit that the proposed (and very first) SMT formulation is comparable to (and sometimes better than) the *best* packing MILP formulations. We also expect that improvements applied to packing MILP formulations [43] could also be applied to the SMT formulation, which is likely to result in further improvements in the execution time of the SMT formulation. We leave applying such improvements to the SMT formulation as future research.

5.2. Comparison with non-packing MILP formulations

Two *non-packing* MILP formulations for scheduling task graphs have recently been proposed in [32, 33]. These non-packing formulations differ from the MILP formulations in Section 4, in that these formulations use processor assignment and positional date (start time for task) decision variables only.

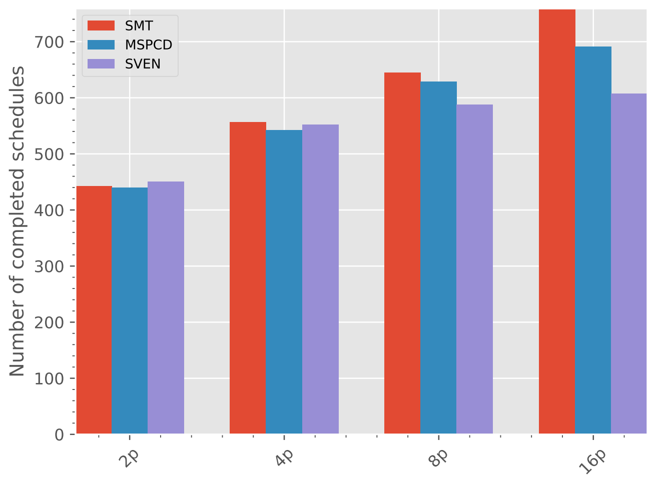


Figure 12: Completed schedules of different formulations over number of processors.

The formulation in [33] is especially similar to the proposed SMT formulation, because their formulation also utilizes the transitive closure of the task graph to reduce the number of constraints and variables when scheduling tasks on the same processor.

In this section we compare the SMT and the *non-packing* MILP formulations. In the rest of the subsections we call the optimised formulation in [33] as the MSPCD formulation, and the *non-packing* formulation in [32] as the SVEN formulation. Like before, we generated 4080 $\times$ 2 MILP results and 4080 SMT results for 1-minute timeout experiments. Finally, we use the same benchmarks as shown in Table 2 for the 12-hour timeout comparisons.

5.2.1. Comparison with one minute timeout

Overall, the proposed SMT formulation outperforms the MSPCD and SVEN formulations as shown in Figure 12. In case of two and four processors, SMT and non-packing MILP formulations schedule almost equal number of task graphs to optimality. However, as the number of processors increases the proposed SMT formulation outperforms both the MSPCD and SVEN formulations.

Structure based performance evaluation. A more detailed comparison for the results of 1-minute timeouts is presented in Figures 13 to 16. These figures give the percentage of completed schedules for each of the formula-

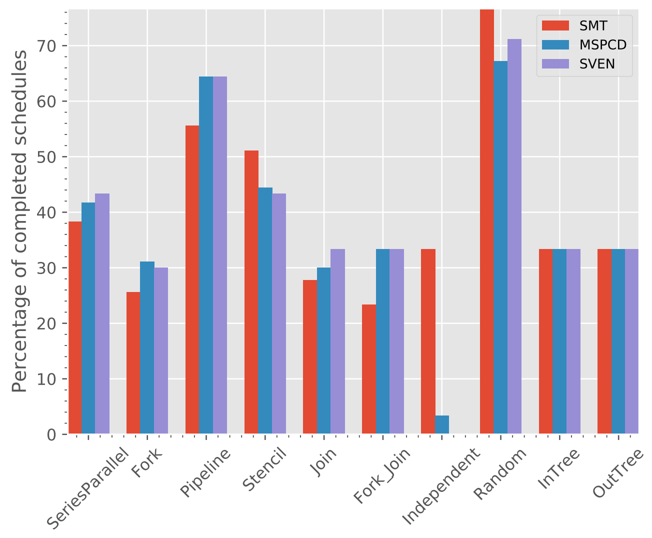


Figure 13: Completion percentage of formulations over graph structures on 2 processors

tions, with varying task graph structures. The independent task-graphs are the hardest to solve for these non-packing MILP formulations, MSPCD and SVEN, as they are unable to solve independent task graphs on the two processor system. As the number of processors increases, MSPCD and SVEN formulations are able to solve equal number of independent graphs to optimality as SMT, but start lagging in solving other graph structures to optimality. This is opposite to the packing MILP formulations, which perform comparably to the SMT formulation for large number of processors (c.f. Figures 7-10), but perform poorly compared to SMT for small number of processors. S. Mallach in his paper [32] also states that the non-packing formulations perform poorly when the number of processors increases — our experiments and the McNemar's hypothesis test results [42], re-enforce his observation.

Effect of CCR on formulation runtime. Like before, we compare the number of completed schedules for 330 task graph set with 0.1, 1.0, and 10.0 CCR, respectively in Figure 17. The MSPCD and SVEN formulations outperform the SMT formulation for a higher CCR value, but perform poorly for lower CCR values.

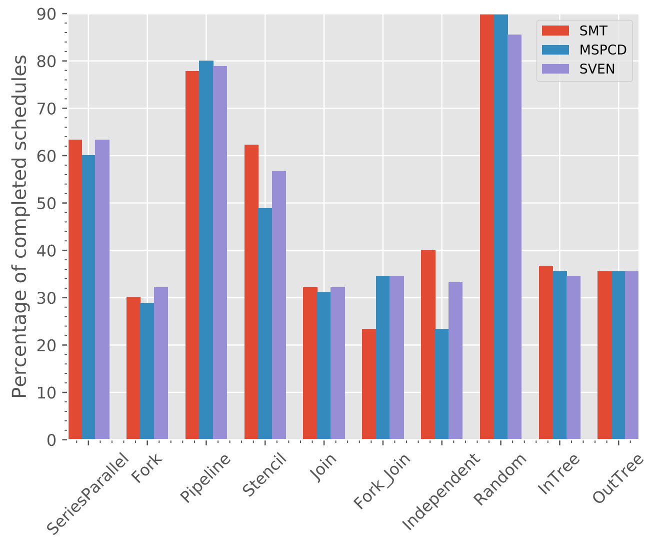


Figure 14: Completion percentage of formulations over graph structures on 4 processors

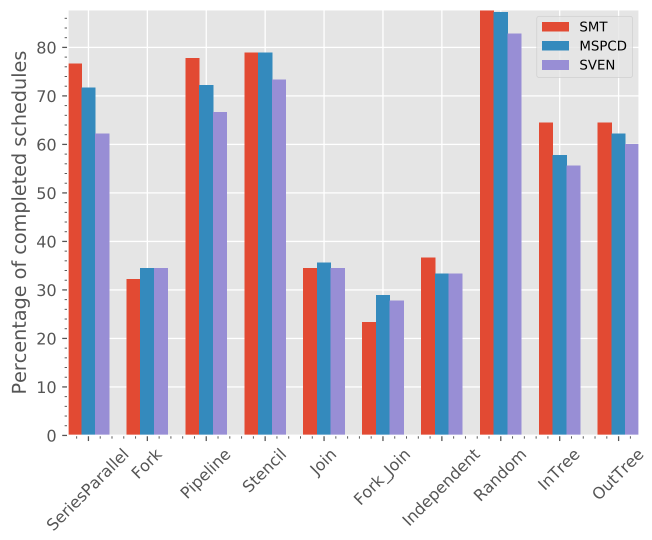


Figure 15: Completion percentage of formulations over graph structures on 8 processors

5.2.2. Comparison with 12 hour timeout

The results of the 12-hour timeout experiments comparing large task graphs being solved by the SMT and the two non-packing MILP formula-

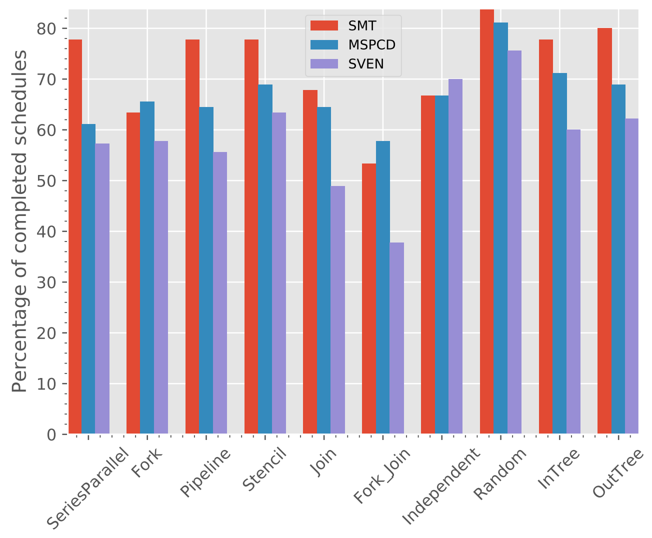


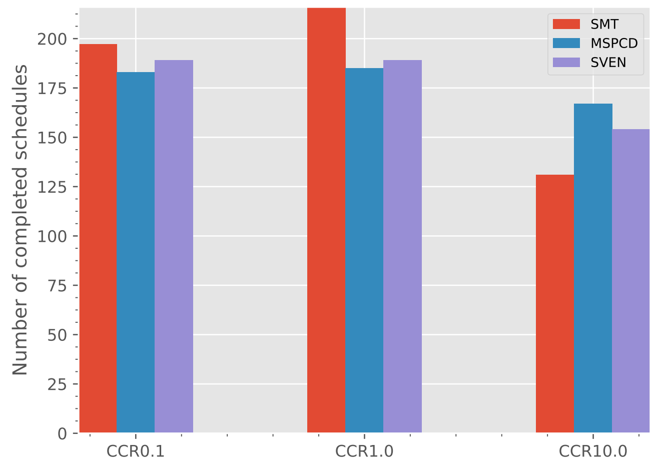
Figure 16: Completion percentage of formulations over graph structures on 16 processors

tions are shown in Table 3. In 81.25% of the cases, SMT performs better than either of the non-packing MILP formulations. MSPCD and SVEN formulations give slightly better results (a few seconds faster) in three cases. As before, there is no benchmark where the SMT formulation is unable to find an optimal solution, but the non-packing MILP formulations do. Yet, there are instances, where the non-packing MILP formulations fail to find an optimal schedule, but SMT is able to provide one.

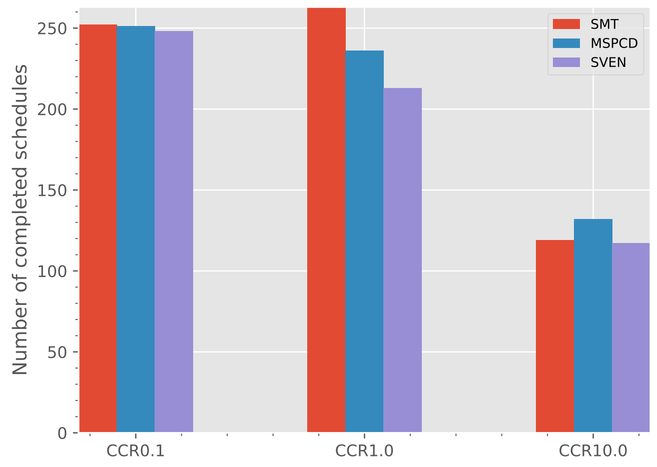
All our results, including the large task graph sets, point to the fact that, in general, SMT is a good solution for scheduling task graphs comparable to (and in many cases better than) both the packing and the non-packing MILP formulations.

6. Related Work

A very large number of algorithms have been proposed for scheduling task graphs in the past decades. As already stated in the introduction, most of these algorithms are heuristics, which do not give a guarantee on the schedule length in relation to the optimal schedule length. Our proposed SMT based approach solves the scheduling problem for small and medium sized instances optimally. Hence we here focus the discussion of related work on optimal scheduling algorithms.



(a) Completed schedules of different formulations over CCR on 4 processors for 330 graph set for a 1 minute timeout



(b) Completed schedules of different formulations over CCR on 8 processors for the 330 graph set for a 1 minute timeout

Figure 17: CCR Comparisons on 4 and 8 Processors Set for a 1 minute Timeout

Table 3: 12 hour Timeout Comparisons on SMT vs. non-packing MILP Formulations

Graph	p	SMT	MSPCD	SVEN
Ogra20_60, n=20 ω=60, Ω=5.7	2	**2s**	*12h 22.5%*	*12h 26%*
	4	**1s**	3.68s	1.7s
	8	**4s**	8s	5.2s
	16	15s	26.64s	**12.2s**
Ogra50_60, n=50 ω=60, Ω=14.7	2	***12h 0.04%***	*12h* 47.33%	*12h* 44.3%
	4	**21.7s**	1m:2s	1m:8s
	8	**1m:0s**	1m:3s	1m:1s
	16	**3m:5s**	4m:7s	11m:8s
t30_60_1, n=30 ω=60, Ω=8.7	2	**2.48s**	2.5s	2.5s
	4	4.6s	**4.33s**	4.36s
	8	**13.2s**	15.26s	16s
	16	**50.7s**	1m:26s	5m:7s
t40_30_2, n=40 ω=30, Ω=5.85	2	11.7s	**4.29s**	5.8s
	4	**10.9s**	14.6s	1m:5s
	8	**18.5s**	73m:2s	57m:7s
	16	**1m:4s**	23m:0s	*12h 4.7%*

One category of optimal scheduling algorithms uses an exhaustive search through the solution space. Given that the addressed scheduling problem is a combinatorial optimization problem, there is an extremely large, but finite number of possible solutions. Exhaustive search algorithms try to smartly enumerate all possible solutions and thereby identify an optimal solution. Two different models have been proposed to span the solution space. The exhaustive list scheduling model essentially uses a list scheduling algorithm, where, in each step, a task is scheduled on a processor [22, 23, 24, 25, 26]. The exhaustive version tries all task orders and all processors exhaustively. The other model, the allocation-ordering model, first allocates tasks to processors and then orders them on the processor. This latter approach has the advantage that the creation of duplicate states can be avoided efficiently [44]. Based on the state space model, different approaches can be used to search this space as known from the area of artificial intelligence, e.g. A* [22, 23, 25, 26], IDA*, Depth-First Branch-and-Bound [24]. In general, successful exhaustive search approaches are very problem specific using as much domain knowledge as possible. An important aspect for the efficient execution of the search are good pruning techniques that discard large parts of the search space without jeopardizing the algorithms ability to find the optimal solution [26].

A more general approach for combinatorial optimization problems and scheduling problems in particular is the use of Mixed Integer Linear Programming (MILP). The MILP formulations can be broadly classified as discrete time and continuous time approaches [45, 46]. The discrete time approach introduces a new variable for each instant of time on each processor [47]. 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 [48, 49]. 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 [50, 51]. 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 [45, 28, 52]. This approach has been employed in the MILP formulations that are used in this paper, Section 4.

Work in [31] proposes a *Satisfiability Modulo Graph Theory* (SMGT) formulation for scheduling task graphs on multi-processor systems. The work in [31] builds a two stage solver based on Davis-Putnam-Logemann-Loveland (DPLL). The first stage uses a pure SAT formulation to obtain a solution in terms of boolean constraints. The second stage, a so called *Makespan Analysis Engine* (MAE) detects deadlock cycles introduced by the SAT solver and the *reason* for a given makespan (the schedule length), if no deadlock is detected. This reason is complemented and later used to reduce the search space akin to *Conflict Driven Clause Learning* (CDCL) solvers. There are two primary differences between our work and the work presented in [31]: (a) we use an integrated SMT solver with a single boolean decision variable, whereas their SAT formulation uses three boolean decision variables per task in the task graph. In the worst case, the search space grows exponentially with the number of Boolean decision variables and hence, in the worst case, our formulation should have a shorter execution time. (b) Their SAT formulation might introduce deadlock cycles when ordering independent nodes on a single processor, these inadvertent deadlocks need to be explicitly handled in their formulation, our formulation on the other hand is free of such deadlocks.

7. Conclusions

This paper proposed a novel optimal scheduling approach to the NP-hard problem of scheduling a task graph with communication delays on a set of homogeneous processors. The approach is based on an elegant and compact SMT formulation of the scheduling problem. Most of the constraints of this formulation are directly derived from the scheduling model and the definition of the problem. Only one decision variable needed to be introduced and linear theory models were applied. To evaluate the performance of this new formulation we performed an extensive experimental evaluation, using a large set of task graphs and scheduled them on different number of processors. The performance of the SMT formulation on these graphs was compared with that of the best known MILP formulations for the scheduling problem. We experimented with straightforward anti-symmetry measures to improve the performance of these existing MILP formulations, but the results did not demonstrate any improvement. SMT outperformed the MILP formulations in many cases. SMT successfully solves problems with a lower number of processors and low to medium CCR, whereas MILP formulations are more successful when there are a high number of processors and a large CCR (approximately 10). From the statistical analysis we also see that the solution times of the two approaches are differently influenced, e.g., SMT formulations are not as strongly influenced by the number of tasks, but more by the number of processors in comparison to MILP. The good performance of SMT is extremely encouraging, given that MILP solution methods have undergone many advances in recent years that have significantly improved their performance. Hence, apart from being competitive already, our SMT approach has further potential to be improved in future work. A starting point for this can be the transfer of the pruning techniques discovered for exhaustive state-space search or the optimization that MILP formulations have used.

References

[1] O. Sinnen, Task Scheduling for Parallel Systems, Wiley, 2007.

[2] M. Drozdowski, Scheduling for Parallel Processing, Springer, 2009.

[3] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: a unified platform for task scheduling on heterogeneous multicore architectures, Concurrency and Computation: Practice and Experience 23 (2) (2011) 187–198.

[4] T. Gautier, X. Besseron, L. Pigeon, KAAPI: A thread scheduling runtime system for data flow computations on cluster of multi-processors, in: International Workshop on Parallel Symbolic Computation, 2007, pp. 15–23. doi:10.1145/1278177.1278182.

[5] J. Planas, R. M. Badia, E. Ayguadé, J. Labarta, Hierarchical task-based programming with StarSs, IJHPCA 23 (3) (2009) 284–299. doi:10.1177/1094342009106195.
URL http://dx.doi.org/10.1177/1094342009106195

[6] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, J. J. Dongarra, PaRSEC: Exploiting heterogeneity for enhancing scalability, Computing in Science & Engineering 15 (6) (2013) 36–45.

[7] O. A. R. Board, OpenMP application program interface, version 4.0, http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf (Jul. 2013).

[8] V. Sarkar, Partitioning and scheduling parallel programs for execution on multiprocessors, Tech. rep., Stanford Univ., CA (USA) (1987).

[9] P. Chrétienne, Task scheduling over distributed memory machines, in: Proc. of the Int. Workshop on Parallel and Distributed Algorithms, North-Holland, Amsterdam, 1989.

[10] H. El-Rewini, H. H. Ali, On considering communication in scheduling task graphs on parallel processors, Journal of Parallel Algorithms and Applications 3 (1994) 177–191.

[11] T. C. Hu, Parallel sequencing and assembly line problems, Operations research 9 (6) (1961) 841–848.

[12] G. Liu, K.-L. Poh, M. Xie, Iterative list scheduling for heterogeneous computing, Journal of Parallel and Distributed Computing 65 (5) (2005) 654–665.

[13] B. S. Macey, A. Y. Zomaya, A performance evaluation of CP list scheduling heuristics for communication intensive task graphs, in: Parallel Processing Symposium, 1998. Proc. of IPPS/SPDP 1998, 1998, pp. 538 –541.

[14] T. Yang, A. Gerasoulis, List scheduling with and without communication delays, Parallel Computing 19 (12) (1993) 1321–1344.

[15] B. Cirou, E. Jeannot, Triplet: a clustering scheduling algorithm for heterogeneous systems, in: Proc. of Workshop on Scheduling and Resource Management for Cluster Computing (ICPP 2001), IEEE Press, Valencia, Spain, 2001, pp. 231–236.

[16] D. Kadamuddi, J. J. Tsai, Clustering algorithm for parallelizing software systems in multiprocessors environment, IEEE Transactions on Software Engineering 26 (4) (2000) 340–361.

[17] V. Kianzad, S. S. Bhattacharyya, Efficient techniques for clustering and scheduling onto embedded multiprocessors, IEEE Transactions on Parallel and Distributed Systems 17 (7) (2006) 667–680.

[18] A. Gerasoulis, T. Yang, On the granularity and clustering of directed acyclic task graphs, IEEE Transactions on Parallel and Distributed Systems 4 (6) (1993) 686–701.

[19] M. I. Daoud, N. Kharma, A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks, Journal of Parallel and Distributed Computing 71 (11) (2011) 1518–1531.

[20] F. A. Omara, M. M. Arafa, Genetic algorithms for task scheduling problem, Journal of Parallel and Distributed Computing (70) (2010) 13–22.

[21] J.-J. Hwang, Y.-C. Chow, F. D. Anger, C.-Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, SIAM Journal on Computing 18 (2) (1989) 244–257.

[22] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (1998) 42–51.

[23] Y.-K. Kwok, I. Ahmad, On multiprocessor task scheduling using efficient state space approaches, Journal of Parallel and Distributed Computing 65 (2005) 1515–1532.

[24] S. Venugopalan, O. Sinnen, Memory limited algorithms for optimal task scheduling on parallel systems, Journal of Parallel and Distributed Computing 92 (2016) 35–49. `doi:10.1016/j.jpdc.2016.03.003`.

[25] A. Z. S. Shahul, O. Sinnen, Scheduling task graphs optimally with A^*, The Journal of Supercomputing 51 (3) (2010) 310–332. `doi:10.1007/s11227-010-0395-1`.
URL `http://dx.doi.org/10.1007/s11227-010-0395-1`

[26] O. Sinnen, Reducing the solution space of optimal task scheduling, Computers & Operations Research 43 (2014) 201–214. `doi:10.1016/j.cor.2013.09.004`.

[27] A. Davare, J. Chong, Q. Zhu, D. Densmore, A.Sangiovanni-Vincentelli, Classification, customization, and characterization: Using milp for task allocation and scheduling, Tech. Rep. UCB/EECS-2006-166, University of California, Berkeley (December 2006).

[28] T. Davidović, L. Liberti, N. Maculan, N. Mladenović, Towards the optimal solution of the multiprocessor scheduling problem with communication delays, in: In Proc. 3rd Multidisciplinary Int. Conf. on Scheduling: Theory and Application (MISTA), 2007, pp. 128–135.

[29] S. Venugopalan, O. Sinnen, ILP formulations for optimal task scheduling with communication delays on parallel systems, IEEE Trans. Parallel Distrib. Syst. 26 (1) (2015) 142–151. doi:10.1109/TPDS.2014.2308175.
URL http://dx.doi.org/10.1109/TPDS.2014.2308175

[30] K. Kuchcinski, Constraints-driven scheduling and resource assignment, ACM Transactions on Design Automation of Electronic Systems 8 (3) (2003) 355–383.

[31] W. Liu, Z. Gu, J. Xu, X. Wu, Y. Ye, Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems, IEEE Transactions on Parallel and Distributed Systems 22 (8) (2011) 1382–1389. doi:10.1109/TPDS.2010.204.

[32] S. Mallach, Improved mixed-integer programming models for multiprocessor scheduling with communication delays, Tech. rep., Institut fur Informatik Universitat zu Koln, 50923 Koln, Germany (2016).

[33] A. A. El Cadi, R. B. Atitallah, S. Hanafi, N. Mladenović, A. Artiba, New mip model for multiprocessor scheduling problem with communication delays, Optimization Letters (2014) 1–17.

[34] L. De Moura, N. Bjørner, Z3: An efficient smt solver, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.

[35] I. Gurobi Optimization, Gurobi optimizer reference manual (2015).
URL http://www.gurobi.com

[36] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories., Handbook of satisfiability 185 (2009) 825–885.

[37] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Communications of the ACM 5 (7) (1962) 394–397.

[38] T. Davidović, T. G. Crainic, Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems, Computers & operations research 33 (8) (2006) 2155–2177.

[39] A. Winter, B. Kullbach, V. Riediger, An overview of the gxl graph exchange language, Software Visualization (2002) 528–532.

[40] J. Friedman, T. Hastie, R. Tibshirani, The elements of statistical learning, Vol. 1, Springer series in statistics Springer, Berlin, 2001.

[41] Q. McNemar, Note on the sampling error of the difference between correlated proportions or percentages, Psychometrika 12 (2) (1947) 153–157.

[42] Mcnemar's hypothesis results, `https://anon1980.bitbucket.io/McNemar_results.html`, accessed: 2017-06-15.

[43] O. Sinnen, Reducing the solution space of optimal task scheduling, Computers & Operations Research 43 (2014) 201–214.

[44] M. Orr, O. Sinnen, A duplicate-free state-space model for optimal task scheduling, in: Proc. of 21st Int. European Conference on Parallel and Distributed Computing (Euro-Par 2015), Vol. 9233 of Lecture Notes in Computer Science, Springer, Vienna, Austria, 2015.

[45] A. Davare, J. Chong, Q. Zhu, D. M. Densmore, 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).
URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-166.html`

[46] C. A. Floudas, X. Lin, Mixed Integer Linear Programming in Process Scheduling: Modeling, Algorithms, and Applications, Annals of Operations Research 139 (1) (2005) 131–162.

[47] Chapter 3 Parallel machines - Linear programming and enumerative algorithms, Annals of Operations Research 7 (1986) 99–132. `doi:10.1007/BF02186435`.
URL `http://dx.doi.org/10.1007/BF02186435`

[48] P. E. Coll, C. C. Ribeiro, C. C. de Souza, Multiprocessor scheduling under precedence constraints: Polyhedral results, Discrete Applied Mathematics 154 (5) (2006) 770–801, iV ALIO/EURO Workshop on Applied Combinatorial Optimization. `doi:10.1016/j.dam.2004.07.009`.

URL http://www.sciencedirect.com/science/article/pii/S0166218X05003069

[49] A. Bender, MILP Based Task Mapping for Heterogeneous Multiprocessor Systems, in: in Proceedings European Design Automation Conference, IEEE, 1996, pp. 190–197.

[50] T. Davidović, L. Liberti, N. Maculan, N. Mladenović, Mathematical programming-based approach to scheduling of communicating tasks, Tech. rep. (2004).

[51] N. Maculan, S. C. S. Porto, C. C. Ribeiro, C. C. de Souza, R. Cid, C. Souza, A New Formulation for Scheduling Unrelated Processors under Precedence Constraints, RAIRO Operations Research 33 (1997) 87–91.

[52] S. Venugopalan, O. Sinnen, Optimal linear programming solutions for multiprocessor scheduling with communication delays, in: Proc. of 12th Int. Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2012), Vol. 7439 of Lecture Notes in Computer Science, Springer, Fukuoka, Japan, 2012, pp. 129–138. doi:10.1007/978-3-642-33078-0_10.

Appendix A.

coef	std	err	z	P>	z	[95.0% Conf. Int.]
Intercept	5.0223	0.288	17.452	0.000	4.458	5.586
C(Structure)[T.Fork$_{\text{Join}}$]	-0.8442	0.261	-3.235	0.001	-1.356	-0.333
C(Structure)[T.InTree]	1.6254	0.256	6.358	0.000	1.124	2.126
C(Structure)[T.Independent]	-1.2082	0.358	-3.374	0.001	-1.910	-0.506
C(Structure)[T.Join]	0.7270	0.265	2.748	0.006	0.209	1.245
C(Structure)[T.OutTree]	1.5439	0.255	6.059	0.000	1.045	2.043
C(Structure)[T.Pipeline]	3.6256	0.278	13.019	0.000	3.080	4.171
C(Structure)[T.Random]	5.2834	0.284	18.634	0.000	4.728	5.839
C(Structure)[T.SeriesParallel]	2.7540	0.232	11.886	0.000	2.300	3.208
C(Structure)[T.Stencil]	3.6438	0.285	12.783	0.000	3.085	4.202
C(Procs)[T.4]	1.3940	0.161	8.676	0.000	1.079	1.709
C(Procs)[T.8]	2.2864	0.170	13.423	0.000	1.953	2.620
C(Procs)[T.16]	3.5823	0.192	18.633	0.000	3.205	3.959
Tasks	-0.3258	0.012	-27.184	0.000	-0.349	-0.302
CCR	-0.4021	0.017	-23.085	0.000	-0.436	-0.368

(a) SMT Logistic Regression results

coef	std	err	z	P>	z	[95.0% Conf. Int.]
Intercept	4.2613	0.106	40.059	0.000	4.053	4.470
C(Structure)[T.Fork$_{\text{Join}}$]	0.5052	0.107	4.700	0.000	0.295	0.716
C(Structure)[T.InTree]	1.1770	0.107	10.981	0.000	0.967	1.387
C(Structure)[T.Independent]	-1.5701	0.155	-10.102	0.000	-1.875	-1.265
C(Structure)[T.Join]	0.2743	0.114	2.407	0.016	0.051	0.498
C(Structure)[T.OutTree]	1.2938	0.108	12.015	0.000	1.083	1.505
C(Structure)[T.Pipeline]	4.2330	0.117	36.239	0.000	4.004	4.462
C(Structure)[T.Random]	4.5931	0.108	42.693	0.000	4.382	4.804
C(Structure)[T.SeriesParallel]	2.6045	0.097	26.837	0.000	2.414	2.795
C(Structure)[T.Stencil]	2.2756	0.116	19.570	0.000	2.048	2.504
C(Procs)[T.4]	1.3421	0.066	20.300	0.000	1.213	1.472
C(Procs)[T.8]	2.2248	0.070	32.008	0.000	2.089	2.361
C(Procs)[T.16]	2.8399	0.073	39.024	0.000	2.697	2.983
Tasks	-0.3540	0.005	-73.085	0.000	-0.364	-0.345
CCR	-0.1770	0.006	-32.023	0.000	-0.188	-0.166

(b) MILP Logistic Regression results

Figure A.18: SMT and ILP Logit regression results