Scheduling DAGs When Processor Assignments Are Specified

Sanjoy Baruah[*]
Washington University in Saint Louis
baruah@wustl.edu

ABSTRACT

The problem of scheduling a workload represented as a directed acyclic graph (DAG) upon a dedicated multiprocessor platform is considered, in which each individual vertex of the DAG is assigned to a specific processor and the entire DAG is required to complete execution within a specified duration. A representation of this scheduling problem as a zero-one integer linear program is obtained.

KEYWORDS

Precedence-constrained jobs; Multiprocessor Scheduling; Restricted Processor Assignment; Exact Schedulability Test; Integer Linear Program

ACM Reference Format:
Sanjoy Baruah. 2020. Scheduling DAGs When Processor Assignments Are Specified. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394810.3394813

1 INTRODUCTION

At the previous edition of this conference in November 2019, Marko Bertogna presented a keynote titled "*A View on Future Challenges for the Real-Time Community*", in which he made a strong and convincing case that future complex real-time application systems are likely to be implemented upon heterogeneous multiprocessor platforms, and that the workloads are likely to be characterized by complex dependencies amongst pieces that are required to execute upon different kinds of processors. He argued that real-time scheduling theory needs to devote more attention to better understanding the behavior of such complex multiprocessor implementations in order that future safety-critical cyber-physical systems can be assured to have predictable timing properties. For a start, he identified the following problem as one of the simplest abstract scheduling problems that needs to be solved in order to be able to do scheduling-theoretic analysis of such complex implementations:

> Given a directed acyclic graph (DAG) in which the vertices (each of which is labeled with a worst-case execution time – WCET) represent individual pieces of computation *that are each assigned to specific processors* and the edges represent precedence constraints between these pieces of computation, determine whether the DAG can be scheduled in a manner that guarantees to meet a specified overall end-to-end deadline.

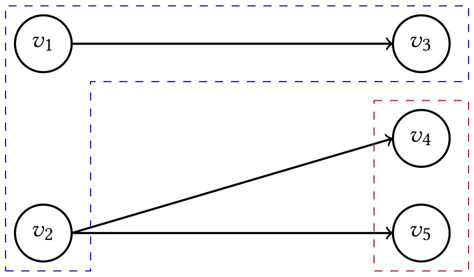
This problem was convincingly motivated in Bertogna's keynote address: modern platforms come equipped with highly specialized accelerators such as GPUs and TPUs (Tensor Processing Units), upon which specific computations need to be executed. However, even this very simple problem is known [6] to be computationally intractable – NP-hard in the strong sense. As a consequence of this complexity result, it is unlikely that efficient algorithms exist that are able to solve this problem optimally; Bertogna described efforts at solving it and its generalizations heuristically (see, e.g., [4]), but lamented the lack of exact, even if inefficient, algorithms for the problem against which the performance of the heuristics could be compared experimentally for small problem instances in order to characterize the effectiveness of the different heuristics. The research reported in this paper represents an effort at obtaining such an exact algorithm.

Contribution. We derive an algorithm for representing the scheduling problem discussed above as an integer linear program (ILP), which can then be solved using standard ILP-solvers. In fact, it turns out that all we need is a *zero-one* ILP — an ILP in which all integer variables are restricted to take on the values zero or one only; this is particularly welcome, since ILP-solvers are in general able to solve zero-one ILPs far more efficiently than "regular" ILPs.

Representing our DAG scheduling problem as a zero-one ILP turns out to be surprisingly non-trivial. While several standard techniques have been developed within the traditional Operations Research (OR) community for ILP-based representations of scheduling problems (See, e.g, [2, Appendix C] for a text-book introduction to some of these methods), these techniques, such as *time-indexing* (in which integer variables are used to represent which job is executing at each time-unit upon each processor) or *ordering* (in which integer variables are used to represent the order in which the jobs are to execute upon each processor), do not seem to be particularly suitable for our problem. In particular, time-indexing does not scale well since the number of integer variables needed is linear in the duration of the schedule (i.e., the value of the end-to-end deadline), while ordering is typically used for representing non-preemptive scheduling problems and it is not obvious how this technique should be extended to allow for preemption. The approach we have developed, and which is detailed in Section 3, requires us to combine some of these traditional OR techniques with ideas (such as the *demand-bound function* [3]) that are explicitly from the domain of real-time scheduling theory: we consider this inter-disciplinary

[*]Funded in part by National Science Foundation Grants CNS-1814739 and CPS-1932530

RTNS 2020, June 9–10, 2020, Paris, France
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394813



$D = 7$

v_i	$\pi(v_i)$	$c(v_i)$
v_1	P_1	2
v_2	P_1	2
v_3	P_1	3
v_4	P_2	2
v_5	P_2	2

Figure 1: An example instance (that is discussed in Example 1)

integration of concepts from two domains to be one of the major intellectual contributions of this work.

Organization. The remainder of this paper is organized in the following manner. In Section 2 we formally describe the scheduling problem that we seek to solve, and provide an illustrative example that is used as a running example in the remainder of the paper. In Section 3 we detail how this scheduling problem may be formulated as a Zero-One Integer Linear Program. In Section 4 we briefly discuss some related work, and then conclude in Section 5 with a short discussion providing some context, and by identifying out some possible directions for followup work.

2 PROBLEM STATEMENT

The problem that we are addressing in this paper may be described in the following manner. We are given a (single) directed acyclic graph $G = (V, E)$, in which the vertices represent sequential pieces of computation ("jobs") that need to be executed, and the directed edges represent precedence constraints between jobs: for each edge the job from which the edge is emanating must complete execution before the job to which the edge leads may begin to execute. Each job is assigned to a specific processor; i.e., each vertex $v_i \in V$ of the DAG is characterized by a parameter $\pi(v_i)$, denoting the processor upon which the job is required to execute. (The processors are assumed to be preemptive — it is not required that the jobs execute non-preemptively upon the processors to which they are assigned.) Each $v_i \in V$ is also characterized by a parameter $c(v_i)$, which denotes its WCET upon the processor $\pi(v_i)$. A deadline D is specified for the DAG. We seek to synthesize a schedule for the jobs in the DAG such that (i) each job v_i gets to execute for a duration $c(v_i)$ upon the processor $\pi(v_i)$; (ii) the precedence constraints represented by the edges in the DAG are respected; and (iii) all the jobs in the DAG complete execution within an interval of duration not exceeding D time units from the instant at which execution (of any job) commences.

Example 1. Throughout this paper we will consider an example instance in which the DAG depicted in Figure 1, comprising the five jobs v_1, v_2, v_3, v_4 and v_5 and three edges (v_1, v_3), (v_2, v_4), and (v_2, v_5), is to be executed upon a 2-processor platform with an end-to-end deadline D of 7 time units. The processor assignment and the WCETs of the individual jobs are also depicted in the figure; here the P_1 and P_2 denote the two processors. (The dashed lines in the DAG represent the mapping of jobs to processors.) □

Given a workload instance of the kind illustrated in Example 1 above, we seek to construct a schedule for the instance that guarantees to complete within the specified deadline. For the problem instance of Example 1, observe that at time zero both job v_1 and job v_2 are ready for execution upon processor P_1, while there are no jobs ready for execution upon processor P_2 at time zero. Figure 2 depicts the different schedules that are obtained for the instance of Example 1 if job v_1 is executed first or if job v_2 is executed first: as can be seen from Figure 2, executing v_1 before v_2 results in a missed deadline while executing v_2 before v_1 yields a correct schedule.

3 AN ILP REPRESENTATION

In this section we describe how the scheduling problem described in Section 2 above may be represented as an ILP. As mentioned in Section 1, representing our schedulability problem as a zero-one Integer Linear Program turns out to be surprisingly non-trivial – standard OR techniques do not seem to be directly applicable. The approach that we have developed, and that we describe in the remainder of this section, integrates some traditional OR techniques with some results from real-time scheduling regarding the relationship between the *demand bound function* [3] of a collection of independent real-time jobs and their feasibility upon a preemptive uniprocessor platform. In order to be able to apply the demand-bound function concept to our multiprocessor scheduling problem, our approach requires us to consider the jobs that have been assigned to each processor separately, and write constraints in which variables represent the instants at which each job begins and completes execution upon the processor to which it is assigned in any feasible schedule. Mutual dependence amongst the schedules upon different processors arises because of precedence constraints between jobs that must execute upon different processors: such dependencies are represented in our ILP formulation by adding constraints that enforce the requirement that a predecessor job must complete execution before its successor may begin to execute.

In somewhat more detail, for each node $v_i \in V$ of the DAG we define non-negative real-valued variables s_i and f_i to represent the instants at which the job represented by node v_i begins and completes execution upon the processor to which it has been assigned.[1]

[1] We reiterate that the processors are assumed to be preemptive: it is not required that the job v_i execute throughout the duration $[s_i, f_i]$. Rather, s_i denotes the first instant at which the job begins to execute, and f_i, the last instant at which it does so.

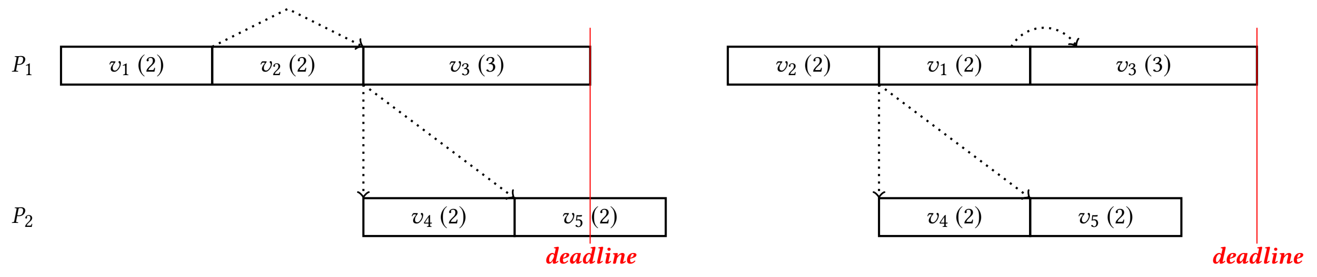


Figure 2: Gantt chart of possible schedules for the example instance of Example 1. Each "box" is labeled with the name of the job whose execution it represents; the maximum duration of this execution is written within parentheses. The dotted lines depict precedences.
The schedule to the left executes v_1 before executing v_2, while the one to the right executes v_2 before executing v_1. The schedule to the left fails to meet the deadline of 7, while the schedule to the right does meet the deadline (and is hence the correct one).

We represent information regarding the jobs' WCETs by adding a constraint

$$f_i \geq s_i + c(v_i) \tag{1}$$

for each vertex v_i. The requirement that all jobs complete by the specified deadline D is encoded in a constraint

$$f_i \leq D \tag{2}$$

for each v_i. Additionally, for each directed edge $(v_i, v_j) \in E$ of the DAG, we add the constraint

$$f_i \leq s_j \tag{3}$$

to encode the requirement that the job from which an edge is emanating must complete execution before the job to which the edge leads may begin to execute.

Example 2. We illustrate the steps above for the example instance of Example 1.

- Since there are five jobs v_1, v_2, v_3, v_4, and v_5, we have the ten variables s_i, f_i for $i = 1, 2, \ldots, 5$.
- We have the five constraints
$$s_i + c(v_i) \leq f_i$$
for $i = 1, 2, 3, 4$, and 5, to represent the jobs' WCETs. E.g, for $i \leftarrow 3$ this constraint, would be written as
$$s_3 + 3 \leq f_3$$
since $c(v_3) = 3$.
- We represent the requirement that all jobs meet the deadline by adding the five constraints
$$f_i \leq 7$$
for $i = 1, 2, 3, 4$, and 5.
- We have three additional constraints, representing the three edges (precedence constraints) in the graph:
$$\begin{array}{rrcl} & f_1 & \leq & s_3 \\ & f_2 & \leq & s_4 \\ \text{and} & f_2 & \leq & s_5 \end{array}$$

□

Considering an individual processor. Once the variables have been defined as described above, and precedence relationships represented via the Constraints 3, we consider each processor separately. That is, we individually consider, for each processor, all the jobs that have been assigned to that processor — we will write constraints for representing feasibility upon the processor under consideration. Recall that variables s_i and f_i represent the instants at which the job v_i begins and completes execution respectively; we can therefore characterize the workload on the processor as comprising a collection of independent jobs[2], one per job v_i assigned to this processor, that is released at time-instant s_i, has a WCET equal to $c(v_i)$, and has a deadline at time-instant f_i. Under the *demand-bound function* characterization [3] of feasibility upon a preemptive uniprocessor platform, a necessary and sufficient condition for this collection of jobs to be feasible is that for all intervals $[t_1, t_2]$ such that t_1 corresponds to the release time of some job and t_2 corresponds to the deadline of some job, the cumulative execution requirement of all the jobs with release time $\geq t_1$ and deadline $\leq t_2$ not exceed the interval duration $(t_2 - t_1)$. Accordingly for each 3-tuple v_i, v_j, and v_k of jobs that have been assigned to the processor under consideration, we define a non-negative real-valued variable c_{ijk} with the intended interpretation that

$$c_{ijk} \geq \begin{cases} c(v_k), & \text{if } (s_i \leq s_k) \bigwedge (f_k \leq f_j) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

That is, c_{ijk} is $\geq$ the WCET of v_k if v_k is scheduled entirely within the interval $[s_i, f_j]$, and $\geq$ zero otherwise. Informally, c_{ijk} is intended to represent (an upper bound on) the amount that job v_k contributes to the demand bound over the interval $[s_i, f_j]$, provided that $s_i < f_j$ and the interval is therefore non-degenerate. (Of course Expression 4 is not in the form of a linear constraint; we will describe later how this intended interpretation may be specified via linear constraints. We will see that doing so requires us to make use of zero-one integer variables.)

Example 3. Let us revisit the example instance of Example 1. We consider separately the jobs assigned to the first and the second processor.

[2]We emphasize that these jobs may be considered as being *independent*: precedence constraints between them are separately represented by constraints of the form given in Expression 3 above.

- For the first processor, we consider the jobs v_1, v_2, and v_3 that are assigned to it. We will therefore define $3 \times 3 \times 3 = \mathbf{27}$ variables c_{ijk} for $(i, j, k) \in \{1, 2, 3\} \times \{1, 2, 3\} \times \{1, 2, 3\}$.
- Similarly for the second processor, we consider only the jobs v_4 and v_5. Hence there will be $2 \times 2 \times 2 = \mathbf{8}$ variables c_{ijk} for $(i, j, k) \in \{4, 5\} \times \{4, 5\} \times \{4, 5\}$.

□

The demand bound function characterization of preemptive uniprocessor feasibility can be stated in terms of these c_{ijk} variables in the following manner. For all intervals $[s_i, f_j]$ such that both job v_i and job v_j are assigned to the processor under consideration, it must be the case that

$$\sum_k c_{ijk} \leq (f_j - s_i) \tag{5}$$

We point out that in fact, under the demand bound function characterization of preemptive uniprocessor feasibility, only intervals $[s_i, f_j]$ *for which the entire scheduling windows* $[s_i, f_i]$ *and* $[s_j, f_j]$ *of both the jobs* v_i *and* v_j *fall within the interval* $[s_i, f_j]$ need to be checked for Condition 5 — this follows from the observation that if either of the jobs does not fall within this interval, then that job does not contribute to the demand over the interval.

Introducing zero-one integer variables. All the variables introduced thus far — the start-time s_i and finish-time f_i for each job v_i, and the c_{ijk} variables for each 3-tuple of jobs v_i, v_j, and v_k that are assigned to the same processor with the intended interpretation of Expression 4 — are allowed to take on arbitrary real values. We now introduce some additional variables that are restricted to taking on the (integer) values zero or one only. These additional zero-one integer variables enable us to write constraints forcing the c_{ijk} variables to take on their intended meaning (as defined by Expression 4), and to express Expression 5 as a linear constraint.[3]

The zero-one integer variables that we introduce are as follows. For each pair of jobs v_i and v_j that are assigned to the same processor, we have two variables, x_{ij} and y_{ij}, with the intended interpretations that

$$x_{ij} = \begin{cases} 1, & \text{if } s_i \leq s_j \\ 0, & \text{otherwise} \end{cases}$$

and

$$y_{ij} = \begin{cases} 1, & \text{if } f_i \leq f_j \\ 0, & \text{otherwise} \end{cases}$$

That is, these zero-one integer variables specify the *ordering* of the start-times and the finish-times of the jobs: assigning x_{ij} the value 0 is equivalent to asserting that s_i occurs no later than s_j while assigning x_{ij} its other possible value, 1, is equivalent to asserting that s_i occurs after s_j (and similarly for the value assigned to y_{ij} establishing the relative ordering of f_i and f_j). This intended interpretation is achieved by means of a fairly standard method from Operations Research, of adding the following constraints; here, M denotes a large positive constant.

$$\begin{array}{rcl} s_i & \geq & s_j - M \cdot x_{ij} \\ s_j & \geq & s_i - M \cdot (1 - x_{ij}) \\ f_i & \geq & f_j - M \cdot y_{ij} \\ f_j & \geq & f_i - M \cdot (1 - y_{ij}) \end{array} \tag{6}$$

We can see why these constraints achieve the intended interpretation by considering the first two inequalities of Inequalities 6:

- If $x_{ij} = 1$, then the first inequality requires that $s_i \geq s_j - M$; since M is assumed to be a very large positive integer, this inequality does not constrain the possible value that may be assigned to s_i. However, the second inequality requires that $s_j \geq s_i - M \cdot (1 - 1)$, i.e., $s_j \geq s_i$, as intended.
- If $x_{ij} = 0$, then the first inequality requires that $s_i \geq s_j - M \cdot 0$, i.e., $s_i \geq s_j$, as intended. The second inequality requires that $s_j \geq s_i - M \cdot (1 - 0)$, i.e., $s_j \geq s_i - M$ (and since M is assumed to be a very large positive integer, this inequality does not constrain the possible value that may be assigned to s_j).

The second two inequalities of Inequalities 6 similarly achieve the intended interpretation regarding the relative ordering of f_i and f_j.

Example 4. Let us revisit the example instance of Example 1, and consider the first processor (consideration of the second processor is similar). For the three jobs on the first processor, a total of $3 \times 2 = \mathbf{6}$ pairs (i, j) need to be defined and constrained as specified in Expression 6. As an optimization, we can exploit symmetry and reduce the number of constraints that need to be written by observing that $x_{ij} = 1 - x_{ji}$ (and analogously for the y_{ij}'s). Hence, the following constraints are the only ones needed:

- For each $(i, j) \in \{(1, 2), (1, 3), (2, 3)\}$

$$\begin{array}{rcl} s_i & \geq & s_j - M \cdot x_{ij} \\ s_j & \geq & s_i - M \cdot (1 - x_{ij}) \\ f_i & \geq & f_j - M \cdot y_{ij} \\ f_j & \geq & f_i - M \cdot (1 - y_{ij}) \end{array}$$

- For each $(i, j) \in \{(2, 1), (3, 1), (3, 2)\}$

$$\begin{array}{rl} x_{ij} & = 1 - x_{ji} \\ y_{ij} & = 1 - y_{ji} \end{array}$$

□

We may use these zero-one variables that we have defined to achieve the intended interpretation for the c_{ijk} variables as specified in Expression 4, by writing the following constraint for each 3-tuple (i, j, k):

$$c_{ijk} \geq c(v_k) - M \cdot (2 - x_{ik} - y_{kj}) \tag{7}$$

It may be verified that this constraint requires c_{ijk} to be $\geq c(v_k)$ if $(s_i \leq s_k)$ and $(f_k \leq f_j)$; otherwise, it does not constrain the value of c_{ijk} at all.

Example 5. For the first processor of the example instance of Example 1, the 3-tuple (i, j, k) may take on any value in $\{1, 2, 3\} \times$

[3]Note that although Expression 5 appears to already be in linear-constraint form, it is not quite so since it is only defined for some values of i and j – those for which $s_i \leq f_j$. But since the s_i and the f_j values are *variables*, we cannot a priori determine for which values of i and j constraints of the form Expression 5 need to be written — this will be determined by the values that get assigned to the s_i's and the f_j's in any solution to the ILP that we are constructing.

$\{1,2,3\} \times \{1,2,3\}$. For each such (i,j,k), we therefore add the constraint

$$c_{ijk} \geq c(v_k) - M \cdot (2 - x_{ik} - y_{kj})$$

For some of the c_{ijk} variables, it is possible to determine the actual values that they are required to take in any feasible solution. Such variables may have their values initialized in the ILP as an additional optimization — in general, the more constrained the optimization problem the quicker an ILP solver is able to find a solution. For instance, it is evident that for any i the variable c_{iii} takes on the value $c(v_i)$ since $s_i \leq s_i$ and $f_i \leq f_i$ are trivially true. Some other such initializations are possible; some of the obvious ones for our example are listed below.

$$\begin{aligned} c_{111} &= c(v_1) \\ c_{222} &= c(v_2) \\ c_{333} &= c(v_3) \\ c_{121} &= c(v_1) \\ c_{122} &= c(v_2) \\ c_{21k} &= 0 \text{ for } k \in \{1,2,3\} \end{aligned}$$

(We point out that adding these initializations only improves efficiency – they have no impact whatsoever on feasibility. Hence adding them is optional.) □

It remains to add constraints that characterize preemptive uniprocessor feasibility upon each processor. As discussed above, this is equivalent to requiring that Expression 5 be satisfied for all intervals $[s_i, f_j]$ such that both job v_i and job v_j are assigned to the same processor, and the entire scheduling windows of both jobs v_i and v_j fall within the interval. For such $[s_i, f_j]$, Constraint 5 can be represented as follows:

$$\sum_k c_{ijk} \leq (f_j - s_i) + M \cdot (2 - x_{ij} - y_{ij}) \tag{8}$$

were the index k ranges over all jobs v_k that are assigned to the same processor as v_i and v_j.

Example 6. For the first processor of the example instance of Example 1, (i,j) may take on any value in $\{1,2,3\} \times \{1,2,3\}$. For each such (i,j), we add the constraint

$$\sum_k c_{ijk} \leq (f_j - s_i) + M \cdot (2 - x_{ij} - y_{ij})$$

As in Example 5, some optional simplification is possible that improves efficiency without impacting feasibility. For example, we may safely delete the constraint corresponding to $(i,j) \leftarrow (2,1)$:

$$\sum_k c_{21k} \leq (f_1 - s_2) + M \cdot (2 - x_{21} - y_{21})$$

(The reason we are able to delete this constraint is because of the precedence relationship from v_1 to v_2 guarantees that $f_1 \leq s_2$ and hence both x_{21} and y_{21} will equal zero, and this constraint will become degenerate.) □

Putting the pieces together. The various steps discussed above for representing an instance of our DAG scheduling problem as a zero-one integer linear program are collected together in Figure 3. (Only the essential steps are enumerated; the optional optimizations introduced in Examples 5 and 6 are not included.) Once the ILP has been formulated, it is handed off to an ILP-solver which determines whether it is feasible and if so, assigns appropriate values to the variables in the ILP. If a solution is found, the variables of interest to us are the s_i and f_i values — the instant at which each job begins and completes execution upon the processor to which it has been assigned. We will use these values to construct the actual schedule upon the individual processors, by (i) modeling each job v_i as being released at time-instant s_i, having a WCET equal to $c(v_i)$, and a deadline at time-instant f_i; and (ii) constructing the EDF-generated schedule of these jobs.

Given the DAG $G = (V, E)$ with the processor assignment $\pi(\cdot)$ and WCETs $c(\cdot)$, obtain a zero-one integer linear program as follows:

(1) For each job $v_i \in V$
- Introduce the non-negative real-valued variables s_i and f_i
- Add the constraints

$$\begin{aligned} f_i &\geq s_i + c(v_i) \\ \text{and } f_i &\leq D \end{aligned}$$

(2) For each edge $(v_i, v_j) \in E$ add the constraint

$$f_i \leq s_j$$

(3) For each (i,j) such that $\pi(v_i) = \pi(v_j)$
- Introduce the zero-one integer variables x_{ij} and y_{ij}
- Add the four constraints

$$\begin{aligned} s_i &\geq s_j - M \cdot x_{ij} \\ s_j &\geq s_i - M \cdot (1 - x_{ij}) \\ f_i &\geq f_j - M \cdot y_{ij} \\ f_j &\geq f_i - M \cdot (1 - y_{ij}) \end{aligned}$$

where M is a large positive integer.

(4) For each 3-tuple (i,j,k) such that $\pi(v_i) = \pi(v_j) = \pi(v_k)$
- Introduce the non-negative real-valued variable c_{ijk}
- Add the constraint

$$c_{ijk} \geq c(v_k) - M \cdot (2 - x_{ik} - y_{kj})$$

(5) For each (i,j) such that $\pi(v_i) = \pi(v_j)$, add the constraint

$$\sum_k c_{ijk} \leq (f_j - s_i) + M \cdot (2 - x_{ij} - y_{ij})$$

Figure 3: Obtaining an ILP Representation of the Schedulability of a DAG

How large is this ILP? For an instance of our scheduling problem comprising n jobs, it is evident that

(1) The number of s_i and f_i variables introduced in Step 1 of Figure 3 is equal to $2n$, while the number of constraints introduced in this step is also equal to $2n$.
(2) The number of constraints introduced in Step 2 of Figure 3 is equal to $|E|$, the number of edges in the DAG.
(3) The number of zero-one variables x_{ij} and y_{ij} that are introduced in Step 3 of Figure 3 is at most $2n^2$, this value being reached if all the jobs are assigned to the same processor. The number

of constraints introduced in this step is twice the number of variables that are introduced.

(4) The number of c_{ijk} variables introduced in Step 4 of Figure 3 is at most n^3, this value being reached if all the jobs are assigned to the same processor. The number of constraints introduced in this step is equal to the number of variables that are introduced.

(5) Finally, the number of constraints introduced in Step 5 of Figure 3 is at most n^2.

We thus see that we have an ILP with $O(n^3)$ real-valued variables, $O(n^2)$ zero-one integer variables, and $O(n^3)$ constraints. That allows us to conclude that *the ILP is of size polynomial in the number of jobs*.

An additional observation. We mention in passing that the technique we derived here is easily modified to deal with a generalization of the considered problem, in which each individual job may have its own arrival-time and deadline in addition to precedence constraints and the overall deadline.

Implementation experience. We have implemented a Python program that reads in problem instances specified as described in Section 2, constructs the ILP representation, and solves this ILP using the Gurobi[4] solver, Version 7.0.2 (mac65, Python). Our primary objective in doing this implementation was to gather some confidence that our ILP formulation is correct; to this end we were able to verify that it does indeed obtain the correct solution for several simple examples (such as the one in Example 1). We plan to submit our implementation code for artefact evaluation if this paper is accepted, and to share it with other interested researchers. While we did not conduct extensive experimental evaluations – such evaluation, on real workloads occurring in actual CPS applications, is planned as future work – we did observe that the Gurobi ILP solver appears to be a very powerful one that incorporates several clever heuristics for optimizing the process of finding a solution; for instance, it obtained a solution to the DAG of Example 1 in less than 0.1 seconds on a MacBook Pro (2.3 GHz Intel Core i5; 16GB Memory). Even on somewhat larger instances – the largest one we tested on had 50 jobs and 50 precedence constraints – the running time never exceeded 1 second.

4 RELATED WORK

We were not able to find any prior work on designing exact algorithms for solving our scheduling algorithm. The only somewhat related algorithm that we found is the *Due-Date Modification* (DDM) algorithm [1]. Under DDM, the deadline of each job is modified to be the smaller of its current deadline and the latest start time (deadline minus WCET) of its successor jobs, and jobs are prioritized for execution according to their modified deadlines. Example 1 (Section 2) bears witness to the fact that DDM is not an optimal algorithm: since the modified deadline of v_1 is $7 - 3 = 4$ while the modified deadline of v_2 is $7 - 2 = 5$, DDM would prioritize v_1 over v_2 and thereby yield the left (incorrect) schedule depicted in Figure 2 rather than the right, correct, one.

[4]https://www.gurobi.com

5 CONTEXT AND CONCLUSIONS

Marko Bertogna, in his keynote address at RTNS 2019, had posed an agenda for future research on real-time scheduling that is motivated by the issues and problems he has faced in building an extremely complex safety-critical cyber-physical system – a self-driving car. In this paper, we describe our efforts at addressing one of the simplest problems in his wish-list of problems: developing an exact algorithm for synthesizing schedules, where possible, for precedence-constrained collections of jobs each of which is restricted to executing upon a specified processor of a multiprocessor platform. Solving even this simple problem turned out to be surprisingly challenging: it required us to draw upon, and integrate, disparate ideas from Operations Research and real-time scheduling theory in order to synthesize the ILP, and then fall back on results from real-time scheduling theory – the optimality of preemptive uniprocessor EDF – in order to synthesize the actual schedule using the individual jobs' start-times and completion times as determined by the solution to the ILP. We emphasize that the integration of *ordering variables* – a fairly standard idea in the OR literature – with *demand-bound function* characterization of preemptive uniprocessor feasibility is, to our knowledge, a novel contribution of this work. We find this idea to be powerful and exciting, and are eager to explore further applications of it to obtain ILP representations of other real-time scheduling problems.

Our agenda for future work includes the following:

- As mentioned earlier, we have implemented our algorithm and have performed some preliminary experiments to validate its correctness, but have not yet done extensive experimental evaluation. We plan to do such evaluation in the near future, using real workloads obtained from actual CPS applications (such as Bertogna's), in order to obtain a better understanding of the scalability of our approach.
- We will seek to apply the lessons and insights we have obtained here to derive ILP-based exact schedulability tests for workloads that are expressed using generalizations to the model assumed in this paper. One particular generalization of interest is this: rather than restricting each job to execute upon one specific processor, what if they were instead restricted to executing upon a subset of the available processors? (We point out that this is not likely to be a straightforward extension of our current approach since the demand-bound function abstraction is no longer applicable, and some other ideas from real-time scheduling theory are needed.)

REFERENCES

[1] Kenneth Baker and J. Bertrand. 1982. A Dynamic Priority Rule for Scheduling against Due-Dates. *Journal of Operations Management* 3 (1982), 37–42. Issue 3.

[2] Kenneth R. Baker and Dan Trietsch. 2009. *Principles of Sequencing and Scheduling.* Wiley Publishing.

[3] S. Baruah, A. Mok, and L. Rosier. 1990. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium*. IEEE Computer Society Press, Orlando, Florida, 182–190.

[4] Slim Ben-Amor, Liliana Cucu-Grosjean, and Dorin Maxim. 2019. Worst-case response time analysis for partitioned fixed-priority DAG tasks on identical processors. In *24th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2019, Zaragoza, Spain, September 10-13, 2019*. IEEE, 1423–1426. https://doi.org/10.1109/ETFA.2019.8869147

[5] M. Garey and D. Johnson. 1979. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman and company, NY.

[6] Klaus Jansen. 1994. Analysis of scheduling problems with typed task systems. *Discrete Applied Mathematics* 52, 3 (1994), 223 – 232.