Discrete Optimization

Dynamic resource allocation: A flexible and tractable modeling framework

Dimitris Bertsimas a,∗, Shubham Gupta b, Guglielmo Lulli c

a Operations Research Center, Massachusetts Institute of Technology, E40-147, Cambridge, MA 02139-4307, United States
b Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139-4307, United States
c Dept. of Informatics, Systems and Communication, University of Milano-Bicocca, 20126 Milano, Italy

A R T I C L E   I N F O

Article history:
Received 7 December 2012
Accepted 27 October 2013
Available online 14 November 2013

Keywords:
Applications of integer optimization
Resource allocation
Scheduling
Fairness

A B S T R A C T

This paper presents a binary optimization framework for modeling dynamic resource allocation problems. The framework (a) allows modeling flexibility by incorporating different objective functions, alternative sets of resources and fairness controls; (b) is widely applicable in a variety of problems in transportation, services and engineering; and (c) is tractable, i.e., provides near optimal solutions fast for large-scale instances. To justify these assertions, we model and report encouraging computational results on three widely studied problems – the Air Traffic Flow Management, the Aircraft Maintenance Problems and Job Shop Scheduling. Finally, we provide several polyhedral results that offer insights on its effectiveness.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Allocation of resources over time is a problem of significant importance that many organizations in industry, government and education face. Correspondingly, resource allocation problems have received considerable attention in the Operations Research literature. In real world applications of resource allocation problems, specific issues arise: assignment of requests to resources over time, allowing the flexibility of utilizing alternative resources to complete the requests, fairness issues among different requests, among others. There has been extensive work on specific examples of resource allocation problems (for example, the extensive literature on Job Shop Scheduling). Still, to the best of our knowledge, we are not aware of a unified approach that can be easily modified to accommodate variations, while simultaneously being computationally tractable for large scale instances. On the contrary, it is widely believed that optimization might not be the right approach for certain classes of resource allocation problems such as scheduling for example. In fact, commercial solvers like ILOG for scheduling problems are typically not optimization based, but rather rule based.

Our aspiration in this paper is to develop a widely applicable, flexible and tractable modeling framework based on binary optimization that is capable of modeling and solving large scale instances for a variety of resource allocation problems over time. The paper has its intellectual origins with the work of Bertsimas and Stock (1998) on air traffic flow management, which is a resource allocation problem over time. In this problem, the resources are airports and sectors of the airspace, the requests are flights and the objective is to minimize delays in the system. To this date, within the scope of air traffic flow management, this modeling approach has proven successful, as it continues to be used extensively by several researchers and practitioners around the world. Two significant generalizations of the model in the context of air traffic flow management that suggest flexibility include: (a) the work of Bertsimas and Gupta (submitted for publication) that shows how fairness issues among airlines can be modeled in a computationally effective way, and (b) the work of Bertsimas, Lulli, and Odoni (2011) that allows the use of alternative routing of flights, when the current resources decrease possibly because of bad weather. Given the success of this modeling approach to air traffic flow management, it is natural to ask:

(a) Can we develop a modeling approach to general dynamic resource allocation problems that is flexible, tractable and widely applicable?
(b) Can we give some insights (both theoretical and empirical) on the reasons of the approach effectiveness in the context of resource allocation?

The broad framework we have in mind for Dynamic Resource Allocation Problem (DRAP) is as follows. The primitive quantities are: a set of resources \( R \) and a set of requests \( I \) belonging to a set of owners \( O \) that need to be processed by these resources over
a time horizon $T$. Each request $i$ needs to be completed by certain time and it can be completed by using alternative sets of resources $R^1_i, \ldots, R^n_i$. Different allocations of resources to requests over time result in delays if the request is completed after its desired time. The overall goal is to allocate resources to requests over time in order to complete the requests as efficiently as possible (minimum delay), potentially using alternative resources and ensuring that the distribution of delays amongst these requests (and implicitly to their owners) is fair.

In the opening sentence of his 1963 book Linear Programming and Extensions (Dantzig, 1963), George Dantzig writes: “the final test of any theory is its capacity to solve the problems which originated it”. Motivated by this philosophy, we first present the modeling framework, and then use it to model and solve the following three widely-studied problems:

1. **Air Traffic Flow Management (ATFM).** Air Traffic Flow Management aims to prevent local demand-capacity imbalances by adjusting the flows of aircraft on a national or regional basis. ATFM models optimize for each flight the time of departure, the route selected, the time required to traverse each sector, and the time of arrival at the destination airport, taking into account the capacity of all the elements of the air traffic management system. In our generalization, each request will represent a scheduled sector-based path from takeoff through landing of a flight. The resources are the runways for takeoff and landing as well as the en-route airspace sectors. The fairness controls will be imposed at the destination airports where the final flight sequences are desired to be as close to the original scheduled sequence as possible. Finally, the alternative resources allows the option of flying a different origin–destination route for each aircraft.

2. **Aircraft Maintenance Problem (ACM).** The Aircraft Maintenance Problem is an important dynamic resource allocation problem in the airline industry. In this problem, an aircraft requests a large number of inspection and repair activities for which resources like equipments, sophisticated tools and highly specialized skills are used over several months.

3. **Job Shop Scheduling (JSP).** Job Shop Scheduling is one of the most notoriously hard combinatorial optimization problem. It entails processing a set of jobs on machines with the objective of minimizing some function of the completion times of the jobs (examples include the makespan, i.e., the maximum completion time and the minsum, i.e., the average completion time), subject to two requirements: (a) the sequence of machines for each job is prescribed; and (b) each machine can process at most one job at a time, and the schedule must be non-preemptive. In this paper, we model this problem within our framework and solve it for the two most widely applicable objective functions we mentioned earlier, makespan and minsum.

1.1. Literature review

In the literature, there is a lack of a scheduling framework of the form inspired in this research effort. Nonetheless, we attempt to enumerate the relevant papers which have the same taste. Bar-Noy, Bar-Yehuda, Freund, Naor, and Schieber (2001) present a unified approach to approximating resource allocation and scheduling. There has been recent work on using approximate dynamic programming (ADP) methods to solve dynamic resource allocation problems which overcome the “curses of dimensionality” of standard dynamic programming methods. Using ADP, Powell and Topaloglu (2005) describe solution strategies for large-scale resource allocation problems under uncertainty. Similarly, Gocgun and Ghatre (2012) develop an ADP method that uses Lagrangian relaxation and constraint generation for dynamic stochastic resource scheduling problems. ADP techniques have also been used by Erdelyi and Topaloglu (2010) to solve a dynamic capacity allocation problem and by Powell and Van Roy (2004) who present computationally efficient algorithms for a mathematical model of dynamic resource allocation motivated by problems in transportation and logistics. In terms of specific applications, Menasce and Casalicchio (2004) design a framework for resource allocation in grid computing, whereas, Alhusaini, Prasanna, and RagHAVENDRA (1999) focus on a unified resource scheduling framework for heterogeneous computing environments.

We next review relevant literature on the three problems discussed in this section.

- **ATFM.** This is an extensively studied problem. Starting with the first paper by Odoni (1987) in 1987, there have been a plethora of proposals attacking various aspects of the problem. One of the most comprehensive models is by Bertsimas and Stock (1998) which considers the problem of controlling release times and speed adjustments of aircraft while air-borne for a network of airports taking into account the capacitated airspace. For a detailed survey of the various contributions and a taxonomy of all the ATFM problems, see Bertsimas and Odoni (1997) and Hoffman, Mukherjee, and Vossen (2011).
- **JSP.** There have been a plethora of proposals for JSP which utilize both exact methods and heuristic approaches. The earliest exact method can be traced back to Giffler and Thompson (1960) in 1960. Thereafter, many branch and bound type algorithms were developed by Carlier and Pinson (1989), Applegate and Cook (1991), Brucker, Jurisch, and Sievers (1994). An approach based on exploiting the disjunctive graph representation of JSP was developed by Adams, Balas, and Zawack (1988) and Balas and Vazacopoulos (1998). These are known as Shifting Bottleneck methods.

1.2. Structure of the paper

Section 2 describes the proposed binary optimization framework. It introduces the constraints for scheduling, usage of alternative resources and fairness controls. Section 3 enumerates several examples of problems and formulates the three applications mentioned in this section within our framework. Section 4 presents theoretical evidence (by providing several polyhedral insights) and Section 5 presents computational evidence on the strength of the overall framework. Section 6 contains concluding remarks.

1.3. Notation and preliminaries

We denote scalar quantities by lowercase, non-bold face symbols (e.g., $w \in \mathbb{R}$, $k \in \mathbb{N}$), vector quantities by lowercase, boldface symbols (e.g., $w \in \mathbb{R}^n$, $n > 1$), and matrices by uppercase, boldface symbols (e.g., $A \in \mathbb{R}^{n \times m}$, $n > 1$).
2. The modeling framework

In this section, we present the proposed binary optimization framework for DRAP. Our framework represents time as discrete time intervals. This choice renders the possibility of formulating the various constraints in the DRAP in a relatively simple manner and usually leads to well-structured optimization models (Floudas & Lin, 2005). Moreover, formulations of discrete-time models generally compute solutions of better quality with respect to formulations of other models, as experienced – for instance – on the single machine scheduling problem by Keha, Khowala, and Fowler (2009). For these reasons, a gigantic number of discrete-time models is available in the literature on scheduling problems and more in general with resource assignment problem over time. The earliest research contributions that employed discrete-time formulations are due to Bowman (1959) and Manne (1960) for Job Shop Scheduling problems. Notable subsequent developments include the work by Pritsker, Watters, and Wolfe (1969) for resource limited multiproject and Job Shop Scheduling.

2.1. Data

The data for this model is composed of requests, \( r \in \mathcal{R} \), each of which utilizes one of the many alternative sets of resources, \( \mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k \subseteq \mathcal{R} \), to be processed. We may also have a subset of requests that must be scheduled, which we refer to as \( \mathcal{M} \). Time is discretized, such that we index time by \( t \in \{0, \ldots, T-1\} \) representing the interval \([t,t+1)\). Each resource, \( r \in \mathcal{R} \), is constrained based on the time of utilization, thus we write \( b_r \) for the capacity of resource \( r \) over the interval \([t,t+1)\). Each request, \( r \in \mathcal{R} \), has a priority, \( p_r \in [0,1] \).

Each request \( i \) specifies (a) the duration \( d_i \) it requires from resource \( r \), (b) a set of feasible times \( T_i \) that it can be processed by resource \( r \), and (c) the amount \( f_i \) of resource \( r \) it utilizes. Request \( i \) may also specify sets \( Q(i,j) \) and \( O(i,j) \) of ordering conditions on the resources it utilizes (based on the alternative sets of resources \( \mathcal{R}_i \)) as follows:

- \((r',r,u) \in Q(i,j)\) for \( r, r' \in \mathcal{R}_i \) with the requirement that request \( i \) must start utilizing resource \( r' \) at least \( u \) time units after it starts utilizing resource \( r \).
- \((r',r,v) \in O(i,j)\) for \( r, r' \in \mathcal{R}_i \) with the requirement that request \( i \) must start utilizing resource \( r' \) at most \( v \) time units after it starts utilizing resource \( r \).

Furthermore, we let \( e_{d_i} \) be the time resource \( r \) processing request \( i \), needs to transition to start processing request \( i \). Let \( E(r) = \{(i, \bar{i}, e_{d_i}) : i, \bar{i} \in \mathcal{R}, r \in \mathcal{R} \} \) be all the ordering conditions on the requests.

2.2. Notation

- \( T \): set of discrete time intervals \([t,t+1)\) dividing the time horizon,
- \( \mathcal{I} \): set of requests,
- \( \mathcal{O} \): set of owners,
- \( \mathcal{R} \): set of resources,
- \( \mathcal{M} \subseteq T \): subset of requests which must be scheduled,
- \( b_r \): capacity of resource \( r \) over time interval \( t \),
- \( T_i = \{T_{i1}, \ldots, T_{it} \} \): set of feasible times that request \( i \) can be processed by resource \( r \),
- \( \mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k \subseteq \mathcal{R} \): alternative sets of resources that can be used to process request \( i \),
- \( \mathcal{R}_i = \bigcup_{r \in \mathcal{R}_i} \mathcal{R}_r \): subset of resources which are capable of being utilized by request \( i \),
- \( t_h \in \mathcal{R} \): (unique) final resource that needs to be utilized by request \( i \) to complete processing,
- \( \ell_i \in \mathbb{N} \): duration that request \( i \) must utilize resource \( r \),
- \( f_r \in \mathbb{R} \): quantity of resource \( r \) that is utilized by request \( i \),
- \( Q(i) = \cup_{j \in \mathcal{R}_i} Q(i,j) \) and \( O(i) = \cup_{j \in \mathcal{R}_i} O(i,j) \): resource ordering conditions for request \( i \),
- \( O(i) = Q(i) \cup O(i) \): set of all resource ordering conditions for request \( i \),
- \( E(r) \): set of all request ordering conditions for resource \( r \),
- \( p_r \in [0,1] \): priority of request \( i \).

In addition to the above, we introduce some more notation specific to the alternative resources and fairness components in their respective sections (Sections 2.4 and 2.5).

Remark 1. We assume that for each request \( i \), there is a unique resource \( (t_h \in \mathcal{R}_i) \) that represents the final resource that request \( i \) utilizes to complete processing. This assumption is without loss of generality and eases the modeling required to enforce that the requests in \( \mathcal{M} \) are scheduled.

To better visualize the concept of alternative resources we use a resource-on-vertex digraph representation \( G(V',A') \) (for request \( i \)). The resources of the alternative sets \( \mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k \) are associated with the set of vertices \( V' \) while the set of arcs \( A' \) represents the set of ordering relations \( O(i) \) between the resources of the set \( \mathcal{R}_i = \cup_{r \in \mathcal{R}_i} \mathcal{R}_r \). In the general case the resources are in a one-to-many correspondence with the vertices of the set \( V' \). We present a concrete example to make the notation clearer.

Example 2.1. Fig. 1 depicts two examples of alternative resources. On the left hand side, four subsets of alternative resources are available for the execution of request \( i \) : \( \mathcal{R}_1 = \{1,2,6,7\}, \mathcal{R}_2 = \{1,3,5,7\}, \mathcal{R}_3 = \{1,3,6,7\} \) and \( \mathcal{R}_4 = \{1,4,5,7\} \). This, \( \mathcal{R}_0 = \{1, \ldots, 7\} \). The use of either one of these resources imposes restriction on the use of the subsequent resource. For instance, if Resource 2 is used then Resource 6 has to be used, while if Resource 3 is used then either Resource 5 or 6 has to be used. Based on these \( \mathcal{R}_0 \)'s, examples of the resource ordering conditions \( O(i) \) are: \( (1,4,9) \in O(i) \) and \( (3,5,10) \in O(i) \). The former specifies that request \( i \) must start utilizing Resource 4 at least 9 time units after it starts utilizing Resource 1, while the latter specifies that request \( i \) must start utilizing Resource 5 at most 10 time units after it starts utilizing Resource 3. Referring to the resource-on-vertex digraph representation \( G(V',A') \), the set of vertices \( V' \) correspond to resources \( \{1, \ldots, 7\} \), while the set of arcs \( A' \) would contain (1,4) and (3,5) (among other ordering relations). The digraph on the right hand side of Fig. 1 represents the case of one-to-many correspondence between resources and vertices of the set \( V' \). Resource 3 is in correspondence with vertex 3' and vertex 3'' to correctly represent the following case: \{1,2,3,5\} and \{1,3,4,5\} are two subsets of alternative resources but set \{1,3,5\} is not. In this case, the ordering relation \( (3,5,10) \) is valid only for the first subset of alternative resources while ordering relation \( (3,4,8) \) is valid only for the second set alternative resources.

2.3. Scheduling constraints

For each request, we need to decide whether to accept the request, and if so, when to schedule its resource utilizations to satisfy the problem constraints. To do this, we define the following decision variables. The first set corresponds to enforcing the requests that have to be scheduled.

\[ x_i = \begin{cases} 
1, & \text{if request } i \text{ is accepted,} \\
0, & \text{otherwise.} 
\end{cases} \]
Note that $x_i = 1, \forall i \in M$.

The second and the more fundamental set of variables correspond to the scheduling decisions for every request demanding a slice of each resource.

$$w_{t,i} = \begin{cases} 1, & \text{request } i \text{ has begun utilizing resource } r \in R_i \text{ by } \\
\text{discrete time } t \in T'_i, \\
0, & \text{otherwise}. \end{cases}$$

The definition of the $w_{t,i}$ variables as "by" rather than "at" is critical for our modeling framework. The reason is that this definition of the variables leads to a natural modeling of a class of connectivity constraints. Moreover, it is quite flexible as it allows us to calculate many interesting properties as a linear combination of the $w_{t,i}$ variables. For instance, if we would like to determine whether request $i$ starts utilizing resource $r$ at time $t$, we can do so by computing $w_{t,i} - w_{t-1,i}$. Additionally, if we would like to determine whether request $i$ is actively utilizing resource $r$ at time $t$, we can do so by computing $w_{t,i} - w_{t-1,i}$. This choice of the scheduling variables is also supported by the computational experience reported in the paper by Andreotta and Brunetta (1998). In this paper, the authors compare several formulations of the ATFM problem obtained by using different definitions of the scheduling decision variables. The formulation that uses the chosen scheduling variables outperforms the other formulations computationally.

The following constraints ensure the allocation of resources over time:

$$\sum_{r \in R_i} f_r \cdot (w_{t,i} - w_{t-1,i}) \leq b_r \quad \forall r \in R_i, t \in T_i. \quad (1a)$$

$$w_{t,i} \leq w_{t+1,i} \quad \forall i \in I, t \in R_i, t \in \{T_i, \ldots, T'_i - 1\}. \quad (1b)$$

$$w_{t,i} \leq w_{t-1,i} + \mathbb{1}(r, r', u) \cdot u \quad \forall i \in I, (t, r, r', u) \in \mathcal{O}(i), \forall t \in T'_i, t - u \in T'_i. \quad (1c)$$

$$w_{t+1,i} \leq w_{t,i} + \mathbb{1}(r, r', t) \cdot u \quad \forall i \in I, (t, r, r', t) \in \mathcal{O}(i), t \in T'_i, t - u \in T'_i. \quad (1d)$$

$$w_{t,i} \leq w_{t-1,i} - \mathbb{1}(r, r', t) \cdot u \quad \forall i \in I, \forall (t, i, \epsilon_{dr_i}) \in \mathcal{E}(i), \forall t \in T'_i, t - l_u - \epsilon_{dr_i} \in T'_i. \quad (1e)$$

$$w_{t+1,i} - \mathbb{1}(r, r', t) \cdot u \quad \forall i \in I, \forall t \in T_i. \quad (1f)$$

Constraints (1a) enforce that all the capacity limits of resources are satisfied. These constraints correctly account for the use of resource $r$ at time period $t$ if requests utilize resource $r$ for a fixed duration, i.e., for $l_u$ consecutive time periods. However, in some applications, the use of a resource by a request does not have a fixed duration, but continues until the request starts using another resource. In this case, to account for the quantity of resource $r$ that is used at time period $t$, constraint (1a) has to be modified as follows:

$$\sum_{r \in R_i} f_r \cdot \max \left( w_{t+1,i} - \sum_{r \in R_i \cap \mathbb{1}(r, r', u) \cdot u} w_{t+1,i} \right) \leq b_r \quad \forall r \in R_i, t \in T_i. \quad (1g)$$

Constraint (1b) ensures time-connectivity, i.e., if a request $i$ begins utilizing resource $r$ by time $t$, then $w_{t,i}$ has to take value 1 for all the subsequent time periods ($t \geq t$). Constraint (1c) and (1d) are resource-connectivity constraints. Specifically, constraint (1c) specifies that request $i$ must start utilizing resource $r$ at least $u$ time units before it starts utilizing resource $r'$ while constraint (1d) enforces request $i$ to start utilizing resource $r'$ within $v$ time units of starting to utilize resource $r$. Constraint (1e) ensures request-connectivity. They handle the cases in which a request is continued, i.e., the first request (say $i$) has to be followed by a subsequent request (say $j$) within some user-specified time interval ($\epsilon_{dr_j}$). Constraint (1f) ensures that the semantics concerning whether a request is accepted or not is satisfied. If the request is not accepted, then all $w_{t,i}$ variables are set to 0 ensuring that it is not scheduled. In contrast, if the request is accepted, then $w_{t,i}$ is set to 1, enforcing that the request needs to be processed at its final resource (by the deadline).

**Remark 2.** Every constraint of the type (1b)–(1d) and (1e) contributes only two non-zero entries in the corresponding row of the constraint matrix. The resulting sparsity of the constraint matrix is an important factor contributing to the tractability of our framework.

The scheduling component of the DRAP framework includes the resource constrained project scheduling problem (RCSP) as a special case. Specifically, if we consider the case of a single request, i.e., $|Z| = 1$, and $u = l_u$ is the time spent in using resource $r$, then Constraints (1a)–(1c) define the solutions of the RCSP, see for example Brucker, Drexel, Mohring, Neumann, and Pesch (1999).

### 2.4. Constraints for alternative resources

In this section, we introduce the constraints required to incorporate the flexibility of alternative resources. Alternative resources allows to model many important settings of real applications, for instance, rerouting in the air traffic management (Bertsimas et al., 2011) and multimode execution of activities in resource constrained project scheduling (RCSP) (Brucker et al., 1999). Note that we do not need any new decision variables to incorporate this flexibility.

#### 2.4.1. Fundamental inequalities

We first generalize the scheduling constraints to the case of alternative resources. For each request $i$ and resource $r$, we define the following sets: $L_i = \{r' \in R_i : \exists (r', r, u) \in \mathcal{O}(i)\}$ and $\mathcal{P}_i = \{r' \in R_i : \exists (r', r, u) \in \mathcal{O}(i)\}$. The former is the set of all the alternative resources that can be utilized soon after resource $r$ (i.e., the set of all direct successors of vertex $r$ on graph $G(V', A')$), the latter is the set of alternative resources that can be used before utilizing resource $r$ (i.e., the set of all direct predecessors of vertex $r$ on graph $G(V', A')$).

$$w_{t,i} \leq \sum_{r \in \mathcal{P}_i} w_{t,i} \quad \forall i \in I, t \in R_i, t \in T'_i, |\mathcal{P}_i| \geq 1. \quad (2a)$$

$$w_{t,i} \geq \sum_{r \in \mathcal{L}_i} w_{t,i} \quad \forall i \in I, t \in R_i, |\mathcal{L}_i| \geq 1. \quad (2b)$$

$$\sum_{r \in \mathcal{L}_i} w_{t,i} \leq 1 \quad \forall i \in I, t \in R_i. \quad (2c)$$
Constraints (2a) generalize the precedence constraints. More precisely, they stipulate that a request \( r \) cannot use resource \( \tau \) by time \( t \) if it has not first used one of the resources of set \( \tau \), by time \( t - L_\tau \). Moreover, Constraints (2b) and (2c) state that request \( i \) must use one of the resources in set \( L_i \), if resource \( r \) has been used.

We next present an important class of valid inequalities. These constraints incorporate and generalize valid inequalities introduced to strengthen the formulation of different problems. Referring to the applications cited above, these inequalities include as special cases the “fork” and “joint” inequalities for the air traffic flow management with rerouting (Bertsimas et al., 2011) and a generalization of the Christofides inequalities (Christofides, Alvarez-Valdes, & Tamarit, 1987) to the multimode RCPS.

2.4.2. Valid inequalities

To strengthen the formulation further, we now introduce two classes of valid inequalities for alternative resources. Borrowing concepts and definitions from the theory of partially ordered sets, a subset of the resources \( A \subseteq R_i \) is an antichain if every distinct pair of points from \( A \) is incomparable in \( O(i) \) (set of ordering constraints). Referring to the example depicted in Fig. 1, sets \( \{2,3,4\} \) and \( \{5,6\} \) are two antichains. Analogously, we can define the rank of a vertex as the length of the longest path on \( G(V^i,A_i) \) that has the vertex as the end-point. If \( A_i \) is an antichain for the ordered set defined on \( R_i \), and request \( i \) may not be scheduled (i.e., \( i \in T_i \setminus L_i \)), then the following constraints are valid inequalities (known as antichain inequalities):

\[
\sum_{r \in R_i} w^{i}_{r,T_i} \leq 1. \tag{3a}
\]

Using the rank definition, the set of vertices \( V_i \) can be partitioned into multiple subsets - \( V_0, \ldots, V_k \), where \( V_k \) is the subset of vertices whose rank is \( k \). The partition is both collectively exhaustive \( \bigcup_{k=0}^{k} V_k = V_i \) and mutually exclusive \( V_k \cap V_{k+1} = \emptyset \) \( \forall k, k \in \{0, \ldots, k\} \) with respect to the set being partitioned. \( V_0 \) and \( V_k \) include only the origin and the destination respectively. Note that, in view of this partition described above, each arc \((i,j) \in A\) has its tail vertex in a subset, say \( V_k \), and its head vertex in the subsequent one, i.e., \( V_{k+1} \).

**Definition 2.1.** Given two consecutive sets of vertices of the partition defined above (for request \( i \)), say \( V_k \) and \( V_{k+1} \), a collection of two subsets, \( B_{k}^i \subseteq V_k \) and \( B_{k+1}^i \subseteq V_{k+1} \), forms an “island” if their elements satisfy the following conditions:

\[
\forall \tau \in B_{k}^i \quad \mathcal{L}_i \subseteq B_{k+1}^i, \quad \text{and} \quad \forall \tau \in B_{k+1}^i \quad \mathcal{P}_i \subseteq B_{k}^i. \tag{3b}
\]

The name island follows from observing that all the arcs having tail vertex in \( B_{k}^i \) have the head vertex in \( B_{k+1}^i \). Hence, there is no arc of the digraph whose tail vertex is in set \( B_{k}^i \) and the head vertex is in \( V_{k+1} \setminus B_{k+1}^i \). Every pair of successive sets \((V_k \text{ and } V_{k+1})\) may give rise to multiple islands. This motivates the definition of a parameter \( Q(k) \) as the maximum number of islands possible between these two sets of vertices. Each index \( q \in \{1, \ldots, Q(k)\} \) corresponds to a unique island defined on \( V_k \text{and} V_{k+1} \). For instance, referring to the digraph \( G(V,A) \) depicted in Fig. 1, we have \( L_i = 3 \) and there is only one island each for \( k = 0,1,2 \). These three islands are: \( (B_{0}^i, B_{1}^i) = \{(1), (2,3,4)\} \), \( (B_{1}^i, B_{2}^i) = \{(2,3,4), (5,6)\} \) and \( (B_{2}^i, B_{3}^i) = \{(5,6), (7)\} \). In case arc \( (3,5) \) is removed from the set of arcs, then for \( k = 1 \) the digraph has two islands, \( (4),(5) \) and \( (2,3),(6) \) respectively. For each island \((B_{k}^i, B_{k+1}^i)\), the following condition holds (relative to request \( i \)):

\[
\sum_{r \in B_{k}^i \cap T_i} w^{i}_{r,T_i} + \sum_{r \in B_{k+1}^i \cap T_i} w^{i}_{r,T_i} \leq \sum_{r \in B_{k}^i \setminus L_i} w^{i}_{r,T_i} + \sum_{r \in B_{k+1}^i \setminus T_i} w^{i}_{r,T_i} \quad \forall k = 0, \ldots, L_i - 1, \forall q \in \{1, \ldots, Q(k)\} \tag{3b}
\]

where \( \tau^q_{imin} = \min(T_i : r' \in B_{k+1}^i) \) and \( \tau^q_{imax} = \max(T_i : r' \in B_{k+1}^i, r' \in B_{k+1}^i) \). These conditions, which are valid, state that a request \( i \) will not use any of the resources \( r' \in B_{k+1}^i \) by time \( t \) unless it has used one of the resource \( r \in B_{k}^i \) by time \( t - L_\tau \). Fig. 2 displays an example of the island valid inequality.

Note that, for \( t = \tau^q_{imax} \), the island inequality holds as equality and becomes

\[
\sum_{r \in B_{k}^i \cap T_i} w^{i}_{r,T_i} = \sum_{r \in B_{k+1}^i \cap T_i} w^{i}_{r,T_i} \quad \forall k = 0, \ldots, L_i - 1, \forall q \in \{1, \ldots, Q(k)\}.
\]

**Remark 3.** The number of the antichain inequalities (3a) and of the island constraints (3b) are polynomial in the number of requests and in the total number of resources included in all the sets of alternative resources.

**Remark 4.** In the special case when the sets of alternative resources are used to model the multimode resource-constrained project scheduling problem (mmRCPS), the island constraints (3b) are equivalent to a generalization of the precedence constraints proposed by Christofides et al. (1987) for RCPS.

2.5. Fairness constraints

In this section, we introduce fairness in our framework. Specifically, we incorporate three widely-used fairness paradigms, namely, (i) First-Scheduled First-Served (FSFS) fairness; (ii) proportional fairness; and (iii) min–max fairness.

2.5.1. FSFS fairness

This fairness criterion is based on a control on the ordering of the requests at each resource based on an agreed upon schedule \( a \) priori. This is intuitively appealing because the agreed upon sequence of processing the requests at each resource embodies internally calculated priorities set forth by the individual stakeholders. To include this fairness in our modeling framework, we introduce the notion of set of pairs of requests, \( C \), which are reversible during the use of resource \( r \). A pair of requests \((i,f) \in C \) if both use this resource and the following conditions hold: \( L_i \cap T_f \neq \emptyset \). For each pair of requests \((i,f) \in C \), we count a reversal, if in the resulting solution, request \( f \) begins processing before request \( i \) in resource \( r \). Moreover, the amount by which request \( i \) begins processing before request \( i \) constitutes \( \text{overtaking} \). Consequently, the fairness proposal is to control the number of reversals and amount of overtaking. For \((i,f) \in C \), the following two set of variables are introduced to model overtaking (the index \( j \in \{0, \ldots, |T_f \cap T_i|\} \) captures the feasible time-periods when overtaking can occur) and reversal respectively.

\[
s^i_{l,j,f} = \begin{cases} 1, & \text{if request } i \text{ has advanced request } j \text{ using resource } r \text{ at time } T_{i,j} + j, \\ 0, & \text{otherwise}. \end{cases}
\]

\[
s^i_{l,j,f} = \begin{cases} 1, & \text{if there is a reversal between requests } i \text{ and } f \text{ in using resource } r, \\ 0, & \text{otherwise}. \end{cases}
\]

To model overtaking, we need the following constraints:
\[ w_{t}^{i} \leq w_{t}^{i} + s_{t}^{i} \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}', \quad (4a) \]
\[ w_{t}^{i} \leq w_{t}^{i} + 1 - s_{t}^{i} \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}', \quad (4b) \]
\[ w_{t}^{i} + s_{t}^{i} \leq 1 \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}', \quad (4c) \]
\[ -w_{t}^{i} + s_{t}^{i} \leq 0 \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}'. \quad (4d) \]

If there is a unit of overtaking between requests \( i \) and \( i' \) at time \( t - T_{t}' \) during the use of resource \( r \), i.e., \( s_{t}^{i} - s_{t}^{i'} = 1 \), then Constraint (4a) becomes redundant, Constraint (4c) implies \( w_{t}^{i} = 0 \) and Constraint (4d) implies \( w_{t}^{i'} = 1 \). Moreover, this assignment is satisfied by Constraint (4b). Similarly, if there is no overtaking at time \( t - T_{t}' \), i.e., \( s_{t}^{i} - s_{t}^{i'} = 0 \), then Constraints 4b, 4c and 4d become redundant and Constraint (4a) ensures that request \( i' \) cannot start using resource \( r \) before request \( i \). That is, Constraint (4a) stipulates that if request \( i' \) utilizes resource \( r \) by time \( t \), then also request \( i \) has to use resource \( r \) by that time. These constraints correctly capture the semantics of modeling overtaking.

To model reversal, we need the following constraints:
\[ w_{t}^{i} \leq w_{t}^{i} + s_{t}^{i} \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}', \quad (5a) \]
\[ w_{t}^{i} \leq w_{t}^{i} + 1 - s_{t}^{i} \quad \forall r \in R, (i, t) \in C', t \in T_{t} \cap T_{t}', \quad (5b) \]

If there is a reversal between requests \( i \) and \( i' \) during the use of resource \( r \), i.e., \( s_{t}^{i} = 1 \), then Constraint (5a) becomes redundant while Constraint (5b) stipulates that request \( i' \) cannot start using resource \( r \) before request \( i \), i.e., if request \( i \) starts using resource \( r \) by time \( t \), then request \( i' \) has to use resource \( r \) by that time. Similarly, if there is no reversal, i.e., \( s_{t}^{i} = 0 \), then Constraint (5b) becomes redundant while Constraint (5a) imposes that request \( i' \) cannot start using resource \( r \) before request \( i \). Thus, a reversal is modeled with the addition of only one variable \( s_{t}^{i} \).

Please see the Supplementary material for a detailed derivation of the constraints introduced in this section to model reversals and overtaking.

2.5.2. Proportional fairness

This fairness paradigm is one of the most fundamental fairness schemes widely used in the economics literature. Under this paradigm, resources are allocated to owners commensurate with the total resources demanded by each one of them. Let \( \eta_{o} \) denote the welfare or utility derived by owner \( o \) per request (which is a function of the decision variables) and \( \gamma \) be the mean of these \( \eta_{o} \)'s, i.e.,
\[ \gamma = \frac{\sum_{o} \eta_{o}}{|O|}. \]

Then, proportional fairness can be enforced in our framework by introducing an additional term \( \xi \cdot |\eta_{o} - \gamma| \) in the objective function where \( \xi \) is the tradeoff parameter. We can then linearize the objective by replacing \( |\eta_{o} - \gamma| \) by \( \xi \) and introducing constraints:
\[ \zeta \geq \eta_{o} - \gamma, \forall o \in O. \quad (6a) \]
\[ \zeta \geq \gamma - \eta_{o}, \forall o \in O. \quad (6b) \]

2.5.3. Min–max fairness

The final fairness paradigm we incorporate is that of min–max fairness which is widely used in telecommunications and networks (e.g., bandwidth allocation and packet routing). In fact, in this paper, we report results for JSP with the makespan objective which corresponds to min–max fairness. The max term in the objective function \( \max_{o \in O} \eta_{o} \) can be simply replaced by a variable \( z \) and the following constraints need to be introduced:
\[ z \geq \eta_{o}, \forall o \in O. \quad (7) \]

2.6. The complete modeling framework

We now have all the ingredients to summarize our complete model for DRAP:
\[ \min_{w, s} F(w, x, s) \]
\[ \text{s.t.} \ (1a)-(1f), \quad (2a)-(2c), \quad (3a)-(3b), \quad (4a)-(4d), \quad (5a)-(5b), \quad (6a) - (6b), \quad (7), \] \[ w \in \{0, 1\}^{n}, \quad s \in \{0, 1\}^{n}{ }^{ \dagger}. \]

The objective function \( F \) is flexible enough to accommodate any linear combination of scheduling variables \( w, x \) and the fairness variables \( s \). This flexibility allows the objective function to be tailored to multiple problem domains. Table 1 presents various possible objective functions and the corresponding applications. For instance, the first objective minimizes a weighted combination of total delay and reversals. The decision-maker can choose the tradeoff parameter \( \xi \) to adjust the degree of controlling reversals. The second one corresponds to makespan, which entails minimizing the delay of the maximum delay request (this indirectly embodies a fairness notion emanating from the min–max fairness paradigm). Finally, the third objective minimizes the total priority of unscheduled requests. As such, it aims to give scheduling precedence to the requests with high priority.

2.7. An iterative application of our framework

Although our modeling framework as a standalone proposal is able to solve many problems tractably (as evidenced by encouraging computational results in Section 5), as also mentioned in van den Akker, Hurkens, and Savelbergh (2000) and Khe et al. (2009), the proposed time-index formulation may have one major disadvantage, i.e., the size. Indeed, under our framework, the time window within which each request can be processed at a resource governs the size of the formulation. This, in turn, dictates the computational efficiency of the model. In this section, we propose an iterative procedure which calls the framework multiple times to further improve upon the efficiency and/or optimality of our framework. The iterative procedure we design builds upon the following two key ideas:

1. Scaling. By scaling the durations that requests need to use resources, we can reduce the size of the formulations being solved. Furthermore, a feasible solution in the scaled space can be mapped back to a feasible solution in the original search space.
2. Adjutable time windows. By adjusting the windows within which requests can be processed in a resource, the search space over which the optimization is to be performed can be controlled. This flexibility is particularly attractive when natural bottlenecks have been identified and emphasis needs to be given on solutions in their neighborhoods.

The algorithm invokes the model in Section 2.6 multiple times. At each iteration \( k \), it solves a new instance of the problem obtained by:
\[ (a) \] scaling the duration \( l_{o} \) that request \( i \) requires resource \( r \) by the parameter \( L_{o} \), that is the new durations become \( |l_{o}/L_{o}| \), and
During which request can be processed by resource \( r \), by the parameter \( D_{ik}^r \).

The new time windows for iteration \( k \) are computed as follows: let \( t'_i \) be the starting time that request \( i \) is processed by resource \( r \) in the optimal solution of the \((k-1)\)th iteration and let \( t_i = \lceil \frac{t'_i}{D_{ik}^r} \rceil \). Then the new window is \( \{ t_i - D_{ik}^r, \ldots, t_i + D_{ik}^r \} \). The scaling parameter \( L_k \) controls the size of the formulation, while \( D_{ik}^r \) modulates the search space on which to optimize the new instance of the problem.

This sequence of steps is repeated until one of the following stopping criteria is met: (i) the improvement in the objective function is smaller than a given threshold; (ii) the number of iterations have reached an upper limit specified a priori; and (iii) a time limit has been reached.

### 3. Example applications

In this section, we first highlight the generality of our framework and then present formulations of the three applications mentioned in the Introduction as special cases of DRAP.

#### 3.1. Generality of the framework

Table 2 enumerates numerous applications which can be modeled within our framework. The range of problem domains presented emphasizes its broad applicability. A brief description of the problems is as follows:

- **Bandwidth Allocation.** In this problem, computer systems (PCs) request bandwidth from a pool of available servers, each with its own capacity. These requests must be scheduled, preferably to minimize the overall wait time, but especially to minimize the wait time for the highest priority requests. Finally, in the presence of competing players, proportional fairness might need to be enforced to ensure overall equity.

- **Internet Packet Routing.** Internet packets need to be routed from origin to destination servers by making use of intermediate routers. The critical requirements for this problem is overcoming local congestion, satisfying precedence constraints at the destination servers and meeting the scheduling requirement of the critical packets.

- **Pipeline Scheduling.** The scheduling problem within the pipeline industry involves allocating different types of pipes (in terms of length, size, material, etc.) in the pipeline network (resources) to individual products (requests) demanding specific types for varying lengths of time.

- **Aircraft Maintenance.** As mentioned in Section 1, ATFM refers to the composite of services that attempt to prevent local demand-capacity imbalances by adjusting the flow of aircraft on a national or regional basis. ATFM models optimize for each flight the time of departure, the route selected, the time required to traverse each sector, and the time of arrival at the destination airport, taking into account the capacity of all the elements of the air traffic management system.

Under our modeling framework, requests in the ATFM problem may fly one of many possible routes, whose order in the request is not specified a priori. The capacity \( L_k \) can be processed by resource \( r \), by the parameter \( D_{ik}^r \).

### 3.2. The air traffic flow management problem

As mentioned in Section 1, ATFM refers to the composite of services that attempt to prevent local demand-capacity imbalances by adjusting the flow of aircraft on a national or regional basis. ATFM models optimize for each flight the time of departure, the route selected, the time required to traverse each sector, and the time of arrival at the destination airport, taking into account the capacity of all the elements of the air traffic management system.

Under our modeling framework, requests in the ATFM problem are aircraft that have to fly from its airport of origin to the airport of destination. The resources are the airports as well as the enroute airspace sectors. The complete execution of a flight (request) requires the utilization of a sequence of resources in set \( R_i \) – origin–destination route flown by aircraft \( i \); whose order in the resources’ utilization is enforced by a set of ordering constraints \( O(i) \). Since each aircraft \( i \) may fly one of many possible routes, the set of alternative resources \( R_i \) is also specified. The capacity of a resource \( b_{ij} \) is the number of aircraft which are allowed to use the resource in the specified period of time. For airports, this

#### Table 1

**Objective** (min) \( F(w, x, s) \) Applications

<table>
<thead>
<tr>
<th>Objective</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize Delays Control Reversals</td>
<td>Air Traffic Scheduling</td>
</tr>
<tr>
<td>Minimize Makespan</td>
<td>Job Shop Scheduling</td>
</tr>
<tr>
<td>Minimize Total Priority of Unscheduled Requests</td>
<td>Aircraft Maintenance</td>
</tr>
</tbody>
</table>

#### Table 2

**Examples of the applications that can be modeled within our framework.**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering</td>
<td>(a) Bandwidth allocation</td>
</tr>
<tr>
<td></td>
<td>(b) Internet packet routing</td>
</tr>
<tr>
<td></td>
<td>(c) Job shop scheduling</td>
</tr>
<tr>
<td>Services</td>
<td>(a) Pipeline scheduling</td>
</tr>
<tr>
<td></td>
<td>(b) Hospital scheduling</td>
</tr>
<tr>
<td></td>
<td>(c) Aircraft maintenance</td>
</tr>
<tr>
<td>Transportation</td>
<td>(a) Air traffic management</td>
</tr>
<tr>
<td></td>
<td>(b) Shipping scheduling</td>
</tr>
</tbody>
</table>

- **Hospital Scheduling.** A hospital has multiple classes of resources like nurses and doctors, operating rooms, and specialized equipment. There are capacity limits for each human resource governed by the hospital’s overall staffing, including available temporary employees and weekly work hour limitations. The scheduling problem is then to allocate the resources to various hospital shifts.

- **Shipping.** The scheduling problem in the shipping industry pertains to the allocation of ships (which can be of multiple types based on size, transportable goods, etc.) between different origin–destination pairs in order to satisfy cargo demand and supply requirements at various nodes.
is the number of aircraft that are allowed to land and to take off, while for en-route sectors, capacity is the number of aircraft that an air traffic controller can oversee simultaneously and may vary with the geographic location of the sector, its geometric configuration and the weather conditions. The set \( E(r) \), set of transition times for resource \( r \), allows to capture the connectivity between flights at airports (resource \( r \)). They handle the cases in which a flight is continued, i.e., the flight’s aircraft is scheduled to perform a subsequent flight within some user-specified time interval \( e_{fi} \).

3.2.1. The decision variables

\[
w_{r,t} = \begin{cases} 1 & \text{if flight } i \text{ arrives at sector } r \text{ by time } t, \\ 0 & \text{otherwise}. \end{cases}
\]

\[
s_{r,t} = \begin{cases} 1 & \text{if there is a reversal between flights } i \text{ and } i' \text{ in using airport } r, \\ 0 & \text{otherwise}. \end{cases}
\]

3.2.2. Constraints

\[
\min_w s \sum_{r \in R} \left( \sum_{t \in T} t \left( w_{r,t} - w_{r,t-1} \right) \right) + \lambda \cdot \left( \sum_{r \in R} \sum_{i \in C} s_{r,t} \right)
\]

s.t.

- airport’s, sectors’ capacity (1g),
- time connectivity (1b),
- flights’ connectivity (1e),
- routing constraints (2a)–(2c),
- (3a)–(3b),
- flights’ overtaking (4a)–(4d),
- flights’ reversal (5a)–(5b),
- proportional fairness (6a)–(6b),
- \( w \in \{0, 1\}^n \), \( s \in \{0, 1\}^n \).

In case, the rerouting option is not included as a control option in the mathematical model, each aircraft may only fly a pre-assigned route. Consequently, the routing constraints (2a)–(2c) can be substituted by (1c) and (1d). Under this assumption, there is only one ordering condition \((r, r', u)\) for each resource \( r \in R_i \), i.e., there is only one resource \( r' \) that has to be used after resource \( r \). In this case, the set of resources \( R_i \) can be ordered, such that, \( r + 1 = r' \). The capacity constraints (1g) can be specialized as follows:

\[
\sum_{t \in T, r \in R, c \in C} (w_{r,t} - w_{r,t-1}) \leq b_n, \quad \forall r \in R, t \in T.
\]

3.3. The aircraft maintenance problem

In the aircraft maintenance problem, we are concerned with hangar maintenance which is performed on the aircraft after a certain number of flight hours (that depends on the aircraft type). Hangar maintenance may involve the check of all the individual parts of an aircraft which can take up to several months. It includes a large number of inspection and repair activities. For instance, disassembly, cleaning, and inspection of dismounted components, replacement work and testing operations (see Gharbi et al. (1997)).

Maintenance activities require particular tools and equipments – e.g., X-ray, magnetic particle and ultrasonic facilities, which are used to detect corrosion, cracks, and other structural abnormalities on the aircraft surfaces (see Aircraft inspection, xxxx) – and highly specialized skills, e.g., mechanical, electrical, or radio-skilled engineers. Moreover, maintenance personnel are licensed to operate on limited types of aircraft (see Dijkstra et al. (1994)).

A request for this problem corresponds to an aircraft undergoing a maintenance program. As described above, the aircraft maintenance program is composed of several activities each requiring specific resources in terms of workers, tools and equipments. The set of all the equipments that can be used on aircraft \( i \) and of all the workers that can operate on aircraft \( i \) form the set of resources \( R_i \). For complete execution of an activity, more than one resource of the set \( R_i \) may be required. The duration of an activity \( l_i \), depends on the expertise of the worker allocated to the activity (Gharbi et al. (1997)) and/or on the attributes of the equipment used. Therefore, alternative resources represent workers with different expertise and/or equipments with different attributes.

Maintenance activities have to satisfy logical precedence conditions. These conditions translate in a set of ordering constraints \( O(i) \) for each task \( i \). The capacity of a resource \( b_{rt} \) is the amount of resource available at time period \( t \), i.e., the number of workers with specific skills and licenses – e.g., electrical engineers with the Boeing 747 license, or the number of specific equipments – e.g., X-ray machines. Typically, \( E(r) = \emptyset \) for ACM problem as the resources which are maintenance personnel and equipment can be used continuously without the need to spend time in transitioning between consecutive requests. Thus, the request connectivity constraints (1c) are not needed.

3.3.1. The decision variables

\[
w_{r,t} = \begin{cases} 1 & \text{aircraft } i \text{ has begun utilizing resource } r \in R_i \text{ by discrete time } t \in T', \\ 0 & \text{otherwise}. \end{cases}
\]

3.3.2. Constraints

\[
\min_w \sum_{r \in R} \sum_{t \in T} t \left( w_{r,t} - w_{r,t-1} \right)
\]

s.t.

- resources’ capacity (1a),
- time connectivity (1b),
- resource ordering (1c) and (1d),
- alternative resources (2a)–(2c),
- \( w \in \{0, 1\}^n \).

3.4. The Job Shop Scheduling problem

Job Shop Scheduling is one of the most famous combinatorial optimization problems. The computational intractability of this problem is evident from the fact that it is NP-Hard and a relatively small instance of the problem, namely a 10 job by 10 machine instance proposed by Giffler and Thompson (1960) remained unsolved for decades. This problem (FT10 henceforth) was solved to optimality for the first time by Carlier and Pinson (1989) in 1989 using a branch and bound type algorithm. Subsequently, numerous other proposals solved the FT10 problem to optimality (e.g., Adams et al. (1988); Applegate and Cook (1991)).

The problem entails processing a set of jobs (each comprising of a set of sequential tasks) on machines with the objective of minimizing some function of the completion times of the jobs. Under our modeling framework, the requests are the set of tasks and resources correspond to the set of machines. Each task \( i \) comes pre-assigned with the machine \( r(i) \) where it needs to be processed. Finally, the multiple tasks comprising each job is modeled by the set of ordering constraints \( O(i) \). For JSP, \( E(r) = \emptyset \) as each task only gets processed by one machine and thus, the request connectivity constraints (1e) are not needed. To enable the precise definition of
minsum in the objective below, we denote by \( \mathcal{I}_f \subset \mathcal{I} \), the set comprising of last task of each job.

### 3.4.1. The decision variables

\[
w_{r,t}^{i,j} = \begin{cases} 
1, & \text{if task } i \text{ has started processing on machine } r(i) \\
0, & \text{otherwise.}
\end{cases}
\]

### 3.4.2. Constraints

\[
\min \max \left\{ \sum_{r(t,r) \neq 0} t \left( w_{r,t}^{i,i} - w_{r,t}^{i,j} \right) \right\} \quad \text{[Makespan] OR}
\]

\[
\sum_{r(t,r) \neq 0} t \left( w_{r,t}^{i,i} - w_{r,t}^{i,j} \right) \quad \text{[Minsum]}
\]

s.t. machines’ capacity (1a),

\[
\text{time connectivity (1b),}
\]

\[
\text{tasks’ precedence (1c),}
\]

\[
w \in \{0,1\}^n.
\]

In the above formulation, \( f^a \in \mathbb{R}, \forall i \in \mathcal{I}, r \in \mathcal{R} \) and \( b^a \in \mathbb{R}, \forall r \in \mathcal{R}, t \in \mathcal{T} \) in Constraints (1a) as each task utilizes one unit of a machine in every time-period and each machine can process at most one task at any time. The parameter \( u \) for every \( (i, r, u) \) in Constraints (1c) is the processing time of task \( i \).

Finally, we remark that the critical requirement of non-preemption (i.e., once a task begins processing, it needs to be completed before the machine can start processing any other task) is naturally enforced in the above formulation. Please refer to Supplementary material for the proof.

### 4. Insights from the polyhedral structure

In this section, we present theoretical evidence on the strength of the constraints for scheduling, alternative resource and fairness which illuminate the reasons for the computational effectiveness of our framework.

#### 4.1. Strength of the scheduling constraints

For each element \( i \in \mathcal{I} \), let \( P(i) \) denote the polyhedron induced by Constraints (1b)–(1d):

\[
P(i) = \left\{ w_{r,t}^{i,j} \in [0,1] \mid \forall r \in \mathcal{R}, \forall t \in \mathcal{T} : \right. \\
\left. w_{r,t}^{i,j} - w_{r,t+1}^{i,j} \leq 0, \forall r \in \mathcal{R}, \forall t \in \{T_1, \ldots, T_i-1\}, \right. \\
\left. w_{r,t}^{i,j} - w_{r,t-u}^{i,j} \leq 0, \forall (r', r, u) \in \mathcal{Q}(i), \forall t \in \mathcal{T}_r : t - u \in \mathcal{T}, \right. \\
\left. w_{r,t-u}^{i,j} - w_{r,t}^{i,j} \leq 0, \forall (r', r, u) \in \mathcal{B}(i), \forall t \in \mathcal{T}_r : t - u \in \mathcal{T}_r \right\}
\]

**Theorem 1.** The polyhedron \( P(i) \) is integral.

**Proof.** Please see Supplementary material.

#### 4.2. Strength of the alternative resource constraints

Let \( \mathcal{P}_{\text{ORA}} = \{ w \in [0,1]^n, s \in [0,1]^n : (1a)-(1f), (2a)-(2c), (3a)-(3b), (4a)-(4d), (5a)-(5b) \} \)

**Theorem 2.** If \( \mathcal{A} \) is an antichain for the ordered set defined on \( \mathcal{R} \) and request \( i \) may not be scheduled (i.e., \( i \in \mathcal{I} \setminus \mathcal{A} \)), then constraint

\[
\sum_{r \in A} w_{r,T}^i \leq 1,
\]

is facet-defining for \( \mathcal{P}_{\text{ORA}} \).

**Proof.** Please see Supplementary material.

In the following theorem, \( \zeta \) denotes the minimum cover of \( \mathcal{B}^*_\mathcal{I} \), i.e., the minimum cardinality subset of \( \mathcal{B}^*_\mathcal{I} \) which covers the vertices in \( \mathcal{B}^*_\mathcal{I} \), with arcs representing the inclusion of elements in sets.

**Theorem 3.** The island constraints \( (3b) \) are \( K \) face defining inequalities for \( \mathcal{P}_{\text{ORA}} \) with \( K = \dim(\mathcal{P}_{\text{ORA}} - \zeta) \).

**Proof.** Please see Supplementary material.

**Remark 5.** Whenever \( \zeta = 1 \), the island constraints \( (3b) \) are facet-defining for \( \mathcal{P}_{\text{ORA}} \).

**Remark 6.** The value of \( \zeta \) depends on the topology of the island. For instance, Fig. 3 depicts two examples of island with same cardinality of the sets \( \mathcal{B}^*_\mathcal{I}, \mathcal{B}^*_\mathcal{I} \) (\( \{\mathcal{B}^*_\mathcal{I} = \mathcal{B}^*_\mathcal{I} \} \) = 3) but different topologies. For the island on the left-hand side, the inequalities \( (3b) \) are facet-defining (\( \zeta = 1 \)) whilst for the island on right-hand side, they are not (\( \zeta = 2 \)).

**Remark 7.** If either \( \mathcal{B}^*_\mathcal{I}, \mathcal{B}^*_\mathcal{I} \) or both are singletons, then constraints \( (3b) \) are facet-defining for \( \mathcal{P}_{\text{ORA}} \).

#### 4.3. Strength of the fairness constraints

For each resource \( r \in \mathcal{R} \) and pair \( (i, i') \in \mathcal{C} \), let \( P_r(i, i', r) \) be the polyhedron induced by Constraints (5a) and (5b):

\[
P_r(i, i', r) = \left\{ w_{r,t}^{i,i'} \in [0,1], s_{i,i',r} \in [0,1] : \right. \\
\left. w_{r,t}^{i,i'} - w_{r,t}^{i,i'} \leq 0, \forall t \in \mathcal{T}_r : t \leq T_r, \right. \\
\left. w_{r,t}^{i,i'} - w_{r,t}^{i,i'} + s_{i,i',r} \leq 1, \forall t \in \mathcal{T}_r : t \leq T_r \right\}
\]

**Theorem 4.** The polyhedron \( P_r(i, i', r) \) is integral.

**Proof.** Please see Supplementary material.

#### 5. Empirical evidence

In this section, we present numerical results on the three applications which were modeled within our framework (in Section 3) to illustrate its computational effectiveness. The optimization
models for all the three problems are solved using the Gurobi branch-and-bound method 4.5.2, implemented using AMPL as the modeling language, on a PC with AMD-Xeon 4 processors, 3 gigahertz, 8 gigabytes RAM with Linux Ubuntu 4.03 OS. We terminate the models until we find a solution with optimality gap of 1% up to a time limit of 3600 seconds.

5.1. Air traffic flow management

We present numerical results on the ATFM problem modeled within our framework. We evaluate the computational performance of the mathematical model on real-world flights data. In our experimental setup, the airspace is subdivided into sectors of equal dimensions (10 by 10) that form a grid, thereby, having a total of 100 sectors. The 55 major airports of the US are then mapped to one of these 100 sectors based on its geographical coordinates. Finally, the flight operations in these datasets correspond to the 5 largest US carriers. Please see Bertsimas and Gupta (submitted for publication) for details on the ATFM problem with fairness controls and Bertsimas et al. (2011) for ATFM problem with the flexibility of alternative resources.

We report numerical results for this problem under both fairness paradigms of FSFS and proportional fairness using our framework. In the US, “Ration-by-Schedule” (RBS) is the fundamental principle for allocation of arrival slots. Under this paradigm - arrival slots at airports are assigned to flights in accordance with the published sequence. But, under capacity reductions at the airports, a feasible solution under RBS might not even be possible. A close equivalent to the RBS solution would be one which has a small amount of overtaking or reversals, i.e., one which follows the FSFS priority discipline. Consequently, FSFS fairness paradigm is particularly appealing in the ATFM context.

The instances reported here have a typical size of the order of 300,000 variables and 800,000 constraints. Table 3 reports the numerical results from our framework under the FSFS paradigm. The objective function minimizes a weighted combination of total delay and reversals (please see Table 1) with the tradeoff parameter \( \gamma \) set to 100. For each one of the datasets, we report the number of reversals and overtaking when fairness is imposed at \( k \) airports (\( k \) is progressively increased from 0). \( k = 0 \) corresponds to the case when no fairness is imposed at any airport. Reversals are potentially controllable up to single digits which is quite encouraging given that the actual flight sequences had thousands of reversals (around 7000 on average). A similar control is possible on overtaking. This obviously comes at a increase in delay cost which is reported in the last two columns. The running times on average are less than an hour which are encouraging for real-time deployment in the ATFM context. The optimality gap is less than 1% in all cases except for one when a time limit of 3600 seconds is attained (the gap in this instance was 4.1%). Finally, one of the most striking observations is that the solver did not branch at all for all but one instance, which highlights the strength of the framework. In the one case when the algorithm requires branching, only 3 branch-and-bound nodes are explored during its execution. These results are quite promising as the framework on its own is able to solve the problem (involving hundreds of thousands of variables and constraints) to optimality (without the need to invoke the iterative procedure).

Fig. 4 plots the box plot for the computational times of the ATFM model for FSFS and proportional fairness. The framework is able to solve the ATFM problem under both fairness paradigms under less than an hour (with an average less than 30 minutes).

5.2. Aircraft maintenance problem

In this section, we evaluate the computational performance of the model introduced in Section 3.3 on randomly generated instances. The size of each instance depends on the number of aircraft requiring maintenance, the number of activities to be executed, the width of the time windows within which to execute each activity and the number of resources that can be utilized for the execution of each activity. We consider instances whose number of aircraft ranged between two and four, with either 30, 60 or 90 activities to be executed for the maintenance of each aircraft, and different level of resources. In our analysis, resources are teams of workers that can handle some specific activities. Therefore, the capacity of each resource is set to one for each time period.

Table 4 reports the computational results of the framework on four aircraft and 60-activity instances. For each group of instances, identified by the level of resources considered (first column of Table 4), the following statistics are reported: average computation time to solve the ten instances forming each group, the optimality gap of the solution and the number of branch-and-bound nodes visited. For the last two statistics, we report both the average value and the third quartile of the distribution.

Our framework solved to (near) optimality most of the instances, as indicated by the small values of the third quartile of the optimality gap’s distribution. Only few instances, more precisely one for each group with 30, 27 and 21 resources, were solved with an optimality gap greater than 10%. Finally, the small number of branch-and-bound nodes visited to solve the instances – indeed, for most of the instances the solver did not branch at all – validates the strength of the underlying polyhedral structure.

We next evaluate the tradeoffs between computational time and solution quality of the iterative procedure (of Section 2.6) ap-

---

### Table 3

Computational performance of our framework under the FSFS paradigm. Note that \( k \) denotes the number of airports where fairness is imposed. \( \mathcal{RV} \) denotes the number of reversals and \( \mathcal{OV} \) denotes the amount of overtaking.

<table>
<thead>
<tr>
<th>Dataset (# of flights)</th>
<th>( k )</th>
<th>Solution time (s)</th>
<th>( \mathcal{RV} )</th>
<th>( \mathcal{OV} )</th>
<th>Delay cost (15 min)</th>
<th>% Increase in delay cost over ( k = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (5092)</td>
<td>0</td>
<td>261</td>
<td>915</td>
<td>1492</td>
<td>3525</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>228</td>
<td>2</td>
<td>4</td>
<td>3691</td>
<td>4.70</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>819</td>
<td>13</td>
<td>30</td>
<td>4246</td>
<td>20.45</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>3600</td>
<td>25</td>
<td>53</td>
<td>4402</td>
<td>24.875</td>
</tr>
<tr>
<td>2 (5844)</td>
<td>0</td>
<td>108</td>
<td>924</td>
<td>1426</td>
<td>3604</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>213</td>
<td>1</td>
<td>2</td>
<td>3785</td>
<td>5.02</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>837</td>
<td>6</td>
<td>9</td>
<td>4028</td>
<td>11.76</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>1024</td>
<td>9</td>
<td>12</td>
<td>4077</td>
<td>13.12</td>
</tr>
<tr>
<td>3 (5780)</td>
<td>0</td>
<td>311</td>
<td>753</td>
<td>1191</td>
<td>2313</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>219</td>
<td>3</td>
<td>6</td>
<td>2406</td>
<td>4.02</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>282</td>
<td>8</td>
<td>19</td>
<td>2383</td>
<td>11.67</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>384</td>
<td>11</td>
<td>24</td>
<td>2648</td>
<td>14.48</td>
</tr>
</tbody>
</table>
plied as follows: we first scale durations using $L_1 = 2$ (and also $L_1 = 4$ for comparison) and leave the time windows unchanged. In the second iteration, we then scale using $L_2 = 1$ and $D_i^2 = 10$ for all requests $i$.

In Table 5, we compare the performance of the standalone framework and the iterative procedure we described above using $L_1 = 2$ and $L_1 = 4$. We find that the iterative framework improves on the running times (sometimes substantially) with a small reduction in the quality of solutions.

Fig. 5 quantifies the performance of the iterative procedure relative to the framework for the instances reported in Table 5. As expected, there is an intrinsic tradeoff between efficiency and optimality. By increasing the scaling parameter, there is a progressive improvement in the running times at a cost of increase in the objective function. The striking observation is that for $L_1 = 2$, the cost is quite close to the optimal cost obtained from the framework (less than 1% on average) with a significant improvement in the running times (45% reduction on average). Similarly, for $L_1 = 4$, there is another level of improvement in running times (90% reduction on average). But, this comes at a relatively higher increase in the cost from the optimal (around 10%).

5.3. Job shop scheduling

In this section, we present the computational experience on JSP. We report the performance of our framework on the widely-studied FT10 instance (Giffler & Thompson, 1960) for both objectives of minsum and makespan. We remark that for the minsum (average completion time) objective, we are not aware of any local-search based procedure for JSP. We have found that a single application of our framework was not sufficient to find high quality solutions, so we report the performance of our framework applied in an iterative fashion.

The specific implementation of the iterative framework of Section 2.6 for both minsum and makespan objectives for the FT10 instance is as follows: in the first iteration, $L_1 = 25$ and $D_i^1 = 6$ for all tasks $i$. In subsequent iterations, the scaling parameter $L_{k+1}$ and neighborhood $D_{k+1}^i$ are chosen as follows:

In Table 5, we compare the performance of the framework and iterative procedure for ACM. Each instance type is described by number of aircraft $\times$ number of activity $\times$ number of resources.

### Table 4

<table>
<thead>
<tr>
<th>Resources</th>
<th>Time (in s)</th>
<th>Gap</th>
<th>B&amp;B nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>$Q_3$</td>
<td>Avg.</td>
</tr>
<tr>
<td>30</td>
<td>2190</td>
<td>1.83</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>2619</td>
<td>2.14</td>
<td>1.32</td>
</tr>
<tr>
<td>24</td>
<td>2516</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>2647</td>
<td>7.67</td>
<td>2.61</td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>Instances</th>
<th>Framework</th>
<th>Iterative procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Objective cost</td>
<td>Time (s)</td>
</tr>
<tr>
<td>$(4 \times 30 \times 18)$</td>
<td>243.8</td>
<td>421</td>
</tr>
<tr>
<td>$(4 \times 60 \times 36)$</td>
<td>418.2</td>
<td>3079</td>
</tr>
<tr>
<td>$(4 \times 90 \times 18)$</td>
<td>754.5</td>
<td>558</td>
</tr>
<tr>
<td>$(4 \times 90 \times 36)$</td>
<td>578.3</td>
<td>4095</td>
</tr>
</tbody>
</table>

Fig. 4. Box plot of the running times for the ATFM problem. For FSFS fairness, the results are reported as a function of the number of airports ($k = 5, 15, 25$) where fairness is imposed.

Fig. 5. Box plots contrasting the performance of Iterative Procedure relative to Framework (i) objective cost (Left); and (ii) running times (Right).
\[ L_{k+1} = \begin{cases} L_k - U_k, & \text{if } \frac{f_k}{L_k} \leq 0.01, \\ L_k, & \text{otherwise}. \end{cases} \]

where \( U_k \sim U\{1, \ldots, 5\} \) is a discrete uniform distribution taking values 1, \ldots, 5 with equal probability, and \( f_k \) denotes the objective function cost in iteration \( k \).

\[ D_{L}^{k+1} = \begin{cases} 10, & \text{for the last tasks of jobs corresponding to the three highest completion times,} \\ 6, & \text{otherwise}. \end{cases} \]

The average and best values reported across multiple runs (in Table 6) correspond to \( L_1 \) being set to either 15, 20 or 25; \( U_k \) chosen from the distributions: \( U\{1, \ldots, n\} \), for \( n = 2, 3, 4 \text{ or } 5 \); and different starting feasible schedules. The stopping criteria is a maximum limit of 60 on the number of iterations.

Table 6 reports the best (as well as average over multiple runs) cost found for the minsum and makespan objectives from our iterative procedure and the best known bounds in the literature. Fig. 6 plots the progress of the iterative procedure (as a function of the iteration number) for the best cost obtained for both minsum and makespan objectives. There are significant gains in the objective cost after first few iterations (around five), but the objective still continuously improves in the many subsequent iterations. This highlights the utility of the iterative procedure, and implies that a single iteration (standalone framework) is not enough for this difficult instance.

The number of branch-and-bound nodes visited to solve the instances are mostly less than double-digit. In fact, the solver did not branch at all for the majority of the runs. This shows the strength of the underlying polyhedral structure.

We now study the sensitivity of the best objective cost to the starting parameter \( L_1 \). A smaller value of this parameter gives an advantage of working with more exact combinatorial problem at hand (e.g. \( L_1 = 1 \) gives the exact original instance).

![Fig. 6. Progress of iterative procedure for (i) best minsum (Left); and (ii) best makespan (Right).](image)

![Fig. 7. Sensitivity of iterative procedure to the starting parameter \( L_1 \) for (i) minsum (Left); and (ii) makespan (Right).](image)
The downside is that a small neighborhood parameter would span a relatively smaller feasible region (because the overall search space is large). In contrast, a larger value of $L_1$ leads to a much smaller overall search space, but the combinatorial details are increasingly lost (e.g., $L_1 = \infty$ would make all durations 1 for any starting instance). Thus, it is expected at the outset that intermediate values of $L_1$ would lead to the best performance. This intuition is validated in Fig. 7 (which plots the best minsum and makespan obtained across multiple runs which are randomized in the same way as described earlier) as the best minsum corresponds to $L_1 = 20$ and the best makespan corresponds to $L_1 = 25$ (note that for the FT10 instance, $L_1$ can lie between 1 and 99).

5.4. Summary of computations

We now summarize the insights gained from the computational results. The strength of the framework is evident from the fact that for all the three problems, it spends little time in branch and bound. This is an important factor that corroborates the tractability of the framework. The assertion is further validated from the fact that for all three applications, the framework takes less than an hour. In fact, for JSP, the computational times are less than 10 minutes in all cases. For ACM, they are less than 30 minutes in all cases. The utility of pursuing an iterative procedure is evident from the following two observations:

1. For JSP, the standalone framework remains far off from optimality. Thus, for JSP, the iterative algorithm is drastically better in finding higher quality schedules (at a relatively small increase in computational burden).
2. In contrast, for ACM, the iterative algorithm is still able to find high-quality solutions but with much shorter running times.

6. Conclusions

In this paper, we have presented a binary optimization framework for modeling a broad class of dynamic resource allocation problems. The key advantages of our framework are (a) modeling flexibility; (b) wide applicability; and (c) tractability. Furthermore, we developed an optimization-based iterative procedure to further improve upon its efficiency and/or optimality.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.ejor.2013.10.063.

References