Computers & Operations Research 51 (2014) 268–281

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Optimal interval scheduling with a resource constraint

Enrico Angelelli *, Nicola Bianchessi, Carlo Filippi

Department of Economics and Management, University of Brescia, Contrada S. Chiara 50, 25122 Brescia, Italy

ARTICLE INFO

Available online 23 June 2014

Keywords:
Scheduling
Fixed job scheduling
Resource allocation
Complexity
Branch and price
Heuristics

ABSTRACT

We consider a scheduling problem where n jobs have to be carried out by m parallel identical machines. The attributes of a job j are a fixed start time s_j, a fixed finish time f_j, a resource requirement r_j, and a value v_j. Every machine owns R units of a renewable resource necessary to carry out jobs. A machine can process more than one job at a time, provided the resource consumption does not exceed R. The jobs must be processed in a non-preemptive way. Within this setting, we ask for a subset of jobs that can be feasibly scheduled with the maximum total value. For this strongly NP-hard problem, we first discuss an approximation result. Then, we propose a column generation scheme for the exact solution. Finally, we suggest some greedy heuristics and a restricted enumeration heuristic. All proposed algorithms are implemented and tested on a large set of randomly generated instances. It turns out that the column generation technique clearly outperforms the direct resolution of a natural compact formulation; the greedy algorithms produce good quality solutions in negligible time, whereas the restricted enumeration averages the performance of the greedy methods and the exact technique.

1. Introduction

We consider a set of n jobs and a set of m parallel machines. Every machine owns R units of a renewable resource, necessary to carry out jobs. The attributes of a job j are a fixed start time s_j and a fixed finish time f_j ($j=1,\ldots,n$). Furthermore, processing job j requires r_j units of resource available on the assigned machine and yields a value v_j. Every machine can process more than one job at a time provided the resource consumption does not exceed R. The jobs must be processed in a non-preemptive way. All data are positive integers.

In such an environment, we may be interested in at least three types of problems:

- Does a feasible schedule exist for all jobs?
- Which is a subset of jobs that can be scheduled with the maximum total value?
- Which is the minimum number of machines required to schedule all jobs?

These problems are abstract versions of some challenges that may arise in different real world situations, exemplified in the following:

- Consider a set of aircraft to be parked in an airport for land side operations. Usually, the parking must take place during a fixed interval of time, from the arrival of a flight to the departure of the next one carried by the same aircraft. Sometimes, the parking space layout is such that a same parking lot may be occupied by either one large aircraft, or two (or more) smaller ones, or even different combinations (a situation like this happens, for instance, at Milano Malpensa airport, Italy). We may look at aircraft as jobs and parking lots as machines, where each parking lot is formed by a given number of parking places that can operate independently or not. We may ask whether it is possible to schedule all planned flights and, if not, which ones to refuse.
- Consider a constellation of satellites equipped with instruments for optical, radar or infra-red observation, whose mission is to acquire images of specific areas of the Earth surface, in response to customers' observation requests. Each image sent to a customer generates a reward. In the design phase, simulated work plans may be used to find the best hardware setting of each satellite. According to the simulated work plan, image data files are acquired by the satellite at planned acquisition times and transmitted to a ground station at planned transmission times. Meanwhile, the files must be stored on given memory devices, working in parallel. For convenience, files cannot be split among different memory devices. We may look at files as jobs and memory devices as machines, and ask whether it is possible to respect the scheduled work and, if not, which is the minimum number of additional memory devices that must be added, or which is the maximum reward we can obtain by scheduling a subset of the jobs.

* Corresponding author.
E-mail addresses: angele@eco.unibs.it (E. Angelelli), bianche@eco.unibs.it (N. Bianchessi), filippi@eco.unibs.it (C. Filippi).

http://dx.doi.org/10.1016/j.cor.2014.06.002
0305-0548/

The above described problems (both in the decision and optimization versions) can consider the generalization of, and the link between, two well known and widely studied classes of problems at the same time. The first class, known as interval scheduling problems, includes problems where typically each job claims for the exclusive assignment of one machine ($r_j = R, m \geq 1$). The second class, known as resource allocation problems, includes problems where jobs compete for the resources of a single machine ($r_j \leq R, m = 1$).

The feasibility question has been addressed in Angelelli and Filippi [1]. In this paper we focus on the first optimality question, the second one will be approached in a future work. Thus, our problem can be explicitly defined as follows:

Operational Interval Scheduling with a Resource Constraint (*OISRC*):

INSTANCE: m identical machines, with each machine owning R units of a renewable resource; n jobs, requiring processing in time interval $[s_j, f_j)$, using r_j units of resource and having a value v_j ($j = 1, \ldots, n$).

QUESTION: Which is a job subset of maximum total value with the property that each job can be processed by a machine so that no two jobs processed by a same machine overlap and the resource availability of each machine is always respected?

After discussing the complexity of OISRC and its connections with interval scheduling and resource allocation, we give the following contributions:

- an approximation result based on a greedy approach;
- a column generation technique based on a natural Dantzig–Wolfe decomposition of a compact integer programming formulation;
- some simple greedy heuristics;
- a restricted enumeration heuristics;
- implementation of each proposed algorithm and testing on randomly generated instances.

We show that the column generation technique offers a dramatic speedup with respect to the direct solution of the integer programming formulation, while the heuristic procedures illustrate the tradeoff between computing time and quality of the obtained solution.

2. Related literature

2.1. Interval scheduling

There is a sizable amount of the literature on interval scheduling problems, but research on this subject has often been tailored to the particular application. Here we mention only the more relevant studies with respect to the present work. We refer to Kolen et al. [29] and Kovalyov et al. [30] for an extensive treatment of the literature on interval scheduling.

Dantzig and Fulkerson [17], Gertsbakh and Stern [22] and Gupta et al. [24] among others study the Fixed Job Scheduling Problem (FJS), where n interval jobs and m identical machines are given, and the question concerns the existence of a feasible schedule for all jobs. This feasibility question turns out to be equivalent to the following, called Basic Interval Scheduling Problem in [29]: What is the minimum number of machines necessary to schedule all jobs? These basic problems can be solved in $O(n \log n)$ time, which is the best possible [24].

FJS has been generalized in various ways. Dondeti and Emmons [20] consider two classes of machines and a cost for processing a job depending on the machine class it is assigned to. A polynomial time algorithm based on network flow techniques is proposed. Huang and Lloyd [26] show that with three or more machine classes the problem becomes NP-hard and they give some approximation results. Bathia et al. [7] introduce availability intervals and a hierarchical structure for machines, giving complexity and approximation results. Kolen and Kroon [28] formalize and classify the complexity of problems with compatibility classes between jobs and machines.

Arkin and Silverberg [3], Bouzina and Emmons [10], Bathia et al. [7] Kroon et al. [31] among others consider settings where a value is associated with each job and the problem is to maximize the value of processed jobs, ranging from polynomially solvable to NP-hard problems.

Recently, Ng et al. [34] consider an interval scheduling problem on unrelated machines, where each assignment of a job to a machine yields a value, and the objective is to find a subset of jobs and their feasible assignments so that the total value is maximized. For this strongly NP-hard problem, they propose an exact algorithm based on a reduction to a maximum weight clique problem and several heuristics.

2.2. Scheduling with a resource constraint

Problems in this class have been proposed in the literature under different names. They differ from each other in the sense that they can be seen as restrictions of more general problems.

Resource Allocation Problem (RAP) [18], also called Bandwidth Allocation Problem (BAP) [16], is an optimization problem. Here, n interval jobs, each one with a resource requirement r_j and a value v_j (e.g., priority or profit), are given. The problem asks for a job subset of maximum total value that can be feasibly scheduled on a single machine with a given resource availability. A generalization of RAP where the resource capacity varies over time was studied as temporal knapsack problem in Bartlett et al. [6] and Caprara et al. [12]. In Chen et al. [16] a particular case of RAP is studied where job values are given as $v_j = r_j(f_j - s_j)$. They also introduced the Storage Allocation Problem (SAP) which is the restriction of RAP to the case of storage layout.

The Dynamic Storage Allocation (DSA) is a decision version of SAP where the question is whether all jobs can be feasibly scheduled (see [21]).

Chen et al. [16] show that BAP and SAP are NP-hard even in the very restricted case of $s_j = 0, f_j = 1, v_j = r_j$ for all j. Moreover, they show that SAP is strongly NP-hard even when the request sizes are bounded. They derive this property from the fact that DSA is strongly NP-complete [21] and it remains so even when the required resources r_j are either 1 or 2 (see [35]).

Calinescu et al. [11] give a randomized polynomial-time approximation algorithm for RAP with guarantee arbitrarily close to 2. They also give a simpler, deterministic algorithm with guarantee 3 and a simple rounding algorithm with guarantee 2 for the special case where each job requires no more than half of the available resource units.

Darmann et al. [18] prove that RAP remains NP-hard even when all the tasks profits are identical. They also give a polynomial-time approximation algorithm with guarantee arbitrarily close to 2 for the special case where no job interval is contained in another job interval. The strong NP-hardness of general RAP has been proven by Bonsma et al. [8].

RAP can be seen as a flow problem with side constraints. Consider the Unsplittable Flow Problem (UFP) (see [4,13]), where an undirected graph $G = (V, E)$ is given, whose edges $e \in E$ have capacity c_e. A set of n pairs (s_j, t_j) of vertices have a value v_j and a demand r_j each. The problem asks for a set of paths connecting pairs (s_j, t_j) such that for every edge $e \in E$ the total demand of the paths traversing e does not exceed c_e and the total value of the selected paths (vertex pairs) is maximized. When the graph is a simple path then we have an UFP on a line graph, whereas when