Two-stage stochastic/robust scheduling using permutable operation groups: a constraint programming approach

Louis Riviere[1],[2] Christian Artigues[1] Hélène Fargier[2]

[1] LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
{louis.riviere,artigues}@laas.fr
[2] IRIT-CNRS, Université de Toulouse,Toulouse, France
fargier@irit.fr

1 Introduction

Even after being studied extensively, scheduling problems are still relevant today due to their wide range of concrete applications with more and more difficult variants, one of the most important variant being the introduction of uncertainty. Indeed, real life scheduling problem seldom have complete information over the input data, and all aspects of the problem can be subjected to uncertainty, task duration, resource availability, etc..
In this paper we study the performances of a particular 2-stage solution method for stochastic or robust scheduling based on permutable operation groups with a simple second stage reordering policy on a single machine problem with release dates, due dates, precedence constraints and aiming for the minimization of the maximum lateness over a sampled set of scenarios. We propose an efficient constraint programming approach for the computation of a set of solutions and results show that even within a limited time, the approach can reach better performances than the classical 2-stage stochastic/robust scheduling approach on the studied instances.

2 Permutable operations groups

Permutable operation groups (POGs) are a subset of partial orders on schedule represented by a sequence of group of tasks such that every possible permutation within each group is a valid solution to the scheduling problem. As such, a single POG represents a set of schedule. They were first introduced in 1989 and their interesting properties have since been studied extensively. More recently [1] applied POGs to an integrated production scheduling and delivery routing problem, computing group sequences using an exact MILP formulation and a tabu heuristic.
In a context of uncertainty, being able to rapidly switch to another feasible schedule is a powerful tool for the scheduler, and the group structure somewhat limits the nervosity of the solution.
It is easy to see that the set of all possible POGs includes all "one sequence" solutions, i.e. POGs where there is only one task per group. As such, POG solutions dominate the classical 2-stage stochastic/robust scheduling approach that outputs a single-sequence at the first stage and adjusts the start times according to this sequence at the second stage, but because of the much larger search space, it is unclear whether it can perform better in a limited amount of time. Applied in a scheduling context, or to be able to compute their objective value over an instance of a problem, a scheduling policy must be decided beforehand to sequence the task within groups. For our experiments, we use a simple "earliest release date first" policy, because of it's realistic applications (ready tasks are started when available). More complex policies could be implemented.

3 Constraint programming implementation

Previous implementation for finding an exact group solution to problems used a MILP formulation [1], which was inefficient for larger instances. The proposed method uses Constraint programming to decide group assignment and simulate scenario realisation. See the following model for the main ideas used for POG computation.

Algorithm 1 Constraint programming model for POG

```
for $s \in 1..NbScenarios$ do
  for (i,j) in $V^2$ do
    $g_i > g_j \Rightarrow End(Jobs[i]) \leq$ Start(Jobs[j]) {$g_i$ denotes the group number of task $i$.}
    $g_i < g_j \Rightarrow End(Jobs[j]) \leq$ Start(Jobs[i]))
    if ReleaseDate[i,s] < ReleaseDate[j,s] then
      $g_i == g_j \Rightarrow End(Jobs[i], s) \leq$ Start(Jobs[j],s))
    else
      $g_i == g_j \Rightarrow End(Jobs[j], s) \leq$ Start(Jobs[i],s))
```

On the other hand, the classical robust approach is also computed using Constraint programming. Adapting the proposed algorithm used in IBM's *"stochastic jobshop"* exemple, we search for the schedule minimizing maximum Lmax simulating a Right shift repair method and using the **IloSameSequence()** constraint to enforce sequence unity across all scenarios.

For both methods, we set the SearchType to Depth first with 1 worker and A time limit varying from 1 minute to 15 minutes. For the POG solver, setting search phase starting with group assignment was paramount for it's efficiency.

4 Computational results

Both solvers are compared running on a [Xeon E5-2695 v4 @ 2.10GHz] CPU on a set of instances with parameters **N** : the number of tasks (from 10 to 100) ; **S** : the number of scenarios (from 2 to 25) ; **variation** : an indicator of the standard variation the scenario values were sampled from ; and **precedence_density** : a measure of the density of the precedence graph. Results show that for small values of N, the POG solver is able to solve the problem optimally, but little gain is to be made. As N grows, the POG solver outperforms the one schedule solver with an average gain of 10% for 50 tasks, but as N keeps growing, the proportional gain recedes, even though the absolute gain still grows. Larger values of S also favors the POG solver thanks to it's flexible solutions, going from an average gain of about 1% for 2 scenarios to 14% for 25 scenarios. Also unsurprisingly, more variation amongst scenarios favors the POG solver, on average the single schedule solver does slightly better when variation is less than 0.15, and reaches a peak proportional difference for 40% variation, with a 19% increase in score on average, but more variation doesn't seem to profit the POG solver as the scenarios become too distant from one another. Precedence density also has a great impact on the relative performance of the solvers. Low density favor the POG solver reaching an average gain of 22% for 1% density, but densities over 15% favor the single-schedule solver slightly on these instances as the precedence constraints force the POG solver to produce sequence-like solutions. Time allocated to the computation has little effect on score improvement for the sequence solver with a score improvement of only 1.5% between a 1 min computation and a 15 min one whereas the POG solver sees an 8% gain in the same conditions.

Références

[1] A. Cheref, C. Artigues, and J.-C. Billaut. Online recoverable robustness based on groups of permutable jobs for integrated production scheduling and delivery routing. Technical Report 16213, LAAS-CNRS, 2016.