HAL open science

A hybrid decomposition approach using increasing clusters for solving scheduling problems with minimal and maximal time lags

Freddy Deppner, Marie-Claude Portmann

To cite this version:

Freddy Deppner, Marie-Claude Portmann. A hybrid decomposition approach using increasing clusters for solving scheduling problems with minimal and maximal time lags. Tenth International Workshop on Project Management and Scheduling (PMS 2006), EURO & University of Poznan, Apr 2006, Poznan/Poland, 4 p. inria-00114183

HAL Id: inria-00114183

https://inria.hal.science/inria-00114183v1

Submitted on 15 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A hybrid decomposition approach using increasing clusters for solving scheduling problems with minimal and maximal time lags

Freddy Deppner ; Marie-Claude Portmann

LORIA-INRIA Lorraine, Campus Scientifique, BP 239, F-54506 Vandoeuvre Cedex

freddy.deppner@laposte.net marie-claude.portmann@loria.fr

tel. (+33) 03 88 15 18 04 tel. (+33) 03 83 58 41 85 fax (+33) 03 83 57 97 94

Abstract

In this article we present a priority rule construction algorithm coupled with a cluster decomposition method in order to construct feasible solutions for general scheduling problems with minimal and maximal time lag constraints. Algorithm performances have been tested on job shop scheduling problems and also on a real chemical freeze-drying process.

Keywords: scheduling, priority rule algorithm, minimal time lag, maximal time lag, makespan, cluster decomposition

1 Introduction

Already in 1985, Hodson et al [HMP85] have described a plant for preparating, cooking and chilling meals where chilling must start within 30mn of the completion of cooking otherwise the product is declared unsafe for human consumption. In a chemical process, filtration must immediately follow formulation in order to avoid precipitation. The use of sterilised materiel must be effective in a given time window after sterilisation, otherwise the production is cancelled.

Except scheduling problems with minimal time lags and extreme case of no-wait, publications on the topic remain poor. One possible explanation could be that the construction of a feasible solution is NP-complete already for some one machine scheduling problems. Leung et al [VW84] have demonstrated that the $1|prec(l), p_i = 1|C_{max}$ problem is NP-complete. By adding a maximal time lag constraint between the first and the last operation equal to the makespan C_{max}, construction of a feasible schedule is equivalent to find an optimal solution for the initial problem. For one machine scheduling problems, Hurink and Keuchel [HK01] have developed a local search procedure based on a extension of critical blocks. Franck et al [FNS01] have proposed a priority rule approach based on the decomposition of operations according to strong components of the disjunctive graph representing the problem and called cycles or clusters presently. They were able to construct quickly feasible solutions for a wide area of scheduling problems. Nevertheless, performances can be poor while, without clusters, performances are often better, the difficulty being then to find feasible solutions. We take advantage of the speed of priority rule algorithms in order to built several solutions with increasing size of clusters until obtaining a feasible solution.

In section 2 we give a formal definition of the considered problem. In section 3 we present a first extension of a classical priority rule algorithm integrating the management of maximal time lags. In order to overcome the poor results of this algorithm in finding feasible solutions, we present a decomposition method using clusters with variable sizes. Some numerical experiences on job shop scheduling problems and a real chemical process with constraints like calendars will be presented in section 4.

2 Notation

In this section we give a formal definition of the considered problem. Let $J = J_1, .., J_n$ be the set of n jobs and $M = M_1, .., M_m$ the set of m reusable resources that can perform one operation at time. Non pre-emption is assumed. For operation O, we note

- r_O its release time
- p_O its processing time (we suppose that processing times are fixed)
- s_O its starting date
- c_O its completion date

- M_O the resource executing operation O

The system must satisfy a set of inequalities called time lags: $s_{O_i} + d(O_i, O_j) \leq s_{O_j} \quad d(O_i, O_j) \in I\!R$

- $d(O_i, O_j) = p_{O_i}$ (classical precedence constraint)
- $0 \leq d(O_i, O_j) < p_{Oi}$ (overlapping)
- $d(O_i, O_j) > p_{O_i}$ (precedence constraint with additional time lag)
- $d(O_i, O_j) < 0$ (maximal time lag, O_i must start its execution at most $|d(O_i, O_j|$ time units after O_j starting time)

General precedence constraint can be defined by a graph between operations: assembly shops can be considered as well as job shops. We define a scheduling horizon $[h; H]$. We consider that we fail in finding a feasible schedule if some operation has to be strictly scheduled after time H.

3 Priority rule algorithm

3.1 An extension of classical priority rule algorithm

Due to their ease of implementation and low time complexity, priority rule algorithms are the most used heuristics for solving scheduling problems (see Baker [Bak74] for a presentation of active schedule generator). At each step, they place the operation with greatest priority, which does not create just before it an idle time on the machine in which an other operation could be completely scheduled. Algorithm ARP-MD on figure 1 is an extension of such an algorithm in order to integrate minimal and maximal time lags constraints. We take into account only minimal time lags in starting time computation. When the placement of operation O induces a maximal time lag constraint violation with previously placed operations, we choose to delay to the right these operations that doesn't satisfy a maximal time lag with O. In order to do that, we increase release dates of these operations of exact amount of time necessary to satisfy the maximal time lag constraint. We then remove operations from the stack of scheduled operations since the first one that doesn't satisfy a maximal time lag constraints with O till O itself.

1. L is the set of all operations ; R is the stack of already scheduled operations (initialy $R = \emptyset$) ; k is the placement rank of scheduled operations (initialy $k = 0$).
2. **While** $L \neq \emptyset$:
   (a) Use the classical selection of Baker active schedule based on priority rule in order to select operation O to be placed.
   (b) Build set $P \subset R$ of all scheduled operations that doesn't satisfy a negative time lag with O.
   (c) If $P = \emptyset$, place O on M, remove O from L, add O to R, Placement rank of O is k. Set $k = k + 1$.
   (d) Else
      i. for each $O' \in P$, modify release time : $r_{O'} = s_O + d(O, O')$
      ii. $k = \min_{O' \in P}(\textit{placement rank of } O')$. Remove from R all operations with placement rank greater or equal to k, put them again in L and remove them from the resources.

   End While.

Figure 1: PRA-MD Modified Priority Rule Algorithm

We stop if modified release dates overpass scheduling horizon end H, which ensures algorithm convergence.

3.2 Decomposition method - Iterative Growing method

Franck et al [FNS01] have proposed a decomposition method according to strong components of the conjunctive graph. Each strong component (or cluster) is scheduled with algorithm MPRA. In such way, operations linked by a maximal time lag constraint are placed together without any additional operation belonging to another cluster and it is easier to find a feasible schedule for each cluster. But instead of taking always the strong components, we build several solutions with increasing size of clusters.

Initially clusters are reduced to single operations. A first solution is built without taking care of maximal time lags. Only once the solution is completely built we check the validity of all maximal time

lag constraints. For each couple of operations that don't satisfy a maximal time lag constraint, we build a new cluster containing both operations and also intermediate operations according to the precedence graph. We rebuilt a new solution with those new sized clusters and so again until obtaining a valid solution. The algorithm PRA-C-IGM called Iterative Growing Method is described on figure 2.

Step 1

1. L is the set of all clusters reduced to single operations ; R is the stack of already scheduled clusters.
2. **While** $L \neq \emptyset$:
   (a) Let $T \subset L$ be the set of clusters that must be scheduled. For each cluster in T, compute starting time (smallest starting time of operations in the cluster) and completion time (greatest completion time of operations in the cluster) using PRA-MD. Let $C \in T$ be the cluster with smallest completion time c.
   (b) Let $C' \in T$ be the cluster which starting time is strictly smaller than c and with highest priority. Place operations of C',remove C' from L, add C' to R.

   End While.
3. If all maximal time lag constraints are satisfyed, exit program.
4. Else, for each couple of operations (O_1, O_2) belonging to different clusters C_1 and C_2 such that $s_1 + d(O_1, O_2) > s_2$, merge C_1 and C_2.
5. Goto Step 1

Figure 2: IGM - Iterative Growing Method

Note: when merging clusters C_1 and C_2, we create a new cluster containing C_1 and C_2 but also "intermediate" clusters according to precedence constraints.

3.3 Numerical Experiences

We have tested MPRA and PRA-C-IGM on job shop benchmarks of size 5x5, 5x10, 5x20, 10x10, 10x20, and 10x30. Processing times have been generated randomly between $p_{min} = 10$ and $p_{max} = 50$. Horizon scheduling has been fixed to $[0; 2.n.p_{max}]$. For each job i, 1, 2 or 3 pairs of operations $(O_{i,j}, O_{i,k})$ $(j < k)$ have randomly been selected with a maximal time lag constraint $d(O_{i,j}, O_{i,k}) = -C \sum_{l=j}^{k-1} p_{i,l}$. Constant C is the "tension" on maximal time lags, initially $C = 3$. We measured algorithm performances for C varying from 3 (low tension) to 1 (high tension corresponding to no wait constraint).

Figure 3 for test set size 10x20 is a typical example of the behaviour of our algorithms when considering the mean number of starting time calculations and makespan evolution. We have added also results for algorithm PRA-C-MAX when considering all strong components. In abscissa we have the tension C, in ordinate the log of the mean number of starting time calculations over the 50 instances of our test set. If the number of starting time calculation for algorithm PRA-C-IGM follows a "normal" progression when the tension is increasing (C varying from 3 to 1), this number is exploding for algorithm PRA-MD.

The makespan evolution presents the same behaviour. We have only considered solutions where all jobs have been scheduled before the horizon end. If PRA-MD take fully advantage of the placement of individual operations when tension C is weak, the degradation becomes sensible for high tension. Moreover, last values are not relevant since for $C = 1.2$ (resp. $C = 1.1$ and $C = 1$) only 54% (resp. 16% and 2%) solutions are feasible. Conversely, algorithms PRA-C-IGM and PRA-C-MAX were always able to find a feasible solution and PRA-C-IGM seems to be a good compromise between PRA-MD and PRA-C-MAX both in terms of speed and solution quality.

Algorithm PRA-C-IGM has been tested on a real freeze-drying process in a chemical enterprise (figure 4). Additional constraints like calendars have to been considered. Calendars are modelized with operations that are placed once for all on resources inducing thus unavailability periods. The enterprise gave us a typical production planning over two weeks and considered as optimal. We have been able to improve it of nearly a half day (3%). The same experience with PRA-MD did not bring us satisfaction. IF ARP-MD was able to construct feasible solutions in some case, the makespan was over 21 days instead of the 14 days of the original planning. It was also very sensible at each variation of the data. One job more and ARP-MD was unable to schedule before the horizon end.

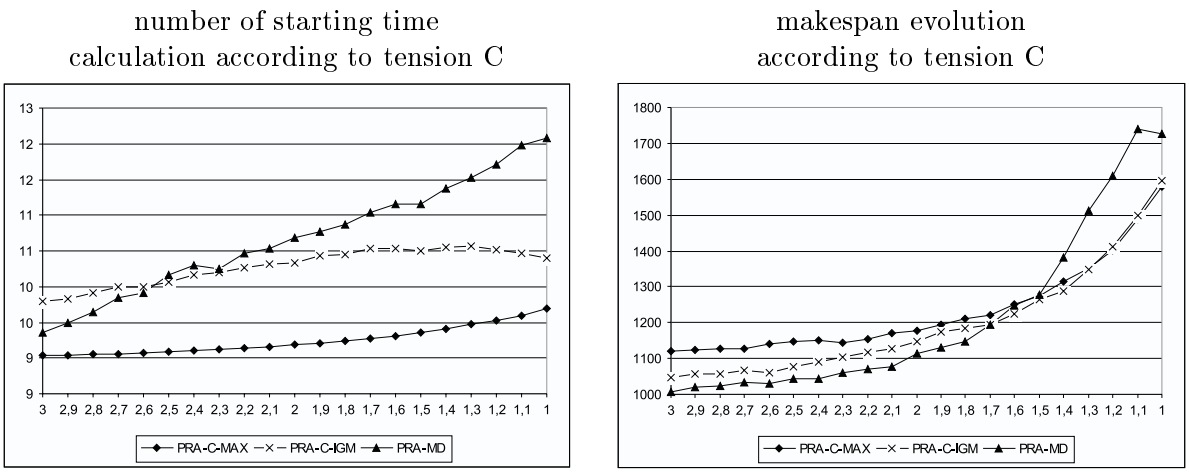

Figure 3: PRA-MD vs PRA-C-IGM

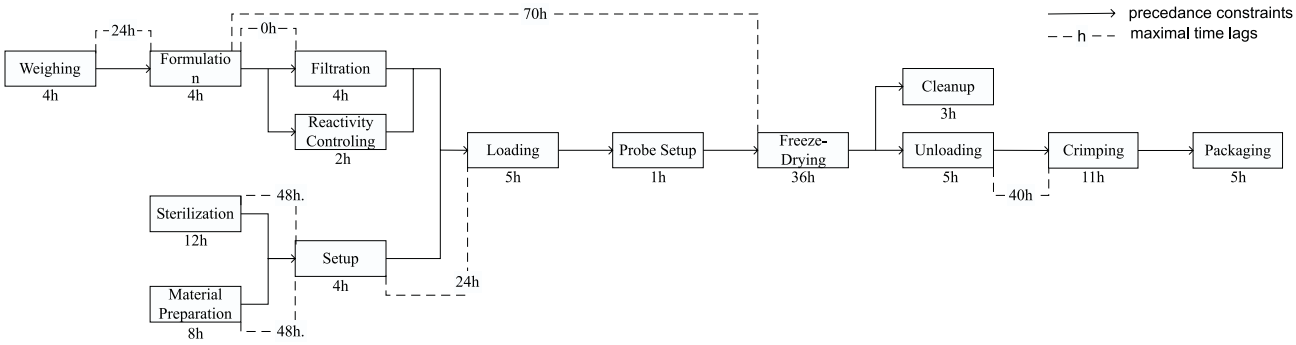

Figure 4: freeze-drying process

3.4 Conclusion

Algorithm PRA-C-IGM takes advantages of the speed of priority rules algorithm and the decomposition method in order to build quickly good feasible solutions. We also designed a Dynamic Growing Method where clusters are increased in parallel with the construction of the schedule. Its performance is globally similar to the performance of PAR-C-IGM we choose to present here.

The choice of the priority rules between clusters and operations influence the quality of the solutions. In order to improve performances in terms of makespan (or other criteria, we can consider), we will integrate PRA-C-IGM in a genetic algorithm, where genes will contain, for example, the priority of the jobs.

References

[Bak74] K.R. Baker. *Introduction to sequencing and scheduling.* John Willey & sons, Inc., 1974.

[FNS01] B. Franck, K. Neumann, and C. Schwindt. *Truncated branch-and-bound, schedule-construction and schedule-improvement procedures for resource-constrained project scheduling*, volume 23 of *OR Spektrum.* Springer Verlag, 2001.

[HK01] J.L. Hurink and J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time lags. *Discrete Applied Mathematics*, 112:179–197, 2001.

[HMP85] A. Hodson, A.P. Muhlemann, and D.H.R. Price. A micocomputer based solution to a practical scheduling problem. *Journal of Operational Research*, 36(10):903–914, 1985.

[VW84] J.Y.T. Leung O. Vornberger and J.D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM Journal on Computing*, 13(3):650–667, 1984.