A Novel Approach to Aggregate Scheduling in Project-Oriented Manufacturing

András Kovács

Budapest University of Technology and Economics
Magyar Tudósok körútja 2/d, 1117 Budapest, Hungary
akovacs@mit.bme.hu

Introduction

In this paper we address medium-term scheduling in project oriented manufacturing systems. These systems may execute hundreds of thousands of manufacturing operations under various capacity and technological constraints within the typical 3-6 months long horizon. This challenges any branch of operation research or artificial intelligence.

We suggest an aggregate scheduling approach to cope with this challenge in case of project-oriented environments, characterized by complex projects with strict individual deadlines. Dimensions of aggregation are time and tasks: the scheduling problem is solved with an aggregate time unit in the order of magnitude of one week. Furthermore, not individual manufacturing *operations* are scheduled, but so-called *activities*, aggregate units of work built of logical groups of discrete operations.

We consider the aggregate scheduler as a part of a hierarchical production planning and scheduling system. It is not simply expected to generate schedules which are good enough in themselves, but also takes into considerations the effects of its decisions on lower levels, i.e., on job-shop schedules.

The main contribution of the paper is a detailed analysis of the effects of decisions made during activity formation on the outcome of the overall scheduling process. Based on these investigations, we suggest an activity model which is quite different from those used by previous approaches. We consider the determination of the best suited activity models an optimization problem, and advise an algorithm to construct them.

The paper is structured as follows. We start with a brief literature overview, followed by the presentation of our scheduling algorithm. The nucleus of the article focuses on the analysis of activity formation decisions. Finally, we describe our first experiences with our pilot system working on real industrial data, and draw the conclusions.

Related Work

In the related literature, two fundamentally different approaches compete in solving medium-term scheduling problems. Industrial practice is still dominated by lead time-based techniques, dividing the production planning problems into capacity and material flow oriented sub-problems. The inadequacy of lead time-based solutions derives from the fact that the execution time of a project significantly depends on a number of actual parameters: shop load, priorities, etc. Current researches try to overcome this difficulty by applying dynamic methods for setting the project milestones. Beyond historical data, priorities and predicted shop load on the route of the arriving project, they take distribution of the work load on the resources, coefficients characteristic for the shop control methods, and many more as input of rule-based systems, regression analysis or neural networks, see for example (Raaymakers and Weijters 2003) or (Sabuncuoglu and Comlekci 2002).

The other family of methodologies, referred to as aggregate scheduling or aggregate production planning, covers a fairly wide range of algorithms (Bitran and Tirupati 1993). In batch process industries, typical questions at this level concern lot sizing, production sequencing and resource assignment. Herein we focus on project-oriented environments, just as (Hackman and Leachman 1989) did, and wish to determine a valid assignment of activities to time units.

A common characteristic of these approaches is that they merge discrete operations requiring the *same resource or set of resources* into one aggregate activity. Since a part can iterate between the same resources several times during its production, the temporal interdependencies of these activities can be very complex. In (Leachman and Kim 1993), a highly sophisticated model is presented for describing valid time-assignments of aggregate activities and relations between them. They use variable duration activities with prescribed intensity curves, overlap relationships, as well as balance-type relationships between intensity curves of dependent activities.

This approach faces serious difficulties in obtaining the required input data, that is seldom available in existing technological databases, nor can be trivially reconstructed from historical archives.

A Mixed-Integer Formulation of the Aggregate Scheduling Problem

Formulation of the aggregate scheduling problem is based on a recently developed resource-constrained project scheduling model (Kis 2002). The special features of this approach are as follows:

- It works with variable-intensity, variable-duration, but fixed-volume tasks, which well fits the concept of aggregate activities composed of a number of discrete operations in a closer detail.
- The tasks in the model may require an arbitrary mix of resources.
- It produces solutions in which each task is scheduled into as few time units as its volume and the capacity constraints allow. A more exact formulation of this property and the gain on it will be presented later.
- It has proven to be fast enough to solve large-scale real-life problems to optimality.

In order to be self-contained, herein we give a brief presentation of the model and the algorithm.

Resource-constrained project scheduling problems (RCPSP) are concerned with scheduling a number of discrete tasks, each requiring some resources. Constraints due to the limited capacities of resources and precedence relations between the tasks are prescribed. The classical model assumes fixed task durations and a constant rate of resource usage during the entire processing of each task (Demeulemeester and Herroelen 2002, and Weglarz 1998).

However, in *aggregate* scheduling the above assumptions cannot be taken and there is also no need to generate detailed solutions for future periods that will certainly be different from the anticipated. Hence, the classical RCPSP model was extended by allowing (1) *preemption* of activity execution, (2) *variable-intensity* activities, and (3) *continuously* divisible resources.

An instance of the problem is given by a set $N = \{1,...,n\}$ of *activities*, a set $R = \{1,...,r\}$ of continuously divisible and renewable *resources,* and a directed acyclic graph $D = (N, A)$ representing *precedence constraints* among the activities. Each activity $i \in N$ must entirely be processed within its *time window*: between its earliest starting time e^i and deadline d^i.

Each activity may require the simultaneous use of some *resources*. The entire processing of activity i requires a total of r_k^i units of resource k, for each $k \in R$. The *intensity* of each activity may vary over time, and the resource usage is proportional to the intensity. If x_t^i is the intensity of activity i in time period t, then it requires $r_k^i * x_t^i$ units of resource k in that period. However, the intensity of executing an activity is limited: in any time period $t \in [e^i, d^i]$ at most $a^i \le 1$ fraction of activity i may be completed.

The *capacity* of each resource $k \in R$ is fixed period by period over the horizon. In each time period t, a certain internal capacity of each resource k is available. Internal resource capacities can be used free of charge. Additional external capacities are also available, but at the expense of some cost per resource units.

The solution of the problem consists of determining for each activity i an intensity x_t^i in each time period $t \in [e^i, d^i]$ such that $0 \le x_t^i \le a^i$, $\sum x_t^i = 1$, all the precedence constraints among the activities are fulfilled, the resource demands do not exceed the resource availabilities in any time period, and it is optimal according to some objective function.

The above problem has been formalized as a mixed integer-linear program, and solved by a branch-and-cut algorithm. Experience gathered on benchmark problems and real-life industrial data (Márkus et al. 2003) has shown that though the problem is NP-complete in the strong sense, it is a viable approach for solving even very large problem instances.

This algorithm is able to solve the above problem for optimization criteria which can be expressed as a linear function of the x_t^i variables, or established with a dichotomizing search. These include project duration, maximum tardiness or weighted tardiness or minimum work in process (WIP). In our current settings, we minimize the cost of external resource usage first, and in a second run, minimize WIP with the previous bound on external resource usage.

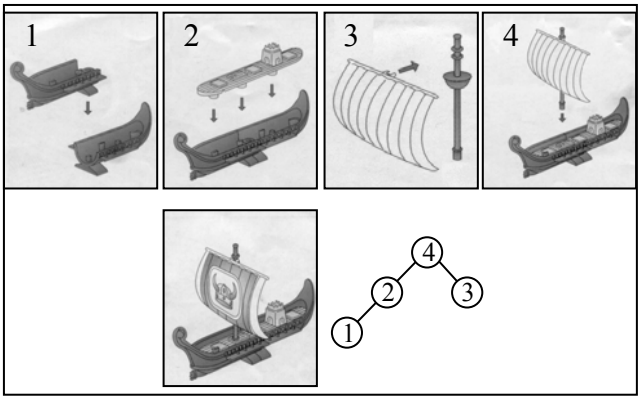

Fig. 1. Assembly operations and project tree of a toy ship. Our project trees contain up to 500 vertices.

Aggregate Scheduling in Components Manufacturing

In components manufacturing each project can be described by a rooted tree, the so-called *project tree*, whose vertices represent manufacturing operations. Vertices with several children denote joining operations, while those with one single child can either represent machining operations or joining a purchased part to a sub-assembly. The execution of the project over time then advances from the leaves towards the root that stands for the finishing operation of the final product. Edges represent strict precedence relations, i.e., the sons of an

operation *v* must all be completed before *v* could be started. Fig. 1 shows the creation of a project tree for a simplistic assembly.

Vertices of the project tree are then to be contracted into components representing the aggregate activities. This partitioning of the project tree will be referred to as the *activity model* of the project, and will be illustrated by the *aggregate project tree*, in which vertices of the project tree belonging to the same activity are contracted into one vertex.

The following section is dedicated to showing that the way activity models are created has a crucial influence on the quality of the aggregate schedule. Departing from identifying the relevant aspects we get to the definition of optimal activity models and also show that such activity models can be constructed by a polynomial-time tree partitioning algorithm.

Consistency of the Schedule

In most previous approaches to aggregate scheduling, the consistency of the schedules was endangered by technologically infeasible solutions hidden behind the abstract notion of activity, since many low-level technological constraints were abandoned in the activity models.

We suggest an approach where operations with arbitrary resource requirements, but constituting a connected component of the project tree can be merged into aggregate activities. In our model of the discrete scheduling problem, all technological constraints are expressed by end-to-start precedence relations between operations. For each job-shop level precedence constraint, if the two connected operations are ordered into the same activity, then the constraint is omitted from the aggregate model. Otherwise, a conventional end-to-start precedence constraint is posted on the two aggregate activities. Note that the precedence graph of the activities will also form a tree. Furthermore, the resource requirements of an activity are the sum of resource requirements of the contained operations. Illustrations of such activity models are presented in Fig. 2 and 3.

Clearly, this method guarantees that aggregate schedules can be disaggregated to feasible discrete operation sequences, though, it might happen that the execution of the latter does not fit exactly into the time window as assigned in the aggregate schedule.

Time-Feasibility of the Schedule

The aggregate schedule designates one or more aggregate time units for the execution of each activity. We define a consistent aggregate schedule *time-feasible*, if it has a job-shop level disaggregation such that each discrete operation falls into one of the time units designated for its containing activity. Clearly, the time-feasibility of a schedule cannot be guaranteed only by prescribing that in each time unit the total duration of operations on a resource should not exceed resource capacity: precedence constraints can prevent resources processing operations continuously.

An aggregate activity is called *n-feasible*, if its discrete operations can be scheduled into a time window whose length is of *n* aggregate time units, provided we omitted all other activities in the plant. Note that determining the feasibility number of an activity is NP-complete, e.g., the job-shop scheduling problem can easily be reduced to it.

Furthermore, the *size* of an activity or its corresponding component in the tree will denote the sum of the durations of the contained operations. In a given aggregate schedule, we define an activity *broken*, if its execution is divided between several time units in the aggregate schedule. Otherwise, we call it *unbroken*. Now, having all the necessary definitions, one can make the following observations:

- In case all activities are unbroken in an aggregate schedule, the inequalities of the project model describe exactly that the aggregate schedule provides a correct approximation of a valid discrete schedule according to energetic reasoning (Baptiste, Le Pape and Nuijten 1999). In all other cases, the estimation is less punctual, since there would be operations whose resource requirements are distributed among various time units, while the operation itself is executed in exactly one.
- If for each activity *i* its maximal intensity is $a^i = 1$, then the above presented algorithm provides an aggregate schedule in which the predominant part of activities will be unbroken.

The latter holds due to a side effect of the linear program formalization. In each stage of the branch-and-cut procedure, the simplex algorithm solving the relaxed linear program returns a basic solution, in which all but as many variables x_t^i as the rank of the matrix of the linear program are zeros (Vajda 1961). Consequently, if $a^i = 1$, i.e., the execution of the activity may fit in one time unit, then *i* will really be unbroken, unless a strict resource capacity constraint precludes it. Experiments with real-life data have show that the ratio of unbroken activities was between 90 and 100 % even for highly loaded problem instances.

All the above observations motivate us to build activities that fit into one aggregate time unit, in order to receive as good approximation of a valid job-shop level schedule as possible. This can be established by setting $a^i = 1$ for all activities. The 1-feasibility of each activity is then a necessary condition of the time-feasibility of the aggregate schedule. It is clearly not a satisfactory condition, since potential interactions of activities are not considered here.

Accordingly, we have to partition the project tree into 1-feasible components. We are looking for a *minimum-cardinality* partitioning respecting this constraint, because a higher number of smaller activities (1) would not ameliorate the estimation of the discrete schedule, but (2) would increase computational complexity and (3) we will see another good reason to do so in the following section.

Since finding maximal 1-feasible components of the project tree is computationally prohibitive (even testing the 1-feasibility of a component is NP-hard), and such a tense activity model would easily result in time-infeasible aggregate schedules, we approximate it by looking for activities the size of which does not exceed the length of an aggregate time unit. We suppose that there is no operation whose duration exceeds this time limit.

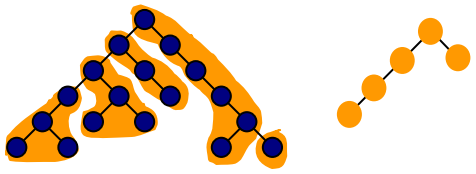

Fig. 2. A minimal cardinality size-bounded activity model of a sample project. Project tree and aggregate project tree. (Height=4, Cardinality=5)

Extra Constraints Added

Though the aggregate scheduling problem is usually considered as a relaxation of the job-shop level problem, some extra constraints are introduced or strengthened as well during aggregation. A precedence constraint states that the connected operations or activities have to be executed in the given order, *in distinct time units*. While this on the job-shop level allows the second operation to start directly after the first has ended, the inherent constraint on the aggregate level prescribes that an aggregate time-unit change (a weekend, for example) has to elapse between the two activities. Consequently, if the activity model of a project contains a precedence chain of length p, then in an aggregate scheduling problem working with this activity model the completion of the project takes at least p aggregate time units. This makes possible that though the project on Fig. 2 could be executed in 2 time units, the inappropriate activity model requires a time window of at least 4 units. With a more adequate activity model, like the one on Fig. 3, the same project can be scheduled within the minimal time window of 2 units.

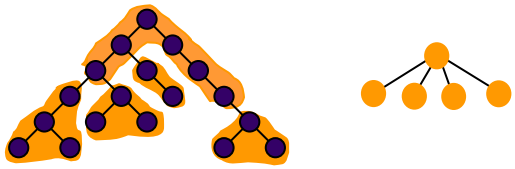

Fig.3. The minimal-height, minimal-cardinality size-bounded activity model of the sample project. Project tree and aggregate project tree. (Height=2, Cardinality=5)

Hence, an activity model containing a maximal precedence chain as short as possible – i.e., a *minimum height* partitioning of the project tree – is looked for. Note that in another view, this means increasing the parallelism between activities.

Optimal Activity Models

After the above considerations, we give the definition of the optimal activity model of a project as follows:

The optimal activity model of a project is a partitioning of the project tree into connected components such that

- *each component respects the weight limit of W, the number of working hours in an aggregate time unit;*
- *the height of the partitioning is minimal;*
- *with the above prerequisites, the cardinality of the partitioning is minimal.*

Then, to each component in the partitioning corresponds an activity. Both its volume and maximal intensity equals the sum of the contained operations' durations, while its resource requirements are computed as follows:

$$r_k^i = \sum_{Op_j \in Act_i} \rho_k^j , \quad i \in N, k \in R$$

Apparently, the creation of the optimal activity model of a project corresponds to a tree partitioning problem with regards to the bi-criteria given by minimum height and minimum cardinality. In (Kovács and Kis 2003), a bottom-up dynamic program is suggested to solve this partitioning problem in $O(n^3)$ time.

Note that since only technological plans are used to establish the activity models, and those are relatively stable, the partitioning can be done off-line.

Experiments

The algorithms suggested in this paper are meant to constitute a module of an integrated production planner and scheduler system under implementation in the Computer and Automation Research Institute of the Hungarian Academy of Sciences. Currently, we are performing the first test with industrial data on them.

The factory of our case study manufactures mechanical products of high value in a make-to-order manner, by using machining and welding centres, assembly and inspection stations. A typical project consists of 20 to 500 discrete manufacturing operations, each taking from 0.5 to 120 hours. Trees of these projects were shrunk into aggregate project models consisting of 1 to 10 activities. The number of resources is around 150. The horizon of the medium-term scheduling problem is 15-25 weeks, and its time unit of one week equals the horizon of the job-shop level problem. The latter is solved with a 0.1 hours horizon by a constraint-based scheduler.

Preparing the optimal activity models of projects never took more than a second per project. Using those activity models, we could create valid and reasonable schedules on both aggregation levels. However, unpredicted orders with tight time windows, whose release and/or due date were prescribed with daily precision required special dealing.

The most exigent deficiency of the current state of our algorithms seems to be the lack of an intelligent interplay between the two schedulers working on different aggregation levels and with different time scales. With the current settings, it often happens that the amount of work appropriated to a given week by the aggregate scheduler requires up to 10-20 % more processing time on the discrete level. However, we do not know whether this problem has a practical relevance, since it is an established custom for our industrial partner to slightly overload the predictive schedule, in order to compensate execution-time shortfall of jobs.

Conclusions

In this paper we have presented a novel approach to aggregate scheduling in project-oriented environments. Our first experiments confirmed that the aggregate scheduling approach is an adequate means to handle the complexity of the medium-term scheduling problem, and to produce optimal schedules by handling together the material- and workflow oriented aspects of production.

Compared with previous aggregate scheduling techniques, we suggest that merging discrete operations of a connected component of the project tree into aggregate activities is profitable with respect to aggregation along resources. However, activity formation has a crucial influence on the quality of the aggregate schedule in several ways. For the special case of components manufacturing, where the project graph is a tree, we gave a definition of optimal activity models and also suggested a tree partitioning algorithm to build such models in polynomial time.

Acknowledgements

A part of the ideas published in this paper are results of joint work with Tamás Kis, József Váncza and András Márkus. Thanks are due to Tadeusz Dobrowiecki for his valuable comments. This research has been supported by the grant NRDP 2/040/2001.

References

Baptiste, P., Le Pape, C., Nuijten, W., 1999. *Satisfiability tests and time-bound adjustments for cumulative scheduling problems*, Annals of Operation Research 92, 305–333.

Bitran, G.R., Tirupati, D. 1993. *Hierarchical Production Planning*, In: Graves, S.C., Rinnooy Kan A.H.G., Zipkin, P.H. (eds), Logistics of Production and Inventory, 523-568, North Holland.

Demeulemeester, E.L., Herroelen, W.S., 2002. *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers.

Hackman, S.T., Leachman, R.C. 1989. *An Aggregate Model of Project-Oriented Production*. IEEE Transactions on Systems, Man, and Cybernetics, 19(2) 220-231.

Kis, T., 2002. *A Branch-and-Cut Algorithm for Scheduling Projects with Variable-Intensity Activities*. Submitted to Mathematical Programming.

Kovács, A. and Kis, T., 2003. *Partitioning of Trees for Minimizing Height and Cardinality*, Submitted to Information Processing Letters.

Leachman, R.C., Kim, S., 1993. *A Revised Critical Path Method for Networks Including Both Overlap Relationships and Variable-Duration Activities*, European Journal of Operational Research 64(1993), 229-248.

Márkus, A., Váncza, J., Kis, T., Kovács, A., 2003. *Project Scheduling Approach to Production Planning*, Annals of the CIRP 52(1). (in print)

Raaymakers, W.H.M., Weijters, A.J.M.M., 2003. *Makespan Estimation in Batch Process Industries: A Comparison Between Regression Analysis and Neural Networks*, European Journal of Operational Research 145 (2003) 14–30.

Sabuncuoglu I., Comlekci, A. 2002. *Operation-Based Flowtime Estimation in a Dynamic Job Shop*, Omega 30 (2002) 423– 442.

Vajda, S., 1961. *Mathematical Programming*. Addison-Wesley.

Weglarz, J. (ed.), 1998. *Project Scheduling. Recent Models, Algorithms and Applications*, Kluwer Academic Publishers.