An Agent Model for Incentive-based Production Scheduling

J. Váncza and A. Márkus
Computer and Automation Research Institute
Hungarian Academy of Sciences
H-1518 Budapest, POB. 63, Hungary

Abstract

In agent models of manufacturing it is hard to reconcile autonomy and cooperation. The paper proposes a solution to this problem within the context of a dynamic production scheduling problem. Our organizational model is based on economic concepts, uses market rules and an incentive mechanism. The solution integrates dynamic order processing, advance least-commitment scheduling and dispatching.

1 INTRODUCTION

Manufacturing needs new organizational principles and structures to face new challenges: customer–driven production in a volatile market environment, globalization, the potential of electronic commerce and virtual enterprises, and the need of integrating human resources. Redundant functions and distributed responsibilities, tasks and resources are necessary for responding to changes, let they come either from internal disturbances or from external market conditions [11]. Transformations of manufacturing organizations already point toward network-like, dynamic and reconfigurable federations where production is carried out by more or less *autonomous* and *cooperative* production units.

Nowadays, such production units – small and large, simple and complex, machines, shops and companies alike – are often modeled in terms of *agents* [2, 12, 17, 23]. Agents have their own (i) beliefs about and (ii) preferences over the state of their environment, and (iii) have particular sets of actions to change it [25]. They operate in environments that are partly unknown and unpredictable. *Autonomous* agents have the opportunity and ability to make decisions of their own. *Rational* agents act in the manner most appropriate for the situation at hand and do the best they can do for themselves. Rationality can be bound by the computational complexity of a decision problem, the scarcity of resources, or by both. An agent with optimizing ambitions but with limited means is a *bounded* rational agent. In a community, agents have to *coordinate* with others; i.e. to take into account the other agents' actions when deciding what to do. Common goals and system-wide constraints require also *cooperation*: agents have to coordinate their actions in order to handle them appropriately. Communication, organization, protocols and search with local scope are just the most important means that make coordination practically executable [9, 26].

Agent technology is particularly appealing to model and solve production planning and control problems in manufacturing. Agents help to capture individual interests, local decision making using incomplete information, autonomy, responsiveness, robustness and modular, distributed, reconfigurable organizational structures. The application of agents in manufacturing raises new questions as well:

- Organization, communication and decision methods are required that *reconcile* autonomy and cooperation.
- The assumption of bounded rationality should be accepted. This makes, however, the problem of coordination and cooperation extremely hard.
- Agents have to respond to changes in their environment, but must have also a strong *commitment* to their own goals and plans. Otherwise they would bring nothing to completion. Hence, agents have to be *opportunistic*, by balancing between planned and reactive behavior.
- Time is a critical factor. Hence, solutions should be *adequate*; i.e., delivered no later then needed, even if the computing resources are distributed and limited.

We tried to investigate and solve the above problems within the framework of a particular *production scheduling* model. When defining this model, our basic consideration was that production scheduling needs not only new solution techniques [5], but also new, more realistic models [19, 20]. Moreover, our aim was to set up a model which is directly applicable in cases where individual and different interests are directly attached to the utilization of resources – as it is in the case of companies. Hence, our model extends the classical job shop scheduling model in several ways: it

- integrates *order processing*,
- requires *on-line*, dynamic scheduling,
- assumes a rich model of production resources, and
- makes *profitability* the key performance criterion.

Below, we present shortly this problem and give the conceptual overview and technical description of a multi-agent system aimed to solve it. The distributed decision mechanism of the system uses a novel commitment scheme which is based on the following principles:

- There may be *parallel assignments* for the same task; i.e., any task may be assigned to multiple resources.
- Resources may have alternative assignments, too. However, the resource assignments are *options* which give the right to perform a particular task, but rarely impose a strict obligation.

These commitment principles complement each other: on one hand they allow to build both task- (order-) and resource-oriented schedule alternatives, and on the other hand make it possible to select, combine and change schedule variants in an efficient way.

Section 3. gives an overview of the multi-agent model and Sections 4. and 5. describe elements of the decision mechanism. Then, simulation experiments are analyzed and finally our method is discussed in the context of similar approaches.

2 THE PRODUCTION SCHEDULING PROBLEM

2.1 Orders, jobs and tasks

In our model, customers' demand is transmitted to the shop by a dynamic *order stream*. Each i *order* has a fixed a_i *arrival time*, r_i *release time*, and d_i *due date* ($a_i \leq r_i < d_i$); as

well as a *contract price* CP_i and a *tardiness penalty weight*, W_i. Orders arrive dynamically, in parallel with processing of the earlier ones.

An order consists of a *task* sequence, where each task k of an order i requires a specific v_{ik} *volume* of a technological *operation type*, o_{ik}. The sequence defines the precedence relations between the tasks of the same order. Tasks of different orders are, however, unrelated.

The shop decides whether to accept or reject an incoming order. If an order is rejected then it disappears without any further effect. Accepted orders become *jobs*. A job j inherits all the specifications of its corresponding order; hence it has a task sequence, arrival and release time, due date, contract price and tardiness penalty weight (for notational conventions, see Table 1).

term	**variable**
arrival time	a
release time	r
due date	d
completion time	c
processing time	p
contract price	CP
tardiness penalty weight	W
weighted tardiness	WT
technological cost ratio	T
technological cost	TC
mark-up	U
payment	P
profit	R
operation type	o
volume	v
processing speed	s

(a) Variables.

object	**index**
order	i
job	j
task	k
machine	m
machining capability	n

(b) Indices.

$a_x \leq r_x < d_x$;
$c_x = r_x + p_x$;
$WT_x = [c_x - d_x]^+ \times W_x$,
where $[X]^+ = X$ if $X > 0$, and 0 otherwise;
$P_x = CP_x - WT_x$;
$R_x = CP_x - WT_x - TC_x$;

(c) Relations: x stands for i, j or k, as appropriate.

Table 1: Notation.

The completion time of order i is denoted by c_i. Customers pay only upon completion: the payment for order i, is the contract price minus the eventual tardiness penalty. This weighted tardiness penalty is calculated as follows: $WT_i = [c_i - d_i]^+ \times W_i$, where $[x]^+ = x$ if $x > 0$, and 0 otherwise. Hence, the P_i payment received upon completion of i equals $CP_i - WT_i$. Note that occasionally this payment may be negative.

2.2 Resources

On the resources' side there is a set of multiple-capability *machines* that are able to perform distinct *types of operations* at various *speeds* and *technological cost ratios*, denoted by the fixed parameters o, s, and T, respectively. A particular *machining capability* n of *machine* m ($m = 1, ..., M$) is given by the (o_{mn}, s_{mn}, T_{mn}) triplet. Each machine may have several distinct capabilities. Typically, the same type of operation can be performed at distinct speeds and costs. The *relative cost* (determined by the T/s ratio) shows the cost of a unit amount of work; operation at higher speeds is more expensive in any case (for an example, see Table 2). Machines will calculate their own contract prices based on particular

technological costs (TC) and machine-dependent *mark-up* (U_m). Mark-up determines the amount added to the cost to calculate the contract price.

operation type	**speed** (volume/time)	**technological cost ratio** (currency/time)	**relative cost** (currency/volume)
milling	30	140	4.67
milling	20	60	3.00
drilling	55	100	1.82
drilling	15	15	1.00
boring	60	100	1.67
boring	30	30	1.00

Table 2: Resource capabilities of a typical machine.

In the job shop, the capabilities of machines may partially overlap. The shop is open, i.e., machines may enter or leave it.

2.3 Processing jobs and tasks

Each task requires performing the given volume of work by using one of the appropriate technological capabilities of the machines. For instance, suppose a task k with $o_k = drilling, v_k = 80$ and a machine with a matching resource capability $(drilling, 20, 60)$. Accomplishing this task will take $80 : 20 = 4$ time units and the *technological cost* will amount to $4 \times 60 = 240$ monetary units.

Generally, a machine m can execute a certain task k of job j, if it has the capability n with the required operation type; i.e., $o_{jk} = o_{mn}$. The technological cost of performing the task in this way, TC_{jkmn} is $v_{jk}/s_{mn} \times T_{mn}$. The technological cost of the complete job, TC_j, is calculated as the sum of the technological costs of all of its tasks. Due to the partially overlapping resource alternatives, the same job can usually be performed with different routings, and, consequently, at different technological costs.

Net *profit* on a job, R_j, results from the payment received for the completed job minus the technological costs: $R_j = (CP_j - WT_j) - TC_j$.

2.4 Statement of the dynamic order processing and scheduling problem

Given the dynamic order stream and the available machining resources, the objective is to select and process orders so that the *average total profit* earned in the *long run* be maximal. Hence, $\sum_{i=1}^{I} R_i/\Delta t$ should be maximal, where orders $i = 1, ..., I$ arrive during time interval Δt, and

$$R_i = \begin{cases} CP_i - WT_i - TC_i & \text{if order } i \text{ has been accepted,} \\ 0 & \text{otherwise.} \end{cases}$$

The solutions must satisfy the following basic technological constraints:

- A machine can process a task only if it is able to provide the required type of operation.
- A machine can process one task at a time.
- The tasks are atomic (cannot be shared, pre-empted, or abandoned), and transportation and setup times are included into their processing times.

- The tasks of a job must be processed in their specified sequence, and only one task can be processed at a time.

Note that the problem is dynamic: no complete specification of the order stream is known ahead. Additionally, machines are not supposed to be invariable either: processing speeds may change, and/or certain capabilities (or even full machines) may fall temporally out of production. Hence, certain scheduling decisions must be made on the fly, considering the actual state and the history of the system.

3 OVERVIEW OF THE MULTI-AGENT MODEL

3.1 Agents on the market

As a solution alternative to the above problem, we have developed a multi-agent organizational model. There are two kinds of agents defined:

- The *management agent* interacts with the consumers: receives the incoming order stream, selects orders, and makes the administration and accounting of the accepted orders.

- Machines are represented by individual *machine agents*. They make their own schedules and decide which tasks to perform. Each machine agent communicates only with the management. There are several machine agents, each controlling resources with different, though partly overlapping technological capabilities.

This is a special decomposition of the production scheduling problem: the management agent decides about the customers' orders but does not have processing capabilities, whereas machine agents have control over their own resources but contact neither the customers nor each other. Hence, they mutually depend on each other: management is in need of particular services, while the machines are in need of requests. Machine agents can decide which task to perform, when, and how (at what processing speed and price). Management has to convince them to complete the accepted orders in a way that, in the long run, maximizes the profit of the *whole* system.

Hence, management pays for services, just in the same way it is paid by the customers. All the agents seek their own profits. The profit of management is the payment received from the customers minus the sum of payments given to the machines for working on the tasks of the orders. The profit of the machines is the difference between payments received from the management and their actual technological costs.

The agents are autonomous and do not know about each others' decision mechanisms. They have to focus on their own decision problems, which are hard enough themselves:

- Management agent (i) processes orders, (ii) sets internal due dates, (iii) negotiates with the machines and (iv) makes tentative resource assignments.

- Machine agents (i) negotiate with management, do (ii) one-machine advance scheduling and (iii) final dispatching.

The agents coordinate their decisions via a *market mechanism*. Agents on the market, in general, are heterogeneous and competitive, act with limited information and have bounded reasoning capacities. Negotiation is made on common terms of goods (services), dates and prices. Hence the market, seen as an information forum, enables communication in

rather simple terms. If the market is regulated by some straightforward rules and no single agent can have overwhelming power, then, according to the everyday experience, repeated encounters lead to *adaptive* behavior with prices proportional to the costs.

The market enables also the integration of an *incentive mechanism* that drives the system as a whole toward some preferred states [13]. In our case, such states are those that satisfy the system-wide scheduling constraints and result – in the long run – as much profit as possible. This can be accomplished, even if the agents do not know about system-wide constraints and global performance criteria.

3.2 The negotiation and execution protocol

The basic cycle of our *negotiation mechanism* follows the principle of the Contract Net Protocol [28], where agents in need of a service distribute requests for proposals to other agents who, in turn, evaluate the requests and send back bids to the requesting agent. Bids are then used to decide whom to contract with. However, in our mechanism the *commitment scheme* is flexible: (i) there are *parallel assignments* for the same task, and (ii) contracts are *options*; i.e., give only the right to perform a particular task.

First, management decides alone whether to accept or reject an order. It does not consult with the machine agents and does not consider detailed shop information either.

Then, management *announces* iteratively the first unprocessed task of a job. As a response, machine agents with matching resources prepare local *advance schedules* and submit *bids*, if appropriate. Each machine agent may send no, one or several bids to the management. Now, management compares the bids received, accepts *some of them*, and sends these, so-called *assignments* to the machine agents. If no bid arrives, the management reannounces the task with different parameters.

The actual processing of tasks is initiated by the machine agents: after completing a task, a machine has to select from among its valid assignments. Hence, machine agents perform also *dispatching*. Whenever a task has been selected by a machine agent, the subsequent task (if any) of the same job is announced by management.

Figure 1 shows the basic negotiation and execution mechanism. A detailed description of each activity is given throughout the subsequent sections.

Management monitors the progress of each job. Whenever it finds that a job might be late because of a pending task, management starts to operate a special incentive mechanism. By applying *in-process tardiness penalty*, management forces machine agents to select – rather sooner than later – the pending task that hinders the job's progress. Figure 2.a. shows how the protocol handles the in-process tardiness penalty mechanism (more details are given later).

Bidding for blocked tasks should be re-iterated. A task is *blocked* when (i) no bid was sent for it, or (ii) there remained no valid assignment for doing it. In such cases management evaluates the progress of the whole job and reannounces the blocked task. This is shown at Figure 2.b.

The communication is basically *asynchronous*, albeit there is an upper bound on the response and message transmission times.

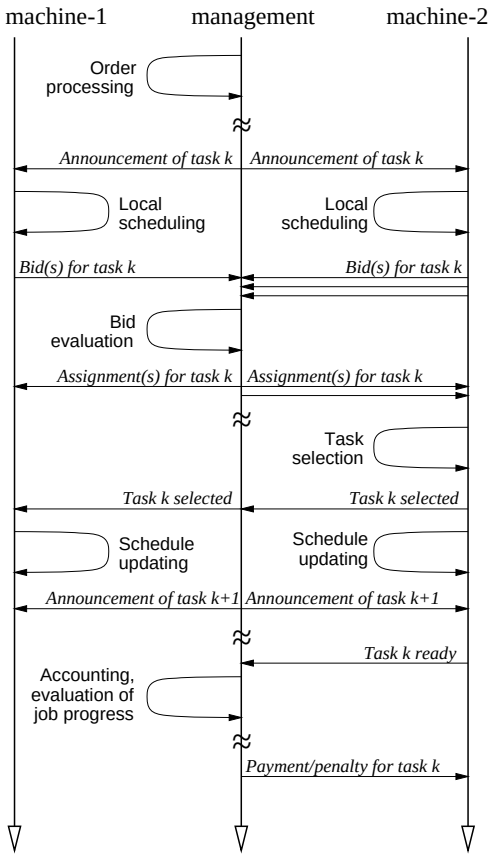
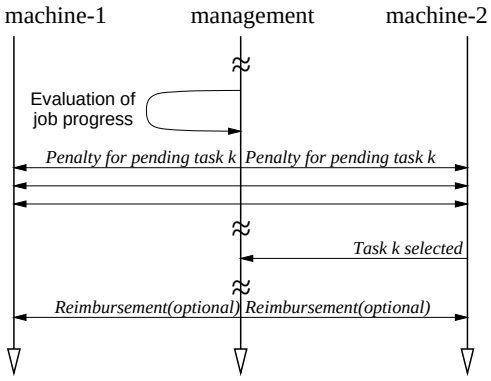
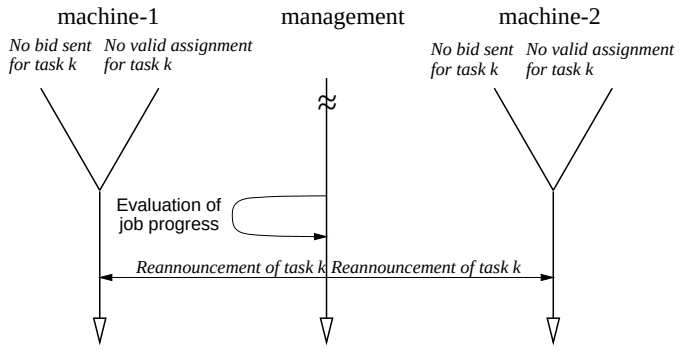


Figure 1: The basic negotiation and execution protocol.



(a) In-process tardiness penalty.



(b) Reannouncement of blocked tasks.

Figure 2: Special cases of the protocol.

4 ACTIVITIES OF THE MANAGEMENT AGENT

Management performs *order processing*, *evaluates* job progress, makes task *announcement* and *assignment*, as well as operates an *in-process tardiness mechanism*.

4.1 Order processing

When an order arrives, Management decides whether to accept or reject it. This decision is based on the estimated processing time and contract price of the new order. Depending upon the situation and availability of historical data, management evaluates a new order either by assuming infinite capacity or by comparing it to aggregate workload information. Hence, estimated processing time $\widehat{p}_{ik}$ and contract price $\widehat{CP}_{ik}$ is calculated separately for each task k of order i by applying one of the following methods:

- In the warm-up phase only the applicable resource capabilities and volume v_{ik} are considered. Management calculates $\widehat{p}_{ik}$ and $\widehat{CP}_{ik}$ by using average processing speeds and technological costs of the available machines.
- Later on, during the normal operation of the system, management uses historical data. For each task k, departing from the volume v_{ik} and the already known processing times and contract prices of the last N task instances of the same operation type, $\widehat{p}_{ik}$ and $\widehat{CP}_{ik}$ are estimated by *linear regression*[24].

The order is accepted only if it seems to be profitable for the management and deliverable on its due time: i.e., if $CP_i \geq \sum_{k=1}^{K} \widehat{CP}_{ik}$ and $d_i - r_i \geq \sum_{k=1}^{K} \widehat{p}_{ik}$.

4.2 Tentative scheduling and evaluation of job progress

Management turns an accepted order into a job, builds a *tentative schedule* for the tasks of the job, updates this schedule and evaluates job progress time and again. Initially, a tentative *time window* and *contract price* is set for each task by using the results of order evaluation. The time window and the contract price give the current constraints for performing a task. The time window specifies the r_k *release time* (i.e, earliest start time) and the d_k *due time* of the task. The contract price, CP_k equals to $\widehat{CP}_k$, as calculated during order processing. After the initial setting the time window of the tasks is relaxed as the specific release and due times of the job allow.

Management attempts to keep the tentative schedule and organize work accordingly. However, since there is no guarantee that the machines will really want to follow this schedule, the actual schedule may deviate from the planned one. Hence, every time before announcing a new task, management *updates* the tentative schedule of the remaining segment of the job. The schedule is updated just in the same way as it has been constructed, though, due to the shifting time-horizon, by using a smaller and more recent dataset.

4.3 Task announcement and reannouncement

Management negotiates with the machine agents over the tasks to be performed. Whenever a machine starts to work on a particular task of job j, management announces the subsequent, still unprocessed task k of job j.

Announcements are sent to all machine agents that are capable to provide the required operation type o_{jk}. Beyond the type and volume of operation, the announcement sets a time window and a maximal price for performing task k. Hence, each announcement consists of the following 5-tuple: $(o_{jk}, v_{jk}, r_{jk}, d_{jk}, CP_{jk})$. For instance, $(drilling, 1000, 305, 350, 1800)$ means that 1000 units of drilling operation are to be performed, the earliest start time is 305, the latest finish time is 350, and management would pay no more than 1800 monetary units for this work.

The values of o_{jk} and v_{jk} are given in the task specification, r_{jk} is set to $c_{j(k-1)}$, the completion time of the preceding task, while d_{jk} and CP_{jk} come from the updated tentative schedule. Note that $c_{j(k-1)}$ can be predicted well since task $k-1$ is just under processing. Since jobs exist for the management only, announcements do not contain any job-related information.

There are two situations when a task gets blocked:

- when no machine agent sends any bids as a response, or
- when all alternative assignments of a task have been lost (see explanation later).

In any of the above cases, the task announcement has to be repeated. Hence, management updates the tentative schedule of the job and *reannounces* the task with a readjusted time window and contract price limit. Since reannouncement is bound to a physical event on the shop floor, reannouncements may not fall into an infinite cycle.

4.4 Bid selection and task assignment

Management selects a *subset* of the bids regarding the same task and notifies the machines which of their bids have been accepted. Accepted bids become *assignments* of the machines. Management may accept several bids; usually there are alternative, *parallel assignments* for each task. The machines do not know about the assignments of each other.

Bid selection has a single rule: management must not accept any dominated bid. A bid is *dominated* if it ends later and costs more than another bid. Favoring any of the machines by accepting its dominated bid would threat the interest of the whole system – this is the reason why dominated bids have to be discarded.

4.5 The in-process tardiness mechanism

Lateness of jobs is controlled by a *proactive incentive mechanism*: whenever the remaining segment of a job j seems to run out of its d_j deadline, management starts to collect penalty in advance from *all* the machines that are technically capable of processing the next – still pending – task of the job. Hence, this is a collective penalty. The penalty mechanism does not create new alternative schedules directly, but it helps to select the next task in a cooperative way. Management stops collecting in-process tardiness penalty when a machine selects a task from a penalized job.

The sum of the collected penalties per time unit is equal to the penalty to be paid per time unit by the management when the job, according to the present expectation, is completed. If the expected final penalty per time unit grows with the lateness of the job, then the collection rate grows as well. If it turns out that the management has collected a sum larger than its actual tardiness penalty, then the machines will be reimbursed.

Penalties and reimbursements are evenly proportioned between the machines. Penalties for different tasks are independent; hence a machine may pay penalties for different tasks at the same time.

The in-process tardiness penalty mechanism belongs to the very core of our incentive system. In this way management fulfills its commitment towards the customers and becomes a transparent mechanism between them and the machines. Since the order selection procedure is fair towards the machines, the overall efficiency of the shop depends greatly on the decisions of the machines. Hence, management should appeal to the collective responsibility of machines.

5 ACTIVITIES OF THE MACHINE AGENTS

Machines do *advance scheduling*, *bidding* and dynamic *dispatching*. All of these activities are related to the creation and maintenance of so-called schedule trees.

5.1 Schedule trees

The promising alternative advance schedules of the machines are represented in their *schedule trees* that are created, updated and known exclusively by their owners. A schedule tree is a dynamic data structure whose updating is prompted by events on the shop floor.

Each node of the schedule tree (except the root) stores an assignment of the machine. The relevant data stored in a node is a triplet of (k, r_k, d_k) which represents task k, its scheduled release and due times, respectively. Branches of the tree represent alternative futures: each branch starting from the root is a *feasible* sequence of assignments. A branch is feasible when all of its assignments can be executed in a certain order. Though, nodes on a branch may be not exactly in their chronological execution order. Considering two nodes on a branch, the deeper node has been inserted into the tree later (for an example, see Figure 3).

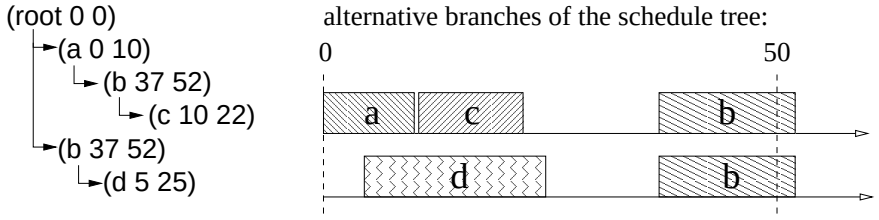


Figure 3: A simple schedule tree and the corresponding schedules.

5.2 Least-commitment scheduling and bidding

Machines respond to an announcement by generating a set of bids. Announcements are valid only in a small, constant time interval, i.e., bids have to be prepared and submitted during that time. Each machine may send more than one bid for an announcement, or may leave the announcement unanswered. Bid preparation must obey the following rules:

- Bids must not hurt the time and financial constraints of the announcement.
- The schedule tree should remain feasible.
- The set of bids prepared by *one* machine should not contain dominated bids.
- Processing of a task should, whenever possible, join the completion of another task.

The first rule constrains the negotiation procedure. The second rule guarantees the feasibility of all local schedule alternatives. The intention behind the last two rules is to avoid combinatorial explosion and to hinder the fragmentation of the machines' time.

Bids are generated by taking into account (i) the announcement of the particular task k given as $(o_k, v_k, r_k, d_k, CP_k)$, (ii) the m machine's capabilities, and (iii) its schedule tree. Each machine, as anyone else in the system, wants to make profit; i.e., to get more for

performing a task than its actual technological cost. Hence, each machine has a local U_m *mark-up* for calculating its particular contract prices. Figure 4 presents the main steps of the bid formation process.

1. Generate processing alternatives for announcement of task k:

   For each n machining capability of m where $o_k = o_{mn}$, calculate processing time and technological cost: $p_{kmn} = v_k/s_{mn}$ and $TC_{kmn} = p_{kmn} \times T_{mn}$. Then, set the contract price, CP_{kmn} to $TC_{kmn} \times (1 + U_m)$.

2. Filter processing alternatives that are too late or too expensive:

   Delete each processing alternative where $r_k + p_{kmn} > d_k$ or $CP_{kmn} > CP_k$.

3. Generate potential assignments for k:

   Try to insert each processing alternative of k into the schedule tree in all possible ways.

4. Generate bids from the potential assignments:

   Get the values of r_{kmn} and d_{kmn} from each potential assignment, and calculate the contract price, CP_{kmn}.

5. Discard all dominated bids, and prune the schedule tree accordingly.

6. Send the remaining bids to management as a response to the announcement of task k.

Figure 4: Process of bid formation.

Bid formation is based on making alternative advance schedules (see step 3 at Figure 4). With other words, machines do *least-commitment* scheduling, as far as the bidding rules allow it. Figure 5 shows the process of local scheduling. Inputs of this process are the (i) schedule tree and (ii) the processing variants of a particular task k. The output is a new schedule tree with all the previous and some new, still tentative task assignments.

1. For each processing variant of task k and for each branch of the tree, do as follows:
   (a) Consider the $[r_k, d_k]$ time window.
   (b) If $[r_k, d_k]$ is fully occupied, no assignment can be put on this branch.
   (c) If $[r_k, d_k]$ is empty, then let $r_{kmn} = r_k$, $d_{kmn} = r_k + p_{kmn}$.
   (d) Otherwise, get the earliest free time slot within $[r_k, d_k]$ that is long enough to contain task k with duration p_{kmn}. This window will start when a task l is due; hence let $r_{kmn} = d_l$, $d_{kmn} = d_l + p_{kmn}$.
2. If no new potential assignment could be made, create a new branch immediately below the root: let $r_{kmn} = r_k$, and $d_{kmn} = r_k + p_{kmn}$.

Figure 5: Generating schedule variants and potential assignments

As for an example, consider the extension of the schedule tree presented at Figure 3. The new task has two processing variants. This example (Figure 6) shows also that the growth of the tree is controlled by the bidding rules.

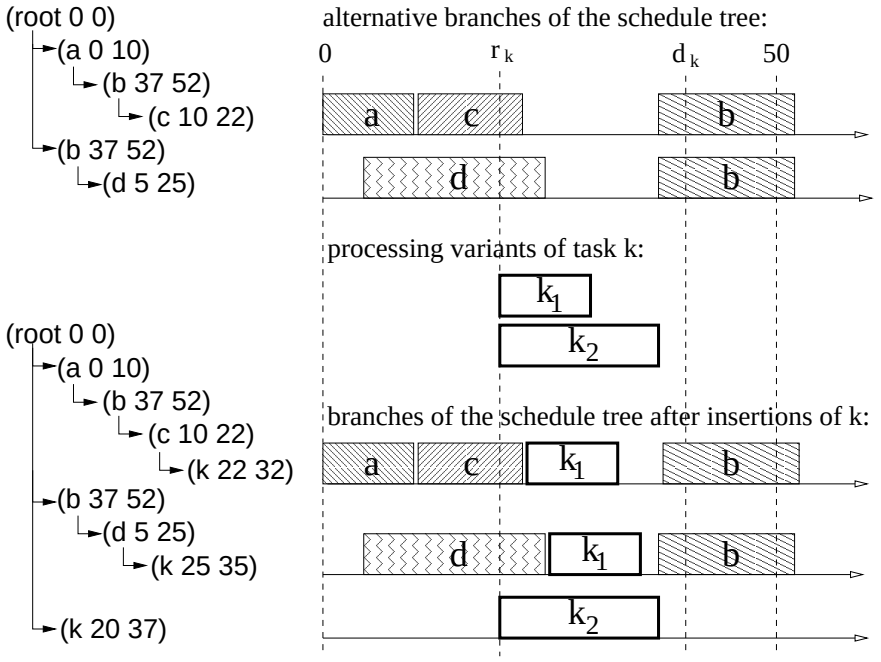


Figure 6: Schedule tree construction ($r_k = 20, d_k = 40$; duration of the two processing variants is 10 and 17).

As a consequence to the bid selection strategy of management, machines keep prices close to the technological costs (it is better to earn a few than nothing). Though, the bidding strategy of a machine can be tailored to specific resources and workloads. It is up to the machines to decide whether they apply any of the following refinements:

- The calculation of the contract price CP_k can be more sophisticated. Typically, a bottleneck machine may charge more for its services, or a task should be charged somewhat more when there remains only a narrow time slot in the schedule (e.g., when a task would take 9 units from a free period of 11 units). Hence, machines may try to charge management for their lost opportunity, too.

- If a machine agent knows that it is more efficient in one of its capabilities (i.e., it is able to earn more profit), then it may bid for the other types of tasks only in times when its expected workload is small.

- In cases when management has started too many jobs in parallel, the basic strategy would create schedule alternatives that will, in all probability, never be realized. In order to avoid this, each machine agent may set a limit on the depth and breadth of its schedule tree.

5.3 Updating schedule trees

The schedule trees need regular *updating*, because as time flies some nodes and branches may become obsolete. Each machine updates its schedule tree any time when

- the release time of a valid assignment has been elapsed, or
- it selects a particular task to work on, or
- a task for which it had valid assignment(s) has been selected by another machine.

When a machine starts to work on an assignment of task k, then (i) it sends a message to management, and (ii) prunes alternative assignments of k and all the colliding assignments of other tasks from its schedule tree. Management in turn (i) prompts the other machines to remove all the assignments of task k from their own schedules, and (ii) announces the next task of the job.

Each machine prunes nodes representing invalidated assignments from its local schedule tree. Hence, updating helps to keep the size of schedule trees limited. Note that updating is prompted always by some physical event in the job shop.

5.4 Final dispatching

The final decision on task execution is in the hands of the machines. Hence, machines perform also *dispatching*. A task assignment gives the machine an *option*: it has the right to do the task under the given conditions, however, it does not necessarily need to do so. The machine can have also other, parallel assignments, and it can decide which one to work on next.

Any time a machine completes a task, it is free to choose to work on an assignment from among those in its schedule tree. However, one rule is in effect here: the machine *must not stay idle* if it has a valid assignment.

The question of selecting a task is rather intricate, since there are a number of dispatching rules, each arguable: (i) choose task with maximal immediate profit, or (ii) with maximal profit per time; (iii) consider the perspectives of the whole tree, by discounting the expected profit of future works; (iv) consider also the opportunity cost of the free time left. In the present version a simple greedy heuristics is applied: machines prefer *maximal immediate profit*. Hence, a machine agent selects an assignment with task k and resource capability n where the value of

$$TC_{kmn} \times U_m - W_m \times p_{kmn}$$

is maximal. W_m is the machine's eventual in-process tardiness penalty: $W_m = 0$ if a penalized, pending task has been selected, and it is the sum of the actual penalties otherwise.

Tie situations – when two or more machines try to choose the same task – are resolved randomly.

6 COMPUTATIONAL EXPERIMENTS

By scheduling with such a nondeterministic and distributed method, the system's worst-case behavior, the avoidance of extreme situations are hard to guarantee. No more than the average behavior can be predicted; the significance of the results can be based solely

on experimentation. Hence, experiments have been carried out with a proof-of-the-concept system implemented in CLOS (Common Lisp Object System).

Two different kinds of experiments were carried out:

- In order to see the reliability and robustness of the system, we tested it against standard scheduling benchmark problems.
- We experimented with various dynamic order streams and resource sets.

6.1 Standard scheduling benchmark problems

In this experimental set-up the system was run *only as a scheduler* on *simple* job-shop scheduling problems. The problems were taken from a problem repository [4]. In order to conform with the standard model, single capacity machines were defined, the performance measure was changed from maximal profit to minimal makespan, and the scheduling problems were transformed into order streams. Orders were given as task sequences as usual. Prices and penalties were set by taking costs proportional to given processing times. As for due dates, we found that for all orders the lower bound of optimal makespan (see [30]) was the most appropriate. This setting resulted in somewhat tight due dates for the orders. Departing from the Fisher and Thompson 6×6 (jobs × machines) instance and one of the Lawrence 10×5 and 15×5 instances, we generated a series of order streams. Then, each *complete* order stream was presented to the system and management was forced to accept all orders.

Lessons learned from these experiments are as follows:

- Given the small problem (6 jobs, 6 machines, 6 tasks per jobs) the known optimum was almost always approached by a 5% margin. This performance degraded with extremely short due date settings only. This phenomena shows that order processing and scheduling must work hand-in-hand – no scheduling can help when the system is overloaded.
- The increase of the number of jobs did not affect – under normal conditions – the performance of the system. However, it had a dramatic effect on the usage of computing resources: local advance scheduling at the machines absorbed computational power and the response times of machine agents could hardly be tolerated. Hence, least-commitment scheduling strategy that keeps alternative futures open is useful only if the problem is really dynamic.

6.2 Experiments with dynamic order streams

When testing the systems with dynamic order streams, we varied the following factors:

- Order streams were generated with different characteristics. Orders in the same stream shared some slowly changing features, such as each order consisted of around 5 tasks, or 10% of them were well-paying rush orders. Hence, demand variations were not turbulent. Table 3 presents the front of an order stream.
- The machines had considerably rich alternative resource capabilities; see Table 2 for a typical machine. Job shops were populated with 5 to 15 machines. The machines had in most of the cases overlapping resource sets. In some experimental set-ups bottleneck machines were applied with unique resources.

i	a_i	r_i	d_i	CP_i	W_i	**sequence of tasks** (k, o_{ik}, v_{ik})
1	3	6	63	3151	55	(1 d 415) (2 m 237) (3 d 404) (4 b 405)
2	6	9	66	3649	64	(1 b 313) (2 b 399) (3 m 385) (4 m 305)
3	9	12	69	2419	42	(1 b (248) (2 b 393) (3 d 304) (4 d 410)
4	12	15	72	3248	57	(1 m 232) (2 d 409) (3 b 392) (4 b 258)
5	15	18	75	2257	40	(1 d 240) (2 d 403) (3 b 235) (4 b 414)
6	18	21	78	3392	60	(1 m 396) (2 b 385) (3 b 233) (4 d 392)
7	21	24	81	3334	58	(1 b 231) (2 m 295) (3 d 237) (4 m 415)
8	24	27	84	2373	42	(1 d 306) (2 m 285) (3 b 248) (4 d 310)
9	27	30	87	2803	49	(1 b 413) (2 m 267) (3 m 244) (4 d 237)
10	30	33	90	3367	59	(1 b 235) (2 b 287) (3 m 318) (4 b 295)
11	33	36	93	2891	51	(1 m 383) (2 d 239) (3 b 408) (4 d 402)
12	36	39	96	3011	53	(1 b 249) (2 d 266) (3 m 304) (4 m 416)
13	39	42	99	3874	68	(1 b 415) (2 b 403) (3 b 393) (4 m 238)
14	42	45	102	2856	50	(1 d 301) (2 m 309) (3 d 309) (4 m 294)
15	45	48	105	3948	69	(1 m 384) (2 m 409) (3 d 399) (4 m 306)

Table 3: The front of a dynamic order stream.

The summarized lessons of the experiments with dynamic order streams are as follows:

- Following their local objectives, management tried to load the system as far as possible, while machine agents attempted to fill in their local schedules by bidding for tasks and by utilizing equally their fast and slow resources in an opportunistic way. The results were schedules without almost any idle time. Figure 7 shows such a typical schedule.

- Since the global goal was not minimizing total lateness, but earning as much profit as possible, management sometimes accepted a well-paying order that elbowed some competing jobs from the machines. This might cause some jobs to be late; nevertheless it never led to lateness of several pending jobs, known as "domino effect" in scheduling.

- Upon the arrival of a new order, management prepared first rather inaccurate time and monetary estimates for the tasks. This caused no problem until the system's load was moderate, because these estimates were refined in the course of the bidding process, and management could also correct them on the fly. In fact, it was often the case. Under heavy load, however, the accuracy of estimates did really matter, since too optimistic estimates delayed the activation of the penalty mechanism. Fortunately, the penalty mechanism used to press the final tasks of the late jobs, and estimates for such tasks were already much better.

- Under moderate load, rational behavior of machine agents caused unbalanced distribution of tasks (and, consequently, of profits): machines that could provide services just slightly cheaper than the others had densely stuffed schedules, while the competitors remained idle.

- When working close to congestion, the in-process tardiness penalty often persuaded machine agents to select from among the pending tasks that seemed to be late. However, by varying the level of penalties it became apparent that above a certain level penalty distorts decision alternatives of the machine agents and deteriorates the overall system performance.

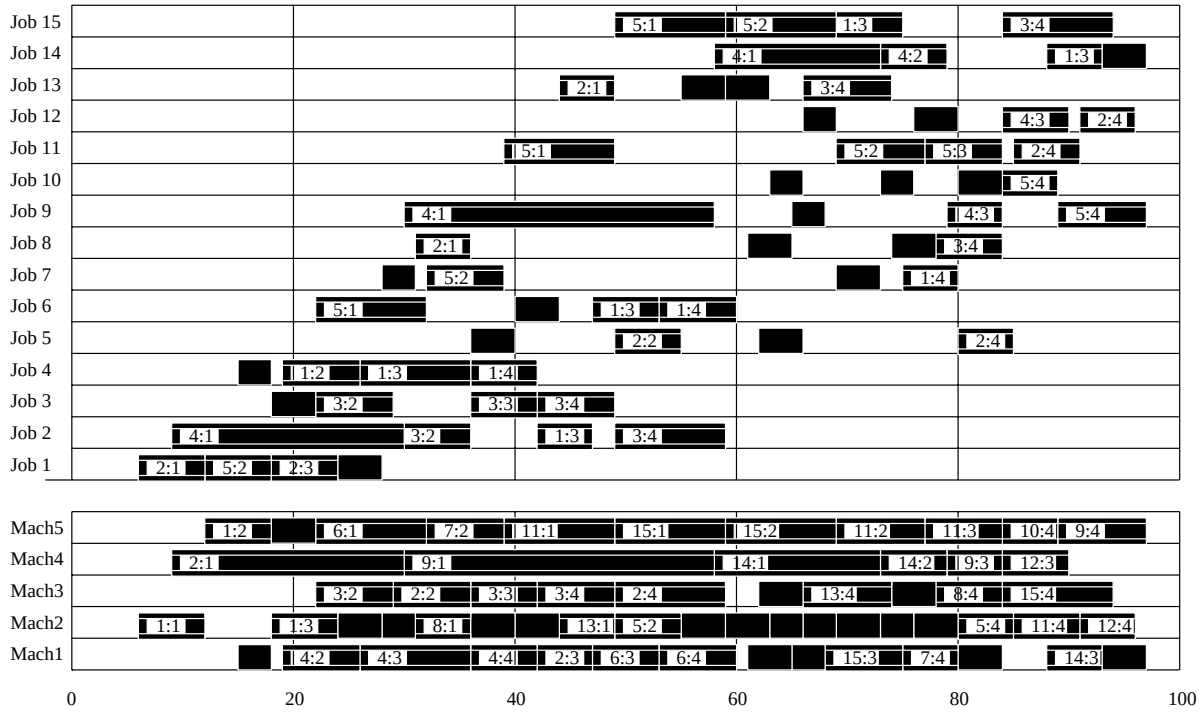


Figure 7: 5 machines schedule of the order stream presented in Table 3. Each order was accepted. Labels in the jobs' lines show machine and task indices, respectively. Labels in the machines' lines correspond to job and task indices. Too short tasks are not labeled.

- Bottleneck machines caused no trouble: after the warm-up phase management could estimate accurately enough their processing times and prices, and after some eventual initial peaks it did not overload the machines in the long run. As bottleneck machines worked more and more, the accuracy of management's estimates improved in time.

7 DISCUSSION

The idea of negotiated distributed scheduling appeared long before this work: early attempts applied various dispatching rules [15, 22, 27] but did not concern advance scheduling. Conversely, [1] realized a predictive scheduler with no reactive capabilities. It is, however, a precursor of our work since it applied market principles when negotiating about due dates and costs. In the last decade several investigations applied negotiation – and the contract net protocol in particular – to solve some form of the distributed production scheduling problem. A thorough overview can be found in [2]; here we can refer only to some later developments [14, 18, 31, 32].

What makes our work different from other approaches is that it integrates order processing, (one-machine) advance scheduling and dynamic dispatching. Order processing – since its controls congestion – sets favorable working conditions for scheduling. Machines with optimizing ambitions plan their future in a least-commitment manner. They make also the final decisions when selecting from a set of eligible tasks. This guarantees that the technological constraints are not violated. Hence, there is no need of tinkering or re-generating global schedules, a quite common practice in distributed scheduling [8, 10]. When select-

ing from among the orders and tasks both management and machine agents may behave opportunistically by preferring well-paying items.

The scheduling method as a whole is opportunistic also in another sense of the word: detailed scheduling decisions are made concerning the most urgent tasks only – those that are in front of the jobs. Some kind of opportunism is inevitable for efficient scheduling, even in static environments [5]; otherwise it would be hard to cope with the inherent complexity of the problems. Successful scheduling methods, like the Shifting Bottleneck and its descendants [3] first look and optimize for a bottleneck resource, and then tailor repeatedly the remaining part of the schedule around the fixed schedule(s). As argued in [21] and [29], dynamic scheduling calls for an opportunistic approach, because the tightly coupled nature of the problem makes it hard to predict the disruptive effects of schedule modifications on the rest of the schedule.

We focused on tasks that are close to the execution just because of the dynamic nature of the problem. This policy resulted in a decision mechanism (i) where the hardest problems – advance least commitment scheduling at the machines – can be solved in a parallel way, and (ii) that works with few iterations and no backtracking. However, the method is applicable only if the scheduling problem is really dynamic. Without further developments it can hardly work on problems where the number of *open* jobs is considerably larger then the number of machines. This is just on the contrary to classical approaches that regard problems with many more jobs than machines as easy [3]. Recently, [33] presented a flexible decomposition scheme that can cover the spectrum between dynamic and static scheduling methods.

Our work is closely related to the developments in the field of *holonic manufacturing systems* [16]. Holonic manufacturing is directly based on the idea of autonomous, and the same time, cooperative agents [7, 6, 10]. One feature that distinguishes holonic manufacturing from other distributed, network-like manufacturing organizations is that it allows temporary hierarchical structures. Hierarchies, or other means of central control (like staff holons, or mediators) organize coordination and facilitate optimization. In our framework, the management agent plays undoubtedly a special role. It has a global view of the system and via order processing strongly determines its behavior. Moreover, the in-process tardiness mechanism, however indirect, is also a means of global control. Under stable operation conditions it remains almost idle; however, as the order stream changes and/or the load increases, its role becomes more and more important.

8 CONCLUSIONS

Rational behavior of the agents leads to hard problems in realizing a manufacturing system where agents have limited information, should coordinate and even cooperate, and make decisions in due time. However, instead of rejecting the assumption of bounded rationality, we have to try to exploit the apparent disadvantages: incomplete and uncertain knowledge, selfish drive and bounded computational resources. The paper shows that in the case of a dynamic production scheduling problem this twist is possible indeed. The requirements for autonomy and cooperation – together with their implications – can be reconciled by means of a market mechanism, where a pressure toward consistent and close-to-optimal global behavior is exercised by an appropriate incentive system.

Acknowledgment

This research was supported by grant T 023305 of the National Research Foundation of Hungary. J. Váncza was also supported by the Bolyai Research Fellowship, while A. Márkus by the "Methodology of Emergent Synthesis" JSPS project (grant no. 96P00702).

References

[1] A. D. Baker. Case study results with the market-driven contract net production planning and control system. In *Proc. of the AUTOFACT '92 Conf.*, pages 31:17–31:35, Detroit, MI, 1992. SME.

[2] A. D. Baker. A survey of factory control algorithms that can be implemented in a multi-agent heterarchy: Dispatching, scheduling, and pull. *Journal of Manufacturing Systems*, 37(4):297–320, 1998.

[3] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.

[4] J. E. Beasley. OR-Library, 1997. Imperial College Management School, http://mscmga.ms.ic.ac.uk/info.html.

[5] J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operations Research*, 93:1–33, 1996.

[6] H. Van Brussel, L. Bongaerts, J. Wyns, P. Valckenaers, and T. Van Ginderachter. A conceptual framework for holonic manufacturing: Identification of manufacturing holons. *Journal of Manufacturing Systems*, 18(1):35–52, 1999.

[7] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: PROSA. *Journal of Manufacturing Systems*, 37:255–274, 1998.

[8] N. A. Duffie and V. V. Prabhu. Real-time distributed scheduling of heterarchical manufacturing systems. *Journal of Manufacturing Systems*, 13(2):94–107, 1994.

[9] E. H. Durfee. Practically coordinating. *AI Magazine*, pages 99–116, 1999.

[10] L. Gou, P. B. Luh, and Y. Kyoya. Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. *Computers in Industry*, 37:213–231, 1998.

[11] J. Hatvany. Intelligence and cooperation in heterarchic manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 2(2):101–104, 1985.

[12] A. K. Jain, M. Aparicio IV, and M. P. Singh. Agents for process coherence in virtual enterprises. *Communications of the ACM*, 42(3):62–69, 1999.

[13] S. Kraus. An overview of incentive contracting. *Artificial Intelligence*, 83:297–346, 1996.

[14] N. K. C. Krothapalli and A. V. Deshmukh. Design of negotiation protocols for multi-agent manufacturing systems. *International Journal of Production Research*, 37(7):1601–1624, 1999.

[15] J.G. Maley. Managing the flow of intelligent parts. *Robotics and Computer-Integrated Manufacturing*, 4(3–4):525–530, 1988.

[16] A. Márkus, T. Kis, J. Váncza, and L. Monostori. A market approach to holonic manufacturing. *Annals of the CIRP*, 45(1):433–436, 1996.

[17] A. Márkus and J. Váncza. Are manufacturing agents different? In S. Albayrak and S. Bussmann, editors, *Proc. of the European Workshop on Agent-Oriented Systems in Manufacturing*, pages 86–103, Berlin, September 1996. Daimler-Benz AG.

[18] F. Maturana, W. Shen, and D. H. Norrie. MetaMorph: an adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, 37(10):2159–2173, 1999.

[19] K. N. McKay and V. C. S. Wiers. Unifying the theory and practice of production scheduling. *Journal of Manufacturing Systems*, 18(4):241–255, 1999.

[20] D. R. Moodie and P. M. Bobrowski. Due date demand management: negotiating the trade-off between price and delivery. *International Journal of Production Research*, 37(5):997–1021, 1999.

[21] P. S. Ow, S. F. Smith, and A. Thiriez. Reactive plan revision. In *Proc. of the Seventh National Conference on Artificial Intelligence*, pages 77–82, Saint Paul, Minnesota, August 1988. AAAI.

[22] H. Van Dyke Parunak. Distributed artificial intelligence systems. In A. Kusiak, editor, *Artificial Intelligence–Implications for CIM*, pages 225–251. IFS Ltd./Springer Verlag, 1988.

[23] H. Van Dyke Parunak. Industrial and practical applications of DAI. In G. Weiss, editor, *Multiagent Systems*, pages 377–421. MIT Press, 1999.

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1986.

[25] S. Russel and P. Norwig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ., 1995.

[26] T. W. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[27] M. J. Shaw. Dynamic scheduling in cellular manufacturing systems: A framework for networked decision making. *Journal of Manufacturing Systems*, 7(2):83–94, 1988.

[28] R. G. Smith. The contract net protocol: High-level communication and control in distributed problem solving. *IEEE Trans. on Computers*, C-29(12):1104–1113, 1980.

[29] S. F. Smith. OPIS: A methodology and architecture for reactive scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 29–66. Morgan Kaufmann, 1994.

[30] E. Taillard. Benchmarks for basic scheduling problems. *European J. of Operational Research*, 64:278–285, 1993.

[31] A. Tharumarajah and R. Bemelman. Approaches and issues in scheduling a distributed shop-floor environment. *Computers in Industry*, 34:95–109, 1997.

[32] M. M. Tseng, M. Lei, and C. Su. A collaborative control system for mass customization manufacturing. *Annals of the CIRP*, 46(1):373–376, 1997.

[33] S. D. Wu, E.-S. Byeon, and R. H. Storer. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.