Intelligent, real-time scheduling for FMS
V. Simeunović, S. Vrančić
Computer System Department, Institute Mihajlo Pupin, Volgina 15, 11060 Belgrade, Yugoslavia
Email: vlada@lab200.imp.bg.ac.yu, sanja@lab200.imp.bg.ac.yu

Abstract

Production scheduling can be defined as the allocation of available production resources over time to best satisfy some set of criteria. Despite the advances in scheduling theory, many actual scheduling problems are still complex to yield analytic solution. This paper presents the constraints of analytic approaches aimed at optimal solution of FMS (Flexible Manufacturing System), explains reasons for creating intelligent scheduler, and announces one possible solution to the allocation problem.

1 Introduction

Production planning and scheduling has been one of the most difficult and challenging tasks for industrial managers, engineers and technologists [1]. Despite the advances in scheduling theory, many actual scheduling problems are still to complex to yield analytic solution. Analytic approaches, aiming at optimal solutions, proved to apply to only a small subset of scheduling problems, those with highly idealized conditions, and a small number of jobs and machines.

Typical assumptions that were made include

- the tasks to be performed is well known and completely defined
- resources and facilities are entirely specified
- sequences for tasks are well defined
- each machine is continuously available
- there is no perturbation or rework
- there is no preemption
- each operation can be performed by only one machine
- the processing time of successive operations of a particular job might not overlap
each machine can handle, at most, one job at a time.

Although some of the above assumptions might be reasonable, the remaining assumptions might not. Most often, schedules generated with these idealized assumptions are inadequate in reality. In addition, problems dealt with in the operations research literature are often restricted to a small number of jobs and machines. In the real world, the numbers are significantly larger, and standard techniques such as linear programming or branch and bound quickly become computationally impractical.

Therefore, manufacturing management have many expectations from artificial intelligence and its applications to the real-world scheduling problems.

In recent years, the AI community has investigated job shop scheduling (factory scheduling) in depth [2], [3], and various paradigms have been presented. However, most of these research focuses its efforts on heuristics and constraints satisfaction, while little attention has been paid to the need for a real-time, distributed scheduling system. The factory scheduling is typical real-time, dynamic environment, and static solutions might quickly become obsolete. Machine breakdown, supplier delivery, operator availability and continual arrival of new orders are only a few of real-time events, representative of the dynamics of the environment. Scheduling can also occur at many different - from weekly and daily scheduling to real-time reactions to machine breakdowns, shortage of supplies and so on. Therefore, the intelligent scheduler should be both predictive and reactive. As the dynamism of the environment increases, it becomes more difficult to distinguish between predictive and reactive behavior, so the need for real-time system is obvious.

To tackle complex scheduling problems in real-time, using limited hardware resource, can be very difficult. A typical scheduling problem comprises several concurrent and sometimes conflicting goals and the resources available to satisfy these goals. Combining goals and resources often results in exponentially growing number of possible solutions, so it can be difficult or impossible to find exact solutions in polynomial time. The job shop scheduling problem belongs to a group of inherently intractable problems commonly referred to a NP-hard. Where an exhaustive search (brute force search, blind search) fails due to time and memory constraints for all but simplest cases, intelligent problem solvers based on heuristic search restrict the problem space and potentially discover near-optimal schedules much faster.

A job shop is a facility that produces goods according to prespecified process plans, under several domain-dependent and commonsense constraints. The scheduling of orders in a job shop is a multifaceted problem. However, any useful scheduling system needs a consistent and flexible scheme to represent various scheduling problem domains. The generic features of a job shop illustrate the flavor of complicated scheduling situation:

- A job shop can produce several parts
- Each part has one or more alternative process plans
- Each process plan consists of a sequence of operations
• Each operation requires resources (machine, personnel, raw material, and so on)
• Each operation has a certain duration on a certain machine
• Most machines can be set up for a number of different operations
• Each order is for one or more units of the same part
• Several orders are typically active at any given time
• Due-date constraints (order deadlines, committed shipping dates) should be met
• The use of expensive equipment have to be maximized
• Throughput of the system should be maximized
• The workload among the machines should be balanced
• The bottleneck resources must not be idle
• The overall idle time should be minimized
• Other constraints such as machine down times further complicate the scheduling problem.

A typical problem has three distinct search spaces:

• **Order permutations**: The sequence of orders to be processed is one of the most significant items on the search agenda. The size of this search space equals the factorial of the number of orders unless bounded by domain constraints
• **Process plans**: Each part to be manufactured can be a component in a number of different process plans
• **Resources**: Since each plan is inherently related to its allocated resources, it is more convenient to estimate the size of the process plan search space and the resource search space as a whole. The sum of the process plan/resource permutations for a given part determines the number of distinct ways a part can be manufactured. Therefore, the total number of process plan/resource selections for a given problem is the product of alternative production methods for each part that is ordered. For a problem with a few plan alternatives per part and a few resource alternatives per operation, the combined process plan/resource search space can be orders-of-magnitude larger than the order permutation space.

Another important issue is rescheduling and reactive scheduling. An operational scheduling system would never develop schedules from scratch. Instead, it would add incoming orders onto an existing schedule. Also, reactive schedulers must deal effectively with conflicts and problems arising in real-time, i.e. it has to be rather flexible to be applied to dynamic, real-world FMS scheduling problems.

It is desirable to have a scheduling system that is capable of observing the environment from the different objectives and to provide multiple scheduling perspectives on a real-time basis. A blackboard architecture with relatively independent knowledge sources could best serve the need for providing multiple perspectives simultaneously.
It is also desirable to be able to function with incomplete or uncertain information, take into account fuzzy information, such as “high priority”, and find a schedule in the absence of complete information.

2 Scheduler development environment

To develop a scheduler for a flexible manufacturing systems it is necessary to choose a development environment that facilitates fast and efficient modeling of all FMS elements (machines, workpieces, operations, etc.), but also that provides an efficient reasoning mechanism for scheduling problem solving. BEST (Blackboard-based Expert System Toolkit) [4] represents such an environment.

Multiparadigm features of BEST enable a system developer to implement each part of the scheduler using the most suitable programming paradigm, i.e., to implement knowledge sources (corresponding to these parts) using production rules, Prolog predicates, or object-oriented programming paradigm. These knowledge sources can be opportunistically activated depending on the system state, thus enabling the implementation of true real-time scheduler.

As a knowledge representation language we use Prolog/Rex [5], [6]. This hybrid language facilitates convenient representation of the knowledge concerning FMS in a form of concepts, basic knowledge representation data type. Concepts represent objects (machines, workpieces, operations, etc.). They can form arbitrary object hierarchies using built-in relations with the inheritance property (is_a, has_a, instance_of, has_instances), as well as some user-defined relations. BEST enables the creation of temporal relations (has_successors, has_predecessors) to determine the operation sequence. This way, an easy definition of the topology of for the production element, process plan execution, or operations sequencing is enabled.

The information about objects is stored in slots. Each slot describes one object attribute with the corresponding value, and, if needed, it can have an attached demon that can control slot access, evaluate slot values, or activate procedures or rules that describe object’s functionality and behavior. To represent procedural knowledge in Prolog/Rex we use different rules (forward-chaining, backward-chaining, constraint, control, and domain knowledge source activation rules), that enable an efficient modeling of expert’s knowledge.

To enhance the applicability of a resource allocation system, BEST facilitates the implementation of customized user interface. Partial or complete data about the allocation process can be presented to the user, using any graphical system under standard MS Windows environment, BEST is fully compatible with.

3 Proposed solution

To solve efficiently the allocation problem, we must develop a corresponding FMS model first. One possible solution is to represent machines and workpieces using two sets: machines and operations. Under operation we consider a procedure applied on a workpiece using the corresponding machine, while
machine represents an object that can perform different procedures on workpieces. Elements of these two sets are represented using concepts.

Each machine within the system is represented using a concept:

```
defconcept( name,
     ( id, identifier),
     (speed, processing speed),
     (peripherals, [list of peripherals]),
     (further_processing, [machines]),
     (security_margin, allowed utilization margin)
).
```

where name represents the concept name, identifier represents a unique identifier of a machine (e.g. machine1), and processing speed corresponds to a processing speed of a workpiece processing on the given machine.

We have defined the processing speed as a number of basic operations per second. List of peripherals describes the ability of a machine to perform different tasks (e.g. grinding, drilling, etc.). Machines represents a list of machines that can be used for further processing of a workpiece (the identifiers of the machines are listed). Allowed utilization margin is defined in percents as a difference between full utilization (100%) and maximal allowed machine utilization.

Similar to concepts that describe machines, we define concepts that describe operations in FMS. The operation concept has the following structure:

```
defconcept( name,
     (id, identifier),
     (importance, importance),
     (criticality, criticality),
     (reliability, reliability),
     (efficiency, efficiency),
     (has_successors, [successor operation]),
     (has_predecessors, [predecessor operation]),
     (peripherals, [operation task]),
     (op_number, number of operations)
).
```

where name represents the operation name, and identifier the unique operation identifier. Importance, criticality, reliability, and efficiency describe the operation characteristics defined by user as a number within the range between 1 and 100. Successor operation and predecessor operation represent slots that enable the creation of an operation graph (identifiers of the successive and former operations are given). Operation task is in relation with slot (peripherals, [list of peripherals]) defined in a concept that represent one machine, and describes a task to be performed on a workpiece (grinding, control-measuring, etc.). Number of basic operations defines the duration of an operation in terms of basic operation unit.

To describe one FMS it is necessary to give some other general data given by system developer. These data are defined using facts:
deffacts(
  (nop, number of operations),
  (nom, number of machines),
  (allocation_time_limit, maximal allocation time),
  (work_time_limit, maximal work time),
  (eff_weight, efficiency weight),
  (rel_weight, reliability weight),
  (cri_weight, criticality weight),
  (imp_weight, importance weight)
).

where the number of operations describes the actual number of operations within the technological process, while the number of machines represents the actual number of machines (the whole quality check system can be treated as one machine). Maximal allocation time defines time limit in which the technological process must be finished, whereas maximal work time defines a time limit for the allocator. Efficiency weight, reliability weight, criticality weight, and importance weight define the weighting factors used to evaluate the overall priority of each operation.

The intelligent allocation system uses the following heuristics:

- critical path heuristic
- process level heuristic
- overall priority heuristic
- treatment of exclusive and relative demands for the execution of operation on a certain machine heuristic

To increase the efficiency of problem search (problem solving). Use of these heuristics is system-defined [7], but they can be changed, removed, or replaced by more efficient ones if found.

4 Implemented prototype

Using BEST we have implemented a prototype scheduler, that consists of an intelligent part and customized user interface [8]. This customized user interface has the following options:

- **FMS menu** - option New FMS ... is used to create dynamically a file that contains the parameters of a concrete FMS, while Exit option enables quitting the program
- **Scheduler menu** - Load option loads a project (into knowledge base), Start option activates inference engine, while Stop option halts inference engine and presents found solutions if they exist.
- **Output menu** - Text... option presents textual information about solutions found, while Graph... option gives a diagram of allocation duration as a function of a number of found solution.

A detailed report about each solution (given in a separate file), consists of operation sequences on machines, statistics for each machine and operation, as
5 An illustrative example

As an example we have used a simple job shop scheduling problem in the DMB factory [9]. Such a system consists of 5 CNC machines and one measuring machine. On these machines 6 different workpieces are produced. Our problem is to implement a scheduler that will allocate 6 workpieces on 6 machines, where the plan of workpiece processing is given using a sequence of operations (figure 1).

We can notice that 12 operations are defined, and the problem is to allocate these 12 operations on 6 machines. We have assumed that a maximal allocation time is limited to 300 sec, while maximal work time is limited to 7200 sec. Parameters input are entered using several dialog boxes. In figure 2 we can see a dialog box for the input of general FMS data.
To enter data about each machine we use a dialog box in figure 3.

In similar way we can enter the parameters for each operation (figure 4).
The last dialog box is used to define the weight factors (figure 5).

After all FMS parameters are defined, a file containing all these data is created. A knowledge base must be loaded, and inference engine activated, before the allocator generates the solutions which can be presented in the textual (figure 6) or graphical form to the user.
The graphical form (figure 7), helps the user to find a solution with the shortest allocation time.

Moreover, special files are created that contain a complete report about the allocated operations (the allocation of operations on machines in time), parameters of allocated operations, machine utilization, and some statistical data.
about the allocator (number of executed rules, number of examined problem states, etc.).

For the given example, the allocator has found 8 solutions, where the last solution has the minimal allocation time - 114 seconds. This solution corresponds to the optimal solution given in figure 8.

Figure 8: Operation allocation on machines

6 Conclusions

In this paper we have listed the problems that must be solved to implement an intelligent scheduler for FMS purposes. We have counted the disadvantages of analytical approach (idealized conditions, too long time needed to find the solutions, etc.), but also the problems of intelligent allocators. We have described BEST-based developing environment for the implementation of modern allocators and an example prototype that efficiently founds the solutions for some allocation problems.

This prototype was tested on a simplified allocation example. We plan to upgrade our prototype to enable real-time performance with a real FMS.
Keywords: FMS, production, scheduling, artificial intelligence

References


