Distributed Manufacturing Scheduling Using Intelligent Agents

Weiming Shen, *National Research Council Canada*

Manufacturing scheduling is an optimization process that allocates limited manufacturing resources over time among parallel and sequential manufacturing activities. This allocation must obey a set of rules or constraints that reflect the temporal relationships between manufacturing activities and the capacity limitations of a set of shared resources. The allocation also affects a schedule's optimality with respect to criteria such as cost, lateness, or throughput.

The globalization of manufacturing makes such optimization increasingly important. To survive in this competitive market, manufacturing enterprises must increase their productivity and profitability through greater shop floor agility. Agent-based manufacturing scheduling systems are a promising way to provide this optimization.

Manufacturing scheduling's complexity

Scheduling problems are not exclusive to manufacturing systems. Similar situations happen routinely in publishing houses, universities, hospitals, airports, transportation companies, and so on. Scheduling problems are typically NP-hard; that is, finding an optimal solution is impossible without using an essentially enumerative algorithm, and the computation time increases exponentially with the problem size. Manufacturing scheduling is one of the most difficult scheduling problems.

A well-known manufacturing scheduling problem is classical job shop scheduling, which involves a set of jobs and a set of machines. Each machine can handle at most one job at a time. Each job consists of a chain of operations, each of which must be processed during an uninterrupted time period of given length on a given machine. The purpose is to find a best schedule—that is, an allocation of the operations to time intervals on the machines that has the minimum duration required to complete all jobs. The total possible solutions for this problem with n jobs and m machines is $(n!)^m$.

The problem becomes even more complex when it includes other variables.

Additional resources

In a real manufacturing enterprise, production managers or human schedulers must consider all kinds of manufacturing resources, not just jobs and machines. For example, a classical job shop scheduling problem with n jobs, m machines, and k operators could have $((n!)^m)^k$ possible solutions.

Simultaneous planning and scheduling

Traditional approaches separating process planning and scheduling can obtain suboptimal solutions at two separate phases. Global optimization of a manufacturing system is only possible when process planning and scheduling are integrated. However, this makes the scheduling problem much more difficult to solve.

Unforeseen dynamic situations

In a job shop manufacturing environment, things rarely go as expected. Scheduled jobs get canceled and new jobs are inserted. Certain resources become unavailable and additional resources are introduced. A scheduled task takes more or less time than anticipated, and tasks arrive early or late. Other uncertainties include power system failures, machine failures, operator absence, and unavailability of tools and materials. An optimal schedule, generated after considerable effort, might become unacceptable because of unforeseen dynamic situations on the shop floor. If this happens, a new schedule must be generated to restore performance. We call such a rescheduling problem *dynamic scheduling* or *real-time scheduling*.

Manufacturing scheduling solutions

The scheduling problem has received considerable attention because of its highly combinatorial aspects (NP-hard), dynamic nature, and practical interest for industrial applications. Consequently, researchers have proposed many different solutions.

Traditional approaches

Because direct methods are not available for complex scheduling problems, search is the usual strategy to solve them. *Generate-and-test* is the simplest approach. It is often reasonable for simple problems but is not reasonable for large, complex problems, and there is no guarantee that a solution will ever be found. Many local search algorithms are generally applicable and flexible. They require a cost function, a neighborhood function, and an efficient method for searching the neighborhood.

Neighborhood search methods offer *heuristic* refinements to generate-and-test. Heuristic search lets us discover solutions that point us in interesting directions (that is, toward the optimum solution), although these solutions might be "bad" in that they might miss the true optimum. A good heuristic method can reach good (although possibly nonoptimum) solutions to hard problems in reasonable time. Heuristic methods try to replace the exhaustive search with some experience, thus reducing computational difficulty.

Constraint satisfaction is another popular approach; it operates in a space of constraint sets rather than in a solution set space.[1,2] Constraint satisfaction algorithms fall into two groups: search algorithms for finding a solution, and preprocessing algorithms (also called *consistency algorithms*) for reducing futile search space. Widely used algorithms include systematic depth-first tree search algorithms (*backtracking*) and hill-climbing algorithms (*iterative improvement*).

Several approaches exploit search strategies that accept even cost-deteriorating neighbors. These approaches need some kind of learning mechanism. One example is *simulated annealing*, a randomized neighborhood search algorithm. It was inspired by physical annealing, which converts a solid to a pure lattice structure by placing it in a heat bath until it melts, then cooling it down slowly until it solidifies into a low-energy state. Simulated annealing has been successfully applied to many single-objective scheduling problems.

Another example is *Tabu search*. This approach combines deterministic iterative improvements with the possibility of accepting cost-increasing solutions occasionally, to direct the search away from local minimums.

In *genetic algorithms*, learning occurs through a solution selection process. GAs discover superior solutions to global optimization problems adaptively—much like biological evolution—looking for small, local improvements rather than big jumps in a solution space. GA methodology seems to achieve small local climbs (adaptations) without gradient information about the objective function being optimized.

All traditional methods, whether analytical, heuristic, or metaheuristic (including GAs, Tabu search, and simulated annealing), encounter great difficulties when dealing with real situations. This is because they use simplified theoretical models and are essentially centralized—that is, a central computing unit performs all computations. This suggests that traditional approaches are inflexible, expensive, and slow to satisfy real-world scheduling problems.

Agent-based approaches

With agent-based approaches, manufacturing resources (for example, machines, operators, robots, and setup stations) can be treated and represented as intelligent agents. They are connected through a local network (for example, an Ethernet). Unlike traditional manufacturing scheduling systems using a centralized scheduler, an agent-based manufacturing scheduling system supports distributed scheduling such that each agent can locally handle the schedule of its machine, operator, robot, or station. However, the participating agents can collectively perform global scheduling through some negotiation mechanism and protocol (for example, a contract net protocol or an auction protocol, which I describe later).

Agent-based approaches have several potential advantages for manufacturing scheduling:[3,4]

- The agent paradigm uses parallel computation through a large number of processors, which could provide high efficiency and robustness.
- The agent paradigm makes integrating process planning and manufacturing scheduling easy, so as to realize the simultaneous optimization of manufacturing process planning and scheduling.
- You can connect resource agents directly to their represented physical devices, so as to realize real-time dynamic rescheduling. This could provide high fault tolerance.
- Agents can develop schedules using the same mechanisms that businesses use (negotiation rather than simple search) in the manufacturing supply chain. Thus, you can directly connect the manufacturing capabilities of different manufacturing enterprises. This will make optimization possible at the supply chain level, in addition to the shop floor level and the enterprise level.
- Individual resources can trade off local performance to improve global performance, leading to cooperative scheduling.
- You can combine other techniques with agent-based approaches at certain levels for learning and decision-making (I discuss this in more detail later).

Majors issues of agent-based manufacturing scheduling

Anyone developing an agent-based manufacturing scheduling system must deal with four main issues among others: agent encapsulation, coordination and negotiation protocols, system architectures, and decision schemes for individual agents.

Encapsulation

Among the different approaches for agent encapsulation in manufacturing scheduling systems, two are distinct: *functional decomposition* and *physical decomposition*. Functional decomposition uses agents to encapsulate modules assigned to functions such as order acquisition, planning, scheduling, material handling, transportation management, and product distribution. No explicit relationship exists between agents and physical entities. Physical decomposition uses agents to represent entities in the physical world, such as operators, machines, tools, products, parts, and operations. An explicit relationship exists between an agent and a physical entity. Both approaches have distributed (not centralized) implementations.

Functional decomposition tends to share many state variables across different functional agents. This can lead to inconsistency and unintended interactions. Physical decomposition naturally defines distinct sets of state variables that individual agents with limited interactions can manage efficiently. However, it needs a large number of resource-related agents, which can lead to other problems (such as communication overhead) and complex agent management. However, functional decomposition is useful in integrating existing systems (for example, CAD tools, materials requirements planning systems, and databases), so as to resolve legacy problems.

Corresponding to these two agent encapsulation approaches are two types of distributed manufacturing scheduling systems.

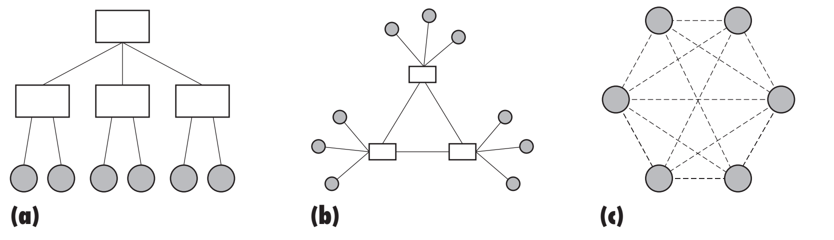

Figure 1. System architectures for agent-based scheduling: (a) hierarchical, (b) federated, and (c) autonomous agent.

In the first, scheduling is an incremental search process that can involve backtracking.[5–7] Agents, responsible for scheduling orders, perform local incremental searches for their orders and might consider multiple resources. The system merges the local schedules to produce a global schedule. This is similar to centralized scheduling.

In the second system, an agent represents a single real-world resource (for example, a work cell, a machine, a tool, or an operator) and maintains this resource's schedule. This agent might negotiate with other agents to carry out overall scheduling. Most agent-based manufacturing scheduling systems use this approach.

Coordination and negotiation protocols

Systems that use functional decomposition are similar to traditional integrated systems; they usually use a predefined coordination mechanism. So, this section primarily discusses systems using physical decomposition.

Most agent-based manufacturing scheduling systems use negotiation protocols for resource allocation. The *contract net protocol*[8] or its modified versions are the most common, although some systems use other protocols such as Kenneth Fordyce and Gerald Sullivan's voting protocol.[7]

Reid Smith first proposed the CNP and demonstrated it on a distributed sensing system.[8] To summarize, each agent (manager) having work to subcontract broadcasts an offer and waits for other agents (contractors) to send their bids. After some delay, the manager retains the best offers and allocates its contracts to one or more contractors, which then process the subtask. The CNP coordinates task allocation, providing dynamic allocation and natural load balancing.

The basic CNP is quite simple and can be efficient. However, when the number of nodes (agents) is large, the number of messages on the network increases. This increase can lead to a situation where agents spend more time processing messages than doing the actual work or, worse, where the system stops because messages have flooded it. To avoid these problems, researchers have proposed various improvements to the basic CNP, such as

- sending offers to a limited number of nodes (agents) instead of broadcasting them;
- anticipating offers; that is, contractors send bids in advance;
- varying the time when commitment is decided;
- allowing decommitment (breaking commitments);
- allowing several agents to answer as a group (coalition formation); and
- introducing priorities for solving tasks.

The negotiation protocol's bidding mechanism can be *part-oriented*,[9] *resource-oriented*,[10,11] or *bidirectional*.[12]

The basic CNP selects a contractor by comparing bids corresponding to a particular offer using predefined criteria. More complex versions introduce penalties, thus bringing the CNP nearer to a *market-based approach*. Several recent agent-based scheduling systems have used such approaches,[9,11] which are becoming increasingly popular. Market-based protocols use a bargaining or auction process, which is simple and easy to use. Some agent-based manufacturing scheduling systems have implemented different auction techniques from real marketplaces.

Katia Sycara and her colleagues proposed a different approach using *texture measures*, where all agents share a common information base, called a *coordination agent*.[5] Each agent computes its own texture measure and submits it to the coordination agent, then reads the integrated texture measure to make a decision. After individual agents make their decisions, they submit their solutions to the coordination agent, which in turn regulates the possible conflicts. While this approach could help agents predict possible conflicts, it would not eliminate conflicts.

Some authors have also realized the game-like nature of independent scheduling decisions and have tried to use game theory to make their agents smarter.[13]

Another solution is to combine the agent-based approach with traditional approaches. I look at this in more detail later.

Architectures

Agent system architectures provide the frameworks within which agents are designed and constructed. Architectures for agent-based manufacturing scheduling systems fall into three categories: *hierarchical*, *federated*, and *autonomous agent* (see Figure 1).

Hierarchical. A typical manufacturing enterprise consists of a number of physically distributed, semiautonomous units, each with some control over local resources and with different information requirements. In this situation, a number of practical agent-based industrial applications still use the hierarchical architecture, although critics often complain about its centralized character. In fact, agent-based distributed manufacturing scheduling systems using functional decomposition usually have a hierarchical architecture because each agent represents a function or a department in a traditional manufacturing system. Examples of such systems include Distributed Asynchronous Scheduling,[6] the Logistics Management System,[7] the Architecture for Distributed Dynamic Manufacturing Scheduling,[10] and Holonic Manufacturing Systems.[14] I describe these in the section "The research literature."

Federated. Because hierarchical architectures suffer serious problems due to centralization, federated multiagent architectures increasingly are considered a compromise solution for industrial agent-based applications, especially for large-scale engineering applications. A fully federated agent-based system has no explicit shared facility for storing active data. Rather, the

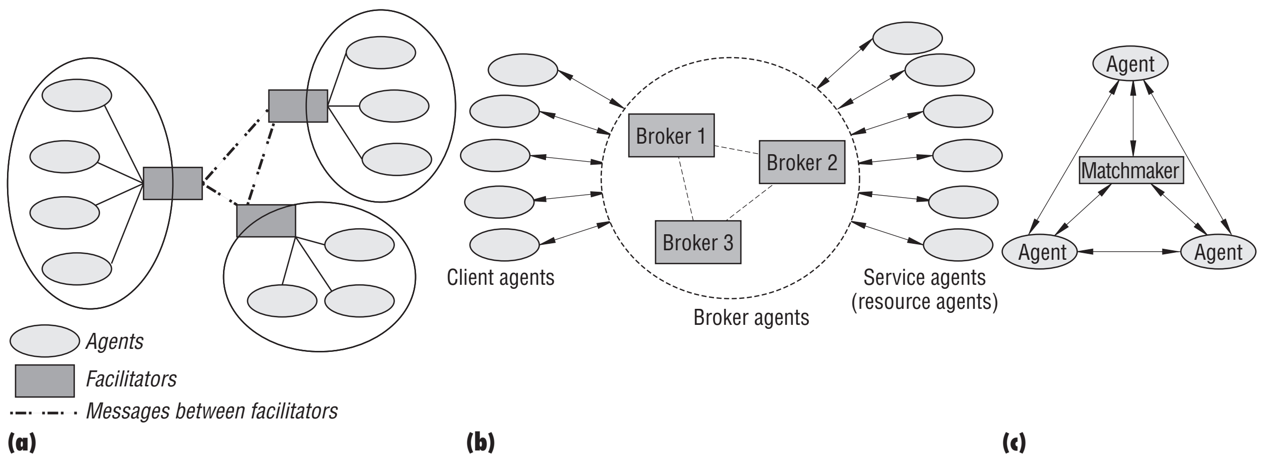

Figure 2. Federated architectures: (a) facilitator, (b) broker, and (c) matchmaker.

system stores all data in local databases and handles updates and changes through message passing.

For agent-based manufacturing systems, four types of federated architectures dominate: *facilitators*, *brokers*, *matchmakers*, and *mediators*. Facilitators (see Figure 2a) combine several related agents into a group. Communication between agents takes place through an interface also called a *facilitator*. Each facilitator provides an intermediary between a local collection of agents and remote agents. It usually does this by providing two main services: routing outgoing messages to the appropriate destinations and translating incoming messages for consumption by its agents. Many agent-based collaborative design systems have used this approach.[4]

Brokers (see Figure 2b), also called *broker agents*, are similar to facilitators, with additional functions such as monitoring and notification. The functional difference between a facilitator and a broker is that a facilitator is responsible for only a designated group of agents, whereas any agent can contact any broker in the same system for finding service agents to complete a special task.

The matchmaker architecture (see Figure 2c) is a superset of the broker architecture, because it uses the brokering mechanism to match agents. The Foundation for Intelligent Physical Agents (www.fipa.org) has proposed *yellow-pages agents* and *directory facilitators*, which are similar to matchmakers.

Besides functioning as a facilitator and a matchmaker, mediators assume the role of system coordinator by promoting cooperation among intelligent agents and learning from the agents' behavior.[3,4] This architecture imposes a static or dynamic hierarchy for every specific task, which provides computational simplicity and manageability. It is suitable for developing distributed manufacturing scheduling systems that are complex and dynamic and that comprise many resource agents.

Federated multiagent architectures can coordinate multiagent activity through facilitation or mediation to reduce overhead, ensure stability, and provide scalability. They promise to be a good foundation on which to develop open, scalable multiagent systems.

Autonomous agent. Different definitions of autonomous agents exist; however, an autonomous agent usually

- is not controlled or managed by any other software agent or human being,
- can communicate and interact directly with other agents in the system and with other external systems,
- has knowledge about other agents and its environment, and
- has its own goals and an associated set of motivations.

The autonomous agent architecture is well suited for developing distributed manufacturing systems consisting of a small number of agents. I describe two such systems in the section "The research literature."

Hybrid. MetaMorph II[3] combines the approaches I've described to develop more flexible, modular, scalable, and dynamic manufacturing systems. By combining the mediator and autonomous agent approaches, MetaMorph II's hybrid architecture lets an agent in one subsystem communicate directly with other subsystems or agents in other subsystems, thereby mitigating bottlenecks.

Decision schemes

Here I address what kind of decision scheme an individual agent should have to realize effective agent-based cooperative scheduling. As in the section on coordination and negotiation, I focus on systems using physical decomposition.

In most agent-based manufacturing scheduling systems using market-based and bidding-based protocols, individual agents need to reply to all kinds of offers. They also sometimes need to compete, negotiate, or bargain with other agents. Rich knowledge and powerful learning and reasoning mechanisms are important. Each agent should have at least knowledge about the capability, availability, and cost of the physical resource (for example, a machine) that it represents. A more sophisticated agent can have knowledge about other agents in its system, knowledge of the products to be manufactured, a knowledge base of previously successful cases, and so on.

An agent's decision scheme depends primarily on two aspects: the coordination or negotiation mechanisms that the multiagent system uses and the agent's local decision-making mechanisms with available knowledge. For example, a CNP needs each agent to reply to an offer with requested information such as cost, start time, processing time, and so on. A game theory-based multiagent system needs agents to follow game rules.[13] A multiagent system implemented with a conversation scheme needs each agent to follow the conversation policies.[4] Local decision making might use different kinds of mechanisms, such as rule- or case-based reasoning mechanisms, according to the agent's knowledge. Updating an agent's knowledge requires a learning mechanism, which can range from being case-based to neural network- or fuzzy logic-based.

The research literature

Over the past two decades, a number of researchers have used agent technology in attempts to resolve the manufacturing scheduling problem. The following paragraphs provide a short research literature review.

Early attempts

Michael Shaw might have been the first to suggest using agents in manufacturing scheduling and factory control. He proposed that a manufacturing cell could subcontract work to other cells through a bidding mechanism.[15] *YAMS* (Yet Another Manufacturing System), another early agent-based manufacturing system, represented each factory and factory component as an agent.[16] Each agent had a collection of plans, representing its capabilities. YAMS used the CNP for interagent negotiation.

Methodologies and techniques

CORTES used microopportunistic techniques to solve the scheduling problem through a two-agent system, where each agent schedules a set of jobs and monitors a set of resources.[5] It was a typical scheduling system using functional decomposition.

Albert Baker proposed a *market-driven contract net* for heterarchical agent-based scheduling.[11] He introduced a market-based mechanism into the contract net-based negotiation and made the manufacturing scheduling system more like other business systems.

The *Logistics Management System* applied integration decision technologies to schedule semiconductor manufacturing.[7] LMS used functional agents, one for each production constraint, and a judge agent to combine the votes of four different critics. Each agent modeled those aspects of the environment needed to satisfy its objective. Its uniqueness was the voting protocol I mentioned earlier.

Jyi Shane Liu and Katia Sycara proposed *Constraint Partition and Coordinated Reaction*, a coordination mechanism for job shop constraint satisfaction. CP&CR assigned each resource to a resource agent and each job to a job agent.[2] A resource agent enforced capacity constraints on the resource; a job agent enforced temporal precedence and release-date constraints within the job. CP&CR was a good example of integrating agent-based approaches with constraint satisfaction.

AARIA (Autonomous Agents at Rock Island Arsenal) encapsulated the manufacturing capabilities (for example, people, machines, and parts) as autonomous agents.[17] Each agent seamlessly interoperated with other agents in and outside its factory. Agents communicate using subject-based addressing; that is, messages are labeled by content rather than by recipient and are forwarded to all agents subscribing to the specified content area.

The increasing use of bidding- or market-based negotiation protocols necessitates research and development of more sophisticated negotiation mechanisms and protocols.

It was one of the few manufacturing scheduling systems that have used autonomous agents.

Kazuo Miyashita proposed an integrated architecture for distributed planning and scheduling combining the repair-based methodology with the constraint-based mechanism of dynamic coalition formation among agents.[1] He implemented the *CAMPS* (*Ca*se-Based *M*ultiagent *P*lanning/*S*cheduling) prototype system, another example of integrating agents with constraint satisfaction. In CAMPS, a set of intelligent agents tried to coordinate their actions for satisfying planning and scheduling results by handling several intra-agent and interagent constraints.

Approaches and architectures

Peter Burke and Patrick Prosser's *Distributed Asynchronous Scheduling* architecture consisted of three types of entities: knowledge resources, agents, and a constraint maintenance system.[6] The knowledge resources contained frame-based knowledge for resources, aggregate resources, operations, process plans, and what they called a *strategic unit*. A strategic unit is the root of a hierarchy (of all entities) and is concerned with the scheduling problem as a whole. Originally, the agents formed a multiagent heterarchy to represent only resources (O-agents). The final architecture had agents for different aggregations of resources (T-agents) and an agent for overseeing scheduling (S-agent). The S-agent assigned operations to T-agents, who assigned them to their respective O-agents. The O-agents then scheduled these operations on their respective resources. DAS could make a correct schedule but had no method for optimizing that schedule.

ADDYMS (Architecture for *D*istributed *D*ynamic *M*anufacturing *S*cheduling) decomposed scheduling into two levels.[10] The first assigned a manufacturing work cell to a task. The second determined local resources, such as operators and tools, that a number of work cells might share. Corresponding to these levels were site and resource agents. A site agent scheduled work at a particular work cell. A resource agent represented each local resource at a work cell. The system comprised several connected site agents, each of which in turn was connected with its subsite agents and some local resource agents.

Grace Lin and James Solberg showed how to use a market-like control model for adaptive resource allocation and distributed scheduling.[9] They modeled the manufacturing shop floor exactly like a marketplace. Each task agent entered the market carrying certain "currency" and bargained with each resource agent on which it could be processed. At the same time, each resource agent competed with other agents to get a more valuable job. They incorporated the market mechanism, using multiple-way and multiple-step negotiation, to coordinate these autonomous agents in the shop floor.

Khalid Kouiss and his colleagues proposed a multiagent architecture for dynamic job shop scheduling.[18] Each agent was dedicated to a work center and performed local dynamic scheduling by selecting an adequate dispatching rule. It selected the most suitable dispatching rules. Depending on local and global considerations, a new selection happened when a predefined event occurred. Opti-

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.