

# Cooperative Multiobjective Decision Support for the Paper Industry

---

Sesh Murthy, Rama Akkiraju,  
Richard Goodwin, Pinar  
Keskinocak, John Rachlin,  
Frederick Wu, James Yeh,  
Robert Fuhrer

*IBM T. J. Watson Research Center, Route 134,  
Yorktown Heights, New York 10598*

Santhosh Kumaran,  
Alok Aggarwal

*IBM Supply Chain Optimization Development*

Martin Sturzenbecker

*IBM Internet Division*

Ranga Jayaraman

*IBM Storage System Division*

Robert Daigle

*IBM Corporation*

---

We built and deployed a decision-support system for scheduling paper manufacturing and distribution, an extremely complex task with multiple stages of production and strong interaction between stages. In contrast to earlier approaches, our system considers multiple scheduling objectives and multiple stages of production and distribution simultaneously using multiple evaluation criteria. Our system functions as an intelligent assistant to the schedulers and generates multiple good scheduling alternatives using a portfolio of algorithms and direct human-expert input. The successful deployment of our system at several paper mills in North America has resulted in significant savings, greater customer satisfaction, and improved business processes.

The manufacture of paper products is a major worldwide industry. The sales volume of paper and allied products in the United States was over \$160 billion in 1997 [Beck et al. 1998]. Paper manufacturing is a capital-intensive business, with equipment and facilities for a new produc-

tion line costing about half a billion US dollars. To compete, paper companies must use their capacity efficiently, respond to customer demand, deliver on time, and provide short lead times. This requires scheduling production to minimize production and distribution costs, inventory

levels, and manufacturing disruptions [Shaw 1998]. The relative importance of these competing objectives varies with the state of the production environment and market conditions. The complexity of the problem is compounded by interactions wherein the scheduling of each stage of the production process affects downstream production or shipping.

In 1993, Madison Paper Inc.'s CEO approached IBM about using artificial intelligence and advanced scheduling techniques to increase production at its mill in Madison, Maine, without having to invest in new manufacturing equipment. At IBM Research, we took up the challenge and began studying the current state of papermill scheduling, looking for opportunities not only to use new scheduling algorithms, but also to improve the information available to the decision makers to improve the quality of their scheduling decisions. As a result, we developed a cooperative multiobjective decision-support system, which is currently used by several mills in North America.

### **Paper Manufacturing**

The paper industry manufactures many kinds of paper to satisfy the diverse needs of printing and packaging. Paper production consists of several stages, which may vary depending on the kind of paper being produced, but the overall process is generally the same (Figure 1).

The first step in paper manufacturing is the production of pulp from logs, wood chips, recycled paper, and other sources of fiber. The pulp is fed into a paper machine along with other ingredients that make up the "recipe" for a particular grade and basis weight of paper, that is, a product. The

grade of paper is determined by physical and optical characteristics, such as smoothness, oil absorbency, gloss, and shade. The basis weight is the weight of a specified area of paper or paperboard. A paper machine produces large reels of paper. The width of the reel, called the *deckle*, is fixed for each machine. Another machine called a *winder* unwinds the reel while slicing it into narrower strips that it then rewinds to form *rolls*. The process of cutting a reel to make rolls is called *trimming* and the portion of the deckle that is not consumed by the rolls is the *trim loss*. The arrangement of the slitting knives on the winder constitutes a *trim pattern*. Typically, several rolls are made from each reel. The widths and diameters of these rolls must match the customer requirements. Detailed plans, which include the selection of patterns and their sequencing, for cutting a set of large reels of paper into smaller rolls are called *trim sheets*. As the rolls are produced, they are wrapped for shipping or temporary storage. In the case of cut-sheet paper products, the rolls are loaded onto a *sheeter*, which unwinds the roll and slices the paper into sheets of the desired size. The sheets are then wrapped and packaged for shipping. (We do not consider cut-sheet production in this paper.) Finally, the end products are loaded onto trucks and rail cars for shipment to customers, warehouses, and ports.

Bierman [1993] gives further details on paper production.

Most paper manufacturers produce to order because the large number of combinations of product type and roll size makes it impractical to stock enough inventory. Each customer order specifies a

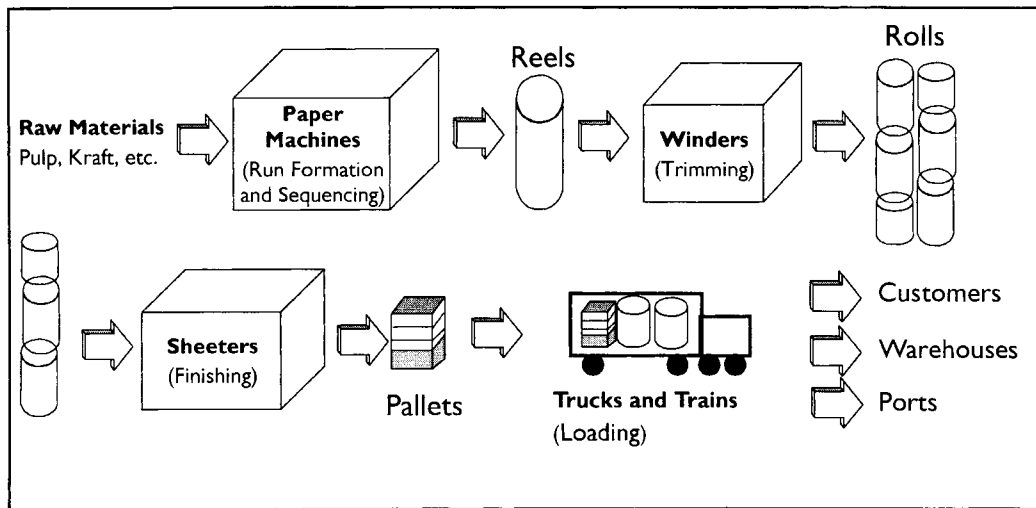


Figure 1: Paper machines use raw materials such as pulp and fillers to produce large rolls of paper (often called reels), which are trimmed and rewound onto rolls. The rolls may be shipped directly to customers, or they may be trimmed into sheets, which are packaged and shipped on pallets.

quantity (in tons or number of rolls), a product type, roll dimensions (width and diameter), due date, and shipping destination. While customers prefer to have their orders filled exactly, there is typically a standard tolerance, for example, plus or minus three percent, on the quantity that can be produced to satisfy an order. To improve efficiency, manufacturers sometimes take advantage of this tolerance and produce more or less than the ordered amount. The amount produced in excess of the order quantity is called *overrun*. Similarly, any production shortfall is called *underrun*. The paper manufacturer is usually responsible for the freight cost from its mill to the customer's location, which can constitute as much as 15 percent of the selling price. To reduce transportation costs, manufacturers try to produce orders in mills close to the orders' final destinations.

A typical large paper-manufacturing en-

terprise has several mills in different locations, each having one or more paper machines. Each paper machine is capable of producing a subset of the company's products at different production rates. Paper production is a continuous process in which a machine can make only one product at a time because each product has its own unique recipe. When a machine switches from one product to another, it continues to operate, but the paper it produces is of poor quality for some time after the change is initiated. The length of this transition time or setup time depends on the products being produced before and after the transition; transitions between similar products are shorter than transitions between dissimilar products (that is, the setup times are sequence dependent). The setup times between products can also be machine dependent. The production between two consecutive setups is called a *run*. Often there is a restric-

tion on the minimum length of a run since very short runs are likely to produce paper with quality problems.

### **Scheduling of Paper Manufacturing**

Given a set of firm orders and a forecasted demand, the schedulers must decide how to allocate orders to mills and machines, sequence the orders on each machine and form a sequence of runs, trim the reels of paper to create rolls of the right size for each order, and load the rolls into vehicles for shipping to the customers and distribution centers in order to maximize profit by minimizing trim loss, transportation costs, and inventory carrying costs; maximize on-time delivery by minimizing both early and late deliveries; minimize manufacturing disruptions; and maximize customer satisfaction by minimizing violations of customer preferences. Most problems in schedule generation are variations of well-known NP-complete problems. For example, run formation and sequencing is an extension of the multimachine job-sequencing problem with sequence-dependent setup times, trimming is an extension of the cutting-stock problem, and loading falls into the domain of the multiple-knapsack problem.

The traditional approach to paper-mill scheduling is to schedule each stage in the process independently. Typically, paper manufacturers allocate orders to paper machines and sequence them manually. They then use one software package for trim scheduling and another for outbound logistics scheduling. Each package focuses on one process step and tries to optimize the schedule based on local objectives. Since these applications do not interact, the complete schedule obtained by com-

binning the subschedules is usually of very low quality. For example, a trim schedule that minimizes trim loss may cause vehicles to be loaded inefficiently, unacceptably increasing shipping costs. To improve the loading schedule, the scheduler must make extensive changes in the trim schedule. This process is time- and labor-intensive since posing what-if scenarios requires moving between applications. To make the process more manageable, companies adopt business rules, such as taking orders only in full vehicle loads and requiring trim schedules to produce exactly the amount ordered. This simplifies scheduling but reduces the company's ability to respond to customer requests, thereby lowering customer satisfaction and reducing efficiency. In addition, most scheduling packages combine multiple objectives into a single objective function and produce a single schedule that attempts to optimize this composite objective function [Pickard 1997]. Since the importance of the underlying objectives may not be precisely known, this approach rarely generates a satisfactory solution [Bennett 1998; Das 1998; Goodwin et al. 1998; Steuer 1989]. Furthermore, presenting schedulers with a single take-it-or-leave-it choice does not show the trade-offs between competing objectives needed to make an informed decision. To explore the solution space, schedulers repeatedly modify the objective function and rerun the scheduling application to see other possible schedules [Yu 1985].

In contrast, our new scheduling system considers all stages of paper production and distribution simultaneously and generates multiple enterprise-wide schedules.

Each schedule contains an allocation of orders to machines, a sequencing of orders on the machines, a trim schedule for each production run, and a loading schedule. The schedules are created by algorithms that take into account the interactions between the process stages and focus on enterprise-wide objectives. The algorithms we developed use such approaches as integer programming, linear programming with and without rounding, network flow, and heuristic methods. By combining multiple approaches and considering interactions between processes, our system improves solution quality compared with earlier approaches. It evaluates each generated schedule in terms of multiple objectives and presents the best schedules to the scheduler. It treats business rules as objectives rather than constraints; therefore, some schedules may violate the rules to improve customer satisfaction or manufacturing efficiency. (These schedules can be shared with manufacturing and customer service representatives and serve as a basis for negotiating exceptions to business rules.) The system also allows schedulers to work with the software to improve schedule quality and generate additional alternatives. By examining these schedules and comparing the alternatives, schedulers gain an understanding of the trade-offs to be made and can select a solution that balances conflicting objectives.

### **Decision-Support Using the Asynchronous Team Architecture**

Our approach to decision support is to augment the skills of the schedulers by offering an intelligent assistant [Reddy 1996] capable of providing the information they

need to make good decisions. The schedulers who use our system are experts and have intimate and extensive knowledge of the production environment, operating procedures, and company policies.

Through years of experience, they have developed heuristics to quickly identify promising schedules. However, these heuristics search only a small part of the solution space and overlook opportunities for alternative and possibly better solutions.

By combining advanced solution techniques with the domain expertise of the schedulers, our system produces high-quality alternative solutions for consideration. We believe that the scheduler's expertise cannot be fully captured in software. As stated by Wagner [1969, page 9], "an operations research model is never sufficient unto itself: it cannot become entirely independent of judgment supplied by knowledgeable managers. . . .The question is not when to apply science and when to rely on intuition, but rather how to combine the two effectively." Our system allows schedulers to create new schedules or modify existing schedules at any level of detail to provide expert guidance. Unlike some earlier approaches [Yu 1985], the iterations in the solution-generation process are not based on changing the objective weights but on more detailed input from schedulers about the solutions. The schedules generated by schedulers, like those generated by the system, serve as a basis for further search and refinement by our system. This contrasts with many other systems, where once a scheduler edits a solution, the system cannot further improve it. There are cooperative expert systems, such as

Scheiker [Numao and Morishita 1991], where the cooperation between the scheduler and the system is similar to our approach; however, the rules that these systems employ are far less capable of making useful improvements to scheduler-generated solutions than the direct-improvement methods we employ.

---

### The traditional approach is to schedule each stage independently.

---

To summarize, the components of our decision-support approach include generating multiple alternative solutions, evaluating the alternatives in business terms, selecting a small subset of these alternatives and presenting them to the decision maker, collaborating with the decision maker to generate new or improved alternatives, and helping the decision maker to understand the trade-offs and select the “best” alternative.

To implement our approach to decision support, we employ the asynchronous-team (A-team) architecture. An asynchronous team is a team of software agents that cooperate to solve a problem by dynamically evolving a shared population of solutions [Murthy 1992; Talukdar, Pyo, and Mehrotra 1983; Talukdar, deSouza, and Murthy 1993]. The agents work asynchronously and embody their own methods for deciding when to work, which solutions to work on, and how to generate new or improved solutions. Agents do not explicitly coordinate their activities with one another. Cooperation is achieved by sharing results in populations; agents can work on solutions created by other agents.

The best solution alternatives are usually the result of contributions made by many agents embodying multiple problem-solving approaches [Salman, Kalagnanam, and Murthy 1997].

An A-team has three types of agents: constructors, improvers, and destroyers (Figure 2). Constructors take only the problem definition and create new solutions. Deterministic constructors attempt to run only once, while constructors that use randomized algorithms may run many times to add a variety of solutions to the population. Improvers try to improve upon the current set of solutions in the population by modifying or combining existing solutions to create new solutions. Destroyers try to keep the population size in check and focus the improvers’ efforts by removing bad solutions from the population. Within the population, each solution is evaluated with respect to multiple objectives. Typically, solutions to scheduling problems are evaluated in terms of cost, timeliness, customer satisfaction, and manufacturing disruptions. As a group, the agents in an A-team evolve the population of solutions to provide a set of good alternatives that illustrates important trade-offs among the competing objectives. Because the agents share access to the population of solutions, an A-team is like a blackboard system but without a central controller [Erman et al. 1980]. In a blackboard system, a central memory, or blackboard, is used to post problems and subproblems and to post possible solutions. A central controller activates knowledge sources to work on these problems and post solutions or new subproblems. A-teams also have certain characteristics of

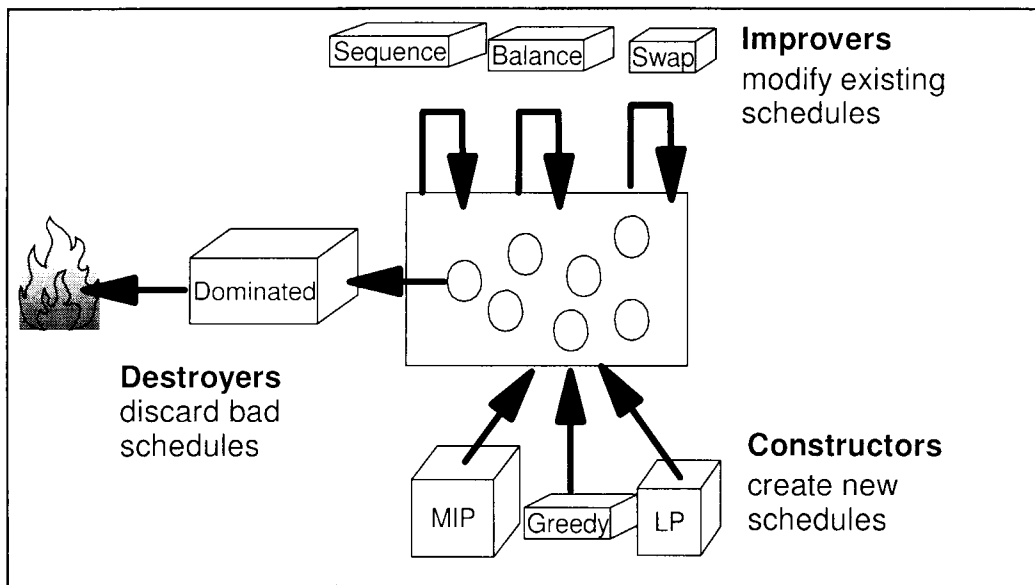


Figure 2: In the A-team architecture, various algorithms are encapsulated as constructor, improver, or destroyer agents. These agents cooperate by working on a common population of solutions, and they evolve the population toward a pareto-optimal frontier.

genetic algorithms in that a population of solutions evolves over time [Holland 1975]. However, unlike genetic algorithms, the mechanisms for altering individual solutions may be highly directed because they take into account domain-specific knowledge, rather than depending upon random mutation or crossover.

To solve multicriteria optimization problems, classical search and optimization methods convert multiple-objective functions into one scalar function, and can at best find one Pareto-optimal solution at a time. To create multiple Pareto-optimal solutions, these methods are applied repeatedly, each time varying the user-defined parameters for aggregating objectives. In contrast, the A-team approach, which is an evolutionary method, can find multiple Pareto-optimal solutions simultaneously, using its population-based

search. Because of this parallel searching and freedom from user-defined parameters, the A-team approach has an advantage over classical methods in solving multicriteria optimization problems. For difficult optimization problems, any single algorithm is not likely to produce high-quality solutions in a reasonable time for all problem instances. The agents in an A-team constitute a portfolio of algorithms that is more robust than any single algorithm [Huberman, Lukose, and Hogg 1997; Lee et al. 1996; Murthy 1992]. Beyond the benefits of using multiple algorithms independently, the A-team architecture allows the agents to cooperate synergistically and leads to higher quality and more diverse solutions. Individual agents may focus on optimizing different objectives and explore different parts of the search space for feasible solutions.

The simplicity and modularity of the A-team architecture make it well suited for creating systems that are used daily in production environments. Modularity allows new algorithms, in the form of agents, to be added to an A-team without making major modifications to the existing system. Algorithms can be incomplete, since partial solutions can be added to the population to serve as seeds for improvers. As a practical matter, modularity allows a customer-support engineer to configure and tune the system easily at installation time by turning agents on or off.

The A-team architecture originated at Carnegie Mellon University [Murthy 1992; Talukdar, Pyo, and Mehrotra 1983; Talukdar, deSouza, and Murthy 1993]. It has been used to solve a variety of problems, including traveling-salesman problems [deSouza 1993], design of robots from task specifications [Murthy 1992], design of fault-tolerant power networks [Chen 1992], and train-scheduling problems [Tsen 1995]. At IBM, we have extended this work by implementing the first A-team-based solutions for real industrial problems. Ours is also the first application of the A-team architecture to multiobjective scheduling. The problems we consider are larger and more complex than any of the problems considered in previous A-team applications. As a result, our application necessitated the use of more powerful agents that perform directed search toward the Pareto frontier. One of our key extensions to the A-team architecture was to add decision makers as agents in the A-team. A scheduler can take on the role of an agent by adding new so-

lutions to the population or by removing or improving existing solutions. Previous work on A-teams had already shown that the architecture enables synergistic cooperation between agents. We found that the human-as-agent paradigm pushed this cooperation to a new level.

Recently it has been shown that agent-based architectures hold promise for solving complex multiobjective optimization problems. Liu and Sycara [1996] have shown the benefits of a tightly coupled agent architecture in solving the job-shop-scheduling problem. Beck and Fox [1994] presented a mediated approach to multi-agent coordination in the context of a supply-chain management system. A model of the supply chain in which each of the key players in the chain is encapsulated in an autonomous agent is described by Swaminathan, Smith, and Sadeh [1996]. Smith et al. [1990] use a blackboard-style architecture for generating and revising factory schedules, selectively employing a set of distinct methods to generate, revise, or analyze specific components of the overall schedule.

Using the A-team architecture, we built the following multiobjective decision-support tools for scheduling paper production:

- (1) A global scheduling package that, given a set of orders, provides multiple candidate enterprise schedules, each of which specifies an allocation of orders to machines, a sequence of runs or orders on each machine, a trim sheet for each run, and a loading schedule;
- (2) A detailed run-formation and sequencing package that, given a machine and an allocation of orders to that machine, pro-

vides multiple sequencing and run-formation alternatives;

(3) A detailed trim package that, given a set of orders in a run, provides multiple candidate trim sheets for that run; and

(4) A detailed load-planning package that, given inventory and a production schedule, provides multiple candidate loading schedules.

The objectives of the A-team can be configured by a consultant when the system is installed. Typically there are four objectives: maximizing profit, maximizing on-time delivery, maximizing customer satisfaction, and minimizing manufacturing disruptions.

### **Algorithm Overview**

Orders are allocated to paper machines based primarily on transportation-cost considerations. Machines are geographically distributed, and producing an order on a machine close to the order's final destination saves transportation costs. Order allocation should also consider the due dates of the orders and the load balance on each machine to ensure on-time delivery. We use the following approaches for allocating orders: linear programming, integer programming, network flow models, and dispatch algorithms. The order-allocation methods create partial solutions in which orders are allocated and given an initial sequence on each machine. The initial sequence of orders defines a sequence of runs on each machine. The problem we consider at this stage generalizes other machine-scheduling problems studied [Ho and Chang 1991; Lee and Pinedo 1997]. It can be characterized as scheduling orders on parallel nonidentical machines, subject to job-machine restrictions, machine- and

sequence-dependent setups, batch-size preferences, job-machine assignment costs, and earliness and tardiness penalties, with additional implications on downstream processes. Akkiraju et al. [1998] discuss further details of our approach for solving this complex scheduling.

The linear-programming and network-flow-based models are fast, but they create allocations in which some orders may be split between mills and machines. In some cases, splitting orders across multiple machines may be undesirable, especially if the machines are geographically dispersed (which complicates order tracking) or if they produce papers of different quality. In these cases, we "fix" the resulting allocation by merging or partially merging some of the split orders. We use several heuristic approaches for fixing allocations with order splitting. The idea is to merge pieces of an order while considering transportation cost, due dates, and the utilization of the machines.

In dispatch algorithms, the method is first to sort the orders according to some criteria and then to allocate each order in the list to a machine based on a priority index of the order-machine pair. The dispatch approach does not split orders across mills and machines. Sort criteria are based on combinations of order properties, such as due date, processing time, and tardiness penalty. For example, we sort orders in increasing order of their due dates, and then we sort orders with the same due date by grade. In our implementation, we use several combinations of primary and secondary sort criteria for sorting. We calculate the priority index by combining several measures of an order-machine

pair, such as order-completion time, earliness, tardiness, transportation cost, and processing times.

For each order-machine allocation, we create new sequences for the orders on each machine, as alternatives to the initial sequence generated by the order-allocation agents. In order sequencing, the goal is to produce the orders on time, to ensure smooth transitions between grades, and to obtain good trim efficiency from the resulting runs. The sequencing problem, at this stage, falls into the domain of single-machine sequencing with sequence-dependent setups with the objective of minimizing weighted tardiness [Abdul-Razaq, Potts, and Van Wassenhove 1990; Du and Leung 1990; Koulamas 1994; Lee, Bhaskaran, and Pinedo 1997]. We use dispatch algorithms for sequencing orders on a given machine; the idea is to select one job at a time from the set of remaining jobs, based on some priority index, and schedule the job with the highest priority as the next job on that machine. We calculate the priority index by combining different properties, such as earliness, tardiness, and processing and setup times. We have several algorithms that follow this framework but use different priority indices. We also use some priority rules from the literature, such as the ones described by Panwalkar, Smith, and Koulamas [1993] and Holsenback and Russell [1992]. When we sequence the orders to form runs, we also consider downstream implications on trim efficiency and shipping. We compute estimates of the trim efficiency and shipping cost, and modify the runs as necessary.

To improve order allocation and se-

quencing, our improvers change the sequence of orders on a machine or swap orders between mills and machines. Some improvers move only one order at a time, whereas others may move a group of orders, for example, a whole run, to a new position. Order-allocation improvers focus mainly on reducing the transportation cost and balancing the load on the machines while considering the impact of any move on other measures, such as tardiness. Sequencing improvers focus on reducing tardiness and setups while considering the impact of any move on other measures, such as transportation cost and trim efficiency. We also have some improvers that try to reduce the number of very short runs and setups. Having very short runs is undesirable because they may violate the minimum-run-size requirements and they are more likely to have poor trim efficiency. These improvers may merge two different runs of the same grade or move some orders from a long run into a short run to decrease setups or the number of short runs and to improve trim efficiency. Again, they consider the impact of these moves on other measures, such as the transportation cost and tardiness.

During each production run, a paper machine produces a set of reels. Cutting the reels into smaller rolls of ordered sizes is called *trimming*. For example, a reel of width 200" can be cut into four rolls of width 43" and one roll of width 27" resulting in one inch of trim loss  $(200 - (4)(43) - (27) = 1)$ . Some of the constraints we consider in trimming are the maximum cutting-pattern deckle (which must not exceed the machine's deckle), the maximum number of rolls in a pattern, and the maxi-

imum number of narrow rolls in a pattern. The main objective in trimming is to minimize the trim loss while minimizing deviations from the quantities ordered.

Our agents first generate a set of efficient patterns that maximizes the use of the deckle. They then select a subset of these patterns to produce the rolls for that run. We use a randomized procedure to generate patterns to ensure that the generated patterns exploit the diversity of roll widths in the run. To select the patterns to use and the repetitions of each pattern, some of the agents use integer programming, whereas others use linear programming with rounding. Having selected the patterns and their repetitions, the agents sequence the patterns with a combination of objectives, such as making high-priority orders first, finishing orders in the sequence of their due dates, minimizing pattern changes (winder setups), minimizing diameter changes, and minimizing the number of orders in process.

The pattern-selection problem with the objective of minimizing trim loss is the well-known one-dimensional cutting-stock problem (CSP) [Gilmore and Gomory 1961, 1963; Pierce 1964]. Several heuristic procedures have been proposed for CSP, mostly based on sequential pattern generation or LP rounding, or a combination of both [Ferreira, Neves, and Castro 1990; Haessler 1988a, 1988b; Stadtler 1990; Vahrenkamp 1992]. Some researchers also considered lower and upper bounds on ordered quantities and winder setups [Goulimis 1990; Haessler 1971, 1988a; Wascher 1989]. For reviews on cutting-stock problems, see Golden [1976], Haessler and Sweeney [1991], Hinxman

[1979], and Dyckhoff [1990]. To the best of our knowledge, the trim problem we solve by simultaneously considering all the constraints and trading off all the objectives has not been considered before.

Keskinocak et al. [1998] give further details of our solution approaches and additional references on trim-related problems.

Some improver agents increase trim efficiency by modifying runs that have relatively low trim efficiency. Such agents may move orders or rolls from one run to another, possibly on different machines. The two modified runs are then retrimmed again. If the resulting trim is improved, the change is kept. We have several methods for selecting orders or rolls to be moved. In one method, we examine the trim patterns having the lowest efficiency in the trim sheet and select an order that is made in those patterns. In another method, we look for a roll size that makes up a large fraction of all the rolls to be made in the run. If this roll size does not trim well, we move some of its orders to another run. We repeat this operation using different combinations of runs and orders in an attempt to improve the trim efficiency of the entire schedule.

In many paper mills, rolls are loaded directly onto vehicles as they emerge from the wrapping machine. Loading is the process of deciding which vehicles to use and how to load them. The choice involves selecting the mode of transportation, selecting the sizes of vehicles to use, and deciding how to load the rolls into the vehicles. Mode choices typically include rail and truck. Rail is generally less expensive but has longer transit times. For each mode, the available vehicles come in fixed sizes

with weight and volume limits. There are also limits on the number of vehicles that can be loaded in a shift and the number and types of vehicles that are available per day. In addition to the physical limitations, insurance regulations restrict loading alternatives. To be insured for shipment by rail, one must place the rolls in the vehicle following a prescribed loading pattern. Because of the restriction on loading rail cars, determining whether a set of rolls can be loaded into a given vehicle is a variant of the multiple-knapsack problem, which is known to be NP-complete [Martello and Toth 1990]. To improve vehicle utilization, one can mix rolls going to different locations in the same vehicle. However, mixing rolls going to different locations incurs drop-off charges and extra distance charges and may increase tardiness due to increased transit times.

The loading problem is similar to a multiple-knapsack problem [Martello and Toth 1990] with multiple sizes of knapsacks, knapsack availability constraints, and complex rules for determining legal combinations of items in a knapsack. To address this problem, we use two basic approaches: traditional knapsack-packing algorithms and algorithms that work with a set of legal loading patterns. Our greedy knapsack-packing algorithms select vehicles based on cost or on-time delivery and load the vehicles until the weight or volume limits are reached. The results are then checked to ensure that the rolls can be loaded following prescribed loading patterns. These algorithms work best for trucks, where this check almost always succeeds.

The second class of algorithms generate and share a set of legal loading patterns. A pattern generator enumerates subsets of rolls that meet weight and volume constraints for each type of vehicle and then filters the set to remove the subsets of rolls that cannot be loaded into a vehicle using a prescribed loaded pattern. The loading problem is then reduced to selecting how many of each pattern to choose to ship all the items. We use a combination of mixed-integer-programming algorithms and greedy algorithms to select patterns. This second approach is most effective for rail cars.

In addition to these two basic approaches, we also make use of local improvement routines. These algorithms look for opportunities to reduce costs, reduce drop shipments, and improve vehicle utilization by swapping items between vehicles and changing vehicle sizes. They perform a stochastic hill climbing and explore the area around the solutions generated by the first two types of algorithms, looking for improvements. Further details about our load-planning solution methods can be found in the appendix and in Keskinocak et al. [1998].

Assembled in the A-team framework, these algorithms cooperate synergistically to generate high-quality, complete enterprise schedules (Figure 3). Initially, the allocation agents create a population of allocations. Run formation and sequencing agents take promising solutions from the population of allocations and create sequenced allocations. Trim agents create trim sheets for promising sequenced allocations and load-plan agents create load plans for promising sequenced allocations

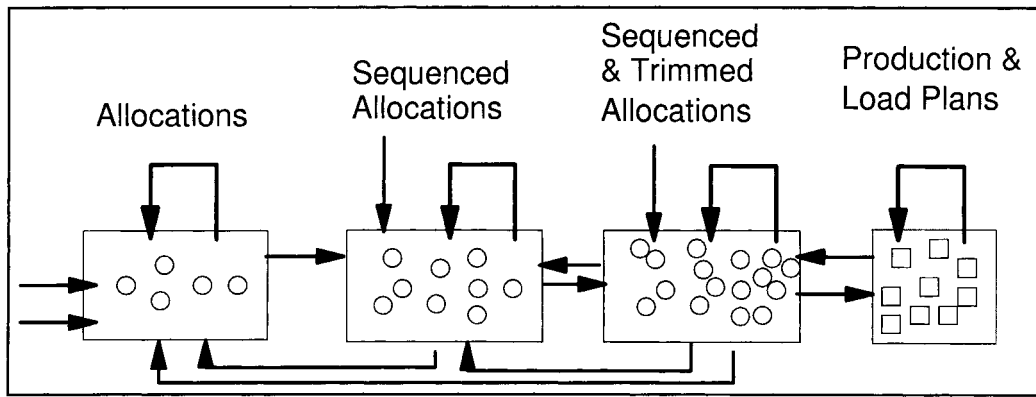


Figure 3: Solutions in the population evolve as different agents work on them. In this figure, the rectangles correspond to populations of solutions (or partial solutions) and the arrows correspond to agents that create or modify solutions.

with trim sheets. Improver agents attempt to improve the (partial) solutions in each population. Improver agents also provide feedback by using information obtained from trim sheets and load plans to create modified or improved allocations and sequenced allocations. In addition, destroyer agents remove solutions from the populations that are far from the Pareto frontier to keep the population size in check

**Facilitating Cooperation Between the Scheduler and the Software**

In our decision-support approach, we promote cooperation between the scheduler and the system to improve solution quality and the decision-making process. While interacting with the system, the scheduler can suggest alternatives and understand and modify solutions. The system can take guidance from the scheduler, offer solution alternatives, and point out constraint violations. Schedulers can submit manually created or modified solutions back to the scheduling system for evaluation and comparison with other solutions, as well as for further improve-

ment by the system. By working with the system to explore the solution space, the schedulers gain insight that gives them confidence in their final decision. To cooperate, the scheduler and the system need to communicate effectively.

Our graphical user interface (GUI), developed specifically for paper-mill scheduling, allows the schedulers to explore the particulars of a scheduling problem and its solutions. The interface facilitates communication by providing multiple views of the data at different levels of granularity and uses the scheduler’s own graphical language as a basis for direct manipulation of a schedule. The views provided by the interface include the following:

- A summary of results presents the objective function evaluations for each enterprise schedule in the nondominated set.
- A detailed view of each enterprise schedule for all the machines in the schedule shows the duration of each run and the run sequence.
- A detailed view of each machine schedule in a given enterprise schedule shows

the duration and product type of each run and the run sequence on the machine.

—A detailed view of a run in a given machine schedule (Figure 4) shows the orders in the run and the trim sheet.

—A detailed view of a loading schedule shows how orders are loaded onto vehicles.

Figure 4 illustrates our approach to user interaction. To understand how the interface is used, consider a typical scenario in

which a scheduler is trying to create a trim sheet for a production run and initially has the system generate a set of candidate solutions. Such is the case in Figure 4 where the system has produced multiple solutions and presented a set of 24 non-dominated solutions. The scheduler has then narrowed the set by using the criteria boxes at the bottom of the grid to view only solutions with no more than seven setups and no more than two orders un-

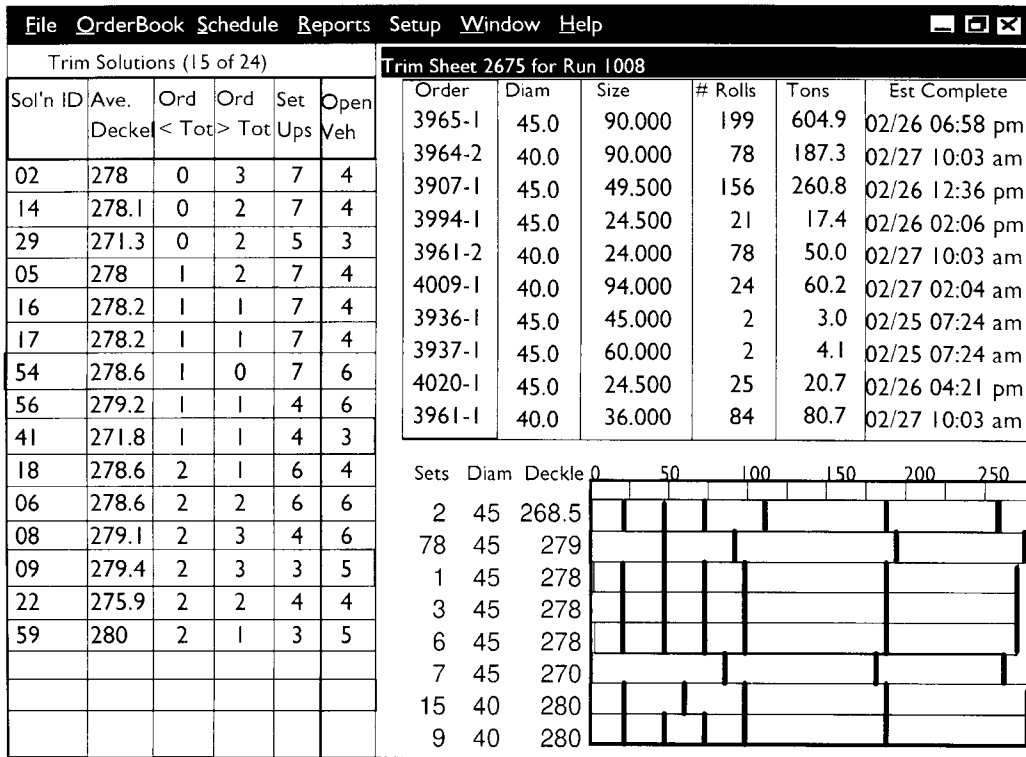


Figure 4: Using the trim sheet editor, schedulers can examine and manipulate trim sheets. The grid on the upper right side gives order specifications including widths, diameters, and the number of rolls. The grid on the lower right shows a graphical representation of a trim sheet solution, which the scheduler can directly manipulate. Each line shows the placement of the slitting knives and the assignment of the resulting rolls to orders. Within the trim sheet, the background color changes for each pattern that requires a new knife setup. Also, selecting an order highlights its position in the trim sheet. Finally, the grid on the left summarizes the set of alternative trim solutions and their evaluations. Each trim sheet has 35 evaluations that measure its quality, such as the number of setups, and downstream consequences, such as the number of open vehicles.

der tolerance. Clicking on a column, the scheduler can sort the solutions by a particular evaluation. In this case, the scheduler has sorted by "orders under tolerance," which is the number of orders for which the quantity produced is under the minimum order quantity. Double clicking on a solution shows its details on the right-hand side and allows the scheduler to determine which orders are being overrun and by how much. In this case, orders 4020-1 and 3961-1 are being overrun by one and six rolls, respectively. To improve the solutions and explore alternatives, the scheduler can directly manipulate the trim sheet by dragging rolls between patterns, reordering, adding, or deleting patterns. Once the scheduler finalizes a trim sheet, it can be attached to the production run as its current trim sheet using the button at the top of the screen and released for production using the check box at the bottom of the screen.

By allowing direct manipulation of the graphical representation of a schedule and by providing multiple levels of detail, our interface enables the communication that is required for effective cooperation between the scheduler and the system.

### **System Requirements and**

#### **Performance**

In a typical installation for a paper company, master schedulers located at the company headquarters use the global scheduling package to create long-term, enterprise-wide schedules that allocate orders to mills and perform initial-run formation, sequencing, trimming, and loading. Schedulers located at each mill take the resulting enterprisewide schedule and use the detailed trim-and-load-plan com-

ponents to perform detailed optimization for each paper machine, each trim sheet, and each load plan. The running time for the A-team is a parameter of the system that can be set to reflect the size of the problem and the amount of optimization required.

To get an idea about the performance and running time, consider an example consisting of 5,000 orders to be scheduled on nine machines located in five mills. Running the A-team for one and a half hours on a single processor IBM RS/6000 Model 591 (256MB of memory) generates approximately 100 enterprisewide solutions, from which about 10 solutions belong to the nondominated set. Our discussions with the scheduling personnel at the mills indicate that generating a single enterprise schedule for an instance of this size would take an experienced scheduler more than a week, if order allocation and run formation was done manually (or given) and a software package was used for trimming. While trimming a run, most of the trim packages consider only the orders in that run, which requires the scheduler to make a lot of manual changes to the composition of the runs. One of the main advantages of our system is that it changes the composition of the runs automatically to improve trim efficiency while meeting other objectives. Hence, the one-and-a-half-hour running time is quite reasonable, given the size and the complexity of the problem. In our system, optimization of a single trim sheet or load plan takes less than a minute. Evaluating the effects of a manual change, like reordering the patterns in a trim sheet, is almost instantaneous.

**Business Impact**

Our paper-mill-scheduling system has significantly improved the scheduling and decision-making process for our customers, giving them substantial monetary returns. Improvements come from the higher quality of the solutions that our system generates and from positive changes in business processes that our approach to decision support fosters. In terms of solution quality, one paper company, Madison Paper Industries, reports a reduction in trim loss of six tons per day and a 10-percent reduction in freight costs [Hoffman 1996; Shaw 1998]. Each of these savings amounts to \$2 to \$3 million per year. Our system has allowed other customers to change their business practices to achieve perfect trim (no trim loss) consistently and to improve their ability to negotiate with customers when accepting orders.

Customarily, the value of a scheduling system is determined only by the quality of the schedules it produces. The schedules our system generates for each stage of the process improve upon the state of the art in scheduling. Adam Stearns of Madison Paper Industries reports, "We would use our old trim package, throw orders into it, and let it trim them the best it could. Then we would let the IBM module take the same orders and manipulate them to come up with a trim. We saw that the IBM package was consistently saving over two inches [of trim loss]—just an incredible amount. . . . Testing shows that we are getting about 10 percent savings annually on distribution costs from the load planning piece alone, which amounts to millions of dollars. . . . We expected the sys-

tem's GUI to make load planning easier, but we didn't expect to gain these efficiencies" [Shaw 1998, page 75].

Our system also achieves a global perspective that results in further schedule improvements by considering options for changing a schedule for one stage of production to improve the schedule for subsequent stages. Schedulers have long recognized that such changes could produce dramatic improvements. However, the effort needed to explore such options was prohibitive when incompatible software packages were used to schedule the various stages and each scheduler was familiar with only the software used to schedule his or her stage of production. To illustrate how our scheduling system takes advantage of this global perspective, we will describe an example that involves moving orders between mills, machines, and production runs to reduce trim loss.

It is often possible to improve trim efficiency by adding orders to a run to get better trim patterns. The difficulty is in identifying potentially good orders to move and in determining the impact of the move on the rest of the schedule. Moving an order to an earlier run can increase inventory costs, reduce paper quality if the length of the future run becomes too short, and increase tardiness by delaying the start of subsequent runs. Moving an order between mills can increase shipping costs and shipping times. Without an integrated system, determining the extent of such impacts can be difficult and time consuming. To reduce complexity, schedulers often use simple business rules to limit their searches and the impacts they need to consider. Such rules might include looking

only at orders already assigned to the same machine and scheduled for production in the next month. With such a limitation, schedulers can ignore impacts on inventory cost, shipping cost, and load balance and consider only whether the move increases tardiness for the other orders assigned to the same machine in the next month. Our integrated software allows these business rules to be relaxed. Both the scheduler and the software can explore a larger set of alternatives to improve the quality of the schedules.

Improving schedule quality has obvious impacts on profitability. However, just as important are the opportunities to change business processes that our system affords by showing multiple good alternatives, some of which may violate some constraints. These solutions can suggest good opportunities to look for alternative means of production and can suggest when it would be profitable to negotiate to change customer requirements or business policies.

One of our customers regularly achieves perfect trim by using solutions with underruns that violate the minimum order quantity for one or more orders. At this mill, orders can be filled from production, from inventory, or from a combination of inventory and production. The mill keeps a small inventory of rolls that it can retrim to meet the specifications of a particular order. The problem for the schedulers is to determine when to use the inventory. Using our software, the schedulers look for solutions that give perfect trim and try to complete underrun orders from inventory. Overruns become new inventory.

Scheduling software generally views or-

der specifications and manufacturing policies as hard constraints that a schedule must meet. However, these constraints are often negotiable if doing so improves finances or customer service. For example, some mills have a policy that limits the number of setups on a winder to at most one per hour. However, the production manager may be willing to violate the policy if it substantially increases trim efficiency or reduces order tardiness. Similarly, customer service representatives can negotiate with a customer to change the specifications of an order but would do so only if it would substantially benefit another order or significantly reduce expenses: “. . .when our sales staff calls with an order for a specific tonnage, roll size and destination, we can enter this into the scheduling system as ‘pending’ and it will tell us when we can do it and what it is going to cost if we do it. That is powerful, powerful information for sales to provide a customer” [Shaw 1998, page 80]. If the solution suggested by the system under- runs a pending order, then the sales staff may offer to split the order, filling the rest of the order at a later date. If a solution overruns an order, the sales staff may offer a volume discount if the customer takes the extra production.

Finally, the global perspective of our scheduling software has sped up the scheduling process and improved communications among schedulers since they can share schedules and see the impact of their changes on other stages of production. This reduces the time needed to generate a new schedule and allows production to react more quickly.

Our scheduling system has been well re-

ceived in the marketplace because it helps paper manufacturers satisfy the needs of their customers while reducing costs. Improvements in schedule quality result in measurable monetary returns that give our system a payback period of less than one year [Shaw 1998]. Other improvements, such as reduced scheduling time and changes in business processes, are harder to quantify but are appreciated by paper manufacturers as they operate in a highly competitive market.

## APPENDIX

### Order Allocation Based on Linear Programming

To model the order-allocation problem as a linear program, we divide our planning horizon into time buckets (for example, a bucket can be a week or a day). In the formulation, we use the following indices:  $j$  for orders,  $k$  for machines,  $i$  for mills,  $b$  for time buckets.

The following quantities are input to the model:

$demand(j)$  : quantity (e.g., tons) of order  $j$ ;

$duedate(j)$  : due date (number of time buckets) for order  $j$  (this is the desired date to ship the order);

$fixedcost(j,k)$  : fixed cost (for example, transportation cost per ton) of assigning one unit of order  $j$  to machine  $k$ ;

$req(j,k)$  : required capacity to produce one unit of order  $j$  on machine  $k$ ;

$cap(k,b)$  : capacity of machine  $k$  in bucket  $b$ ;

$earlycost(j)$  : penalty for producing one unit of order  $j$  early for one bucket;

$latecost(j)$  : penalty for producing one unit of order  $j$  late for one bucket;

$M(j)$  : set of machines eligible to produce order  $j$ ;

$revenue(j)$  : revenue from satisfying order  $j$ ;

$ucost(j)$  : penalty per unit of unmet

demand  $j$ .

The following variables are used in the formulation:

$X(jkb)$  : quantity (tons) of order  $j$  produced on machine  $k$  in bucket  $b$ ;

$P(jb)$  : total quantity (tons) of order  $j$  produced in bucket  $b$ ;

$inv(jb)$  : positive inventory of order  $j$  in bucket  $b$  ( $b < duedate(j)$ );

$backlog(jb)$  : negative inventory (backlog) of order  $j$  in bucket  $b$  ( $b > duedate(j)$ );

$U(j)$  : unsatisfied quantity of order  $j$ .

Using the variables and input parameters above, we model the allocation problem as a linear program as follows:

$$\begin{aligned} \text{Min } & \sum_j \sum_k fixedcost(jk) \sum_b X(jkb) \\ & + \sum_j earlycost(j) \sum_b inv(jb) \\ & + \sum_j latecost(j) \sum_b backlog(jb) \\ & + \sum_j [revenue(j) + ucost(j)] U(j) \end{aligned}$$

subject to

$$\begin{aligned} \sum_{k \in M(j)} X(jkb) &= P(jb) \\ &\text{for every pair } (j,b), \end{aligned} \quad (1)$$

$$\begin{aligned} \sum_b P(jb) + U(j) &= demand(j) \\ &\text{for every order } j, \end{aligned} \quad (2)$$

$$\begin{aligned} \sum_j X(jkb) req(jk) &\leq cap(kb) \\ &\text{for every pair } (k,b) \end{aligned} \quad (3)$$

$$\begin{aligned} inv(jb) &= \sum_{t \leq b} P(jt) \text{ for } b < duedate(j) \\ &\text{and for every pair } (j,b), \end{aligned} \quad (4)$$

$$\begin{aligned} backlog(jb) &= demand(j) - \sum_{t \leq b} P(jt) \\ &\text{for } b > duedate(j) \text{ and for every } j. \end{aligned} \quad (5)$$

All variables are nonnegative.

The objective is to minimize cost minus revenue (or equivalently maximize profit)

while meeting the demand as much as possible within the given time buckets. Constraints (1) compute the quantity of order  $j$  produced in bucket  $b$ . Constraints (2) ensure that the total quantity of order  $j$  produced plus any unsatisfied quantity is equal to the quantity of demand  $j$ . Constraints (3) ensure that, for each machine, the production amount does not exceed the available machine capacity. Constraints (4) and (5) compute the amount of inventory and backlog for each order in each bucket. Note that variables  $P(jb)$ ,  $inv(jb)$ ,  $backlog(jb)$ , and  $U(j)$  can easily be eliminated from the formulation but are included for clarity.

The summation  $\sum_b X(jkb)$  in the solution to the above linear program denotes the total quantity of order  $j$  allocated to machine  $k$ . In this allocation, some orders may be split among different mills or machines. Typically, allocating parts of an order to different machines is undesirable due to quality and order tracking considerations. To prevent order splitting across machines or across mills, one approach is to add 0–1 variables to the formulation and an additional set of constraints. Unfortunately, in that case the resulting formulation becomes an integer program [Nemhauser and Wolsey 1988]. In a typical application, we may have about 1,000–5,000 orders to be allocated to five to ten machines in three to six mills over four to 12 buckets (weekly buckets in a one-to-three month horizon). For such large instances, it becomes impractical to solve the integer program. Therefore, we usually solve the linear programming relaxation and “fix” the resulting allocation by merging some of the split orders. We use several heuristic approaches for fixing allocations with order splitting. The idea is to merge pieces of an order, which are allocated to different machines, while considering transportation cost, order due dates, and load balance on the machines. We say

that an order needs fixing if it is split in the current allocation and if this splitting is undesirable due to the reasons mentioned above. Here is the basic framework for the fixing heuristics we use:

- (1) Pick an order that needs fixing.
- (2) Select  $k$  pieces of this order and allocate them to the same machine.
- (3) If there are still orders that need fixing, go to Step 1.

While we select an order that needs fixing (in Step 1), we consider the number of pieces the order is split into, the number of different machines to which the order is allocated, and the maximum distance between any two mills to which the order is allocated. If some or all of these quantities are large for some order, that order gets priority for being fixed. Each of our fixing heuristics considers different combinations of the quantities above for selecting the next order to fix.

#### Trim Solution Methods

To generate trim solutions for a production run, trim construction agents first separate the run into subruns, then they create a set of cutting patterns to produce the rolls in the subrun, and finally they merge and sequence the patterns from all the subruns. A sequenced collection of patterns that has been selected to produce and cut the reels in a run is called the *trim sheet* (for that run).

Before we create a trim sheet for a run, we first divide the run into subruns. Each subrun is a group of orders whose characteristics, such as core type, diameter, and wind direction, would allow them to be cut together in the same pattern on a winder. The partitioning of a run into subruns is usually not unique, and the choice of partitioning can have a major impact on the trim efficiency of the run. Using randomized grouping algorithms, the subrun formation methods explore the possibilities, which results in a variety of trim solutions.

Let  $b = (b_1, \dots, b_k)$  denote the demand vector for a subrun, where  $b_i$  is the number of rolls demanded of width  $i$ ,  $k$  is the number different widths, and  $\{w_1, \dots, w_k\}$  is the set of widths in the subrun. Let  $b^{\min}$  and  $b^{\max}$  be the vectors that represent the tolerances in the ordered amounts, where  $b^{\min} \leq b \leq b^{\max}$ . A pattern can be represented as a vector  $a = (a_1, \dots, a_k)$ , where the  $i$ th entry denotes the number of repetitions of  $i$ th width in the pattern.

To create a trim sheet for a subrun, our agents use the following framework. The agents first generate and store for reuse a large number of good, feasible cutting patterns. These patterns must obey machine- and order-specific constraints, such as maximum number of rolls and roll position. They then select a subset of these patterns and decide how many of each pattern to use to create a “good” trim sheet. We use several selection policies, which we explain below. Finally, they allocate rolls to orders so as to minimize deviations from ordered amounts.

A large population of patterns is generated by one of several means. For small problems, complete enumeration of all feasible patterns can be done very efficiently. For large trim problems, we estimate the number of feasible patterns based on the set of distinct widths and the maximum deckle, and we invoke either the enumerative pattern generator or a random pattern generator to create the target number of patterns.

The random pattern generator follows this procedure:

Step 0: Combine all constraints on the maximum number of rolls of each width allowed in a pattern.

Step 1: Set the remaining width  $R$  equal to the deckle of the machine. Mark all the widths green.

Step 2: If the set of green widths is empty, go to Step 5. Otherwise, from the set of green widths, select a width, say  $w$ , ran-

domly. Let  $x$  be the maximum number of additional rolls of width  $w$  we can put in this pattern, where  $x$  is the lesser of  $\lfloor R/w \rfloor$  and the maximum number of rolls allowed of this width.

Step 3: If  $x = 0$ , mark this width red. If  $x > 0$ , select an integer  $y$  randomly between 1 and  $x$  and include an additional  $y$  repetitions of width  $w$  in this pattern.

Step 4: Update the remaining width  $R$  by setting  $R = R - yw$ . Return to Step 2.

Step 5: If the generated pattern is not already in the set of patterns we generated earlier, include it in the set. Otherwise, discard the pattern.

We have also implemented a proprietary algorithm to generate patterns in the presence of defects in the reels.

Each time an agent runs, a different subset of the patterns in the population is selected randomly as input to the pattern-selection process. The pattern-selection algorithms decide how many of each pattern to use to create a “good” trim sheet. Each of the several trim-constructor agents uses a different formulation of an LP or MIP. We have found that the groups of patterns selected by repeated solution of the linear relaxation of the integer program are a valuable resource for repeated invocations of the trim-constructor agents. We store these useful patterns in a pattern cache from which agents that use integer program methods can randomly choose.

Let  $R$  denote the deckle of the machine. The trim loss  $L_j$  of a given pattern  $j$  represented as  $(a_{j1}, \dots, a_{jk})$  can be calculated as follows:

$$L_j = R - \sum_i w_i a_{ij}$$

The constructors that strongly discourage deviations from ordered quantities use the following formulation:

Min  $cx + P_1s + P_2e$  subject to

$$Ax + s - e = b, x \text{ integer} \quad (\text{IP } 2)$$

where  $P_1$  and  $P_2$  denote the penalty for negative and positive deviations from ordered quantities, respectively. The cost of pattern  $j$ , denoted by  $c_j$ , can be equal to the trim loss  $L_j$  of that pattern, or another related quantity.

Other constructors assign lower penalties for deviations within bounds and higher penalties for deviations outside bounds. The following integer program models this situation:

$$\begin{aligned} \text{Min } & cx + P_{11}s_1 + P_{e1}e_1 + P_{s2}s_2 + P_{e2}e_2 \\ \text{subject to } & b^{\min} \leq Ax + s_1 + s_2 - e_1 - e_2 \\ & \leq b^{\max} \quad e_1 \leq b^{\max} - b, \\ & s_1 \leq b - b^{\min}, \quad x \text{ integer.} \end{aligned} \quad (\text{IP } 3)$$

In (IP 3),  $s_1$  and  $e_1$  measure the negative and positive deviations within the lower and upper bounds on ordered quantities, whereas  $s_2$  and  $e_2$  measure the additional deviations that are outside the bounds. The penalties  $P_{s1}$  and  $P_{e1}$  for deviations within bounds are smaller than the penalties  $P_{s2}$  and  $P_{e2}$ .

Next we discuss some of our pattern-selection algorithms. Note that in successive invocations of these algorithms, the input patterns are selected from the population randomly, and thus different solutions are likely to result.

Pattern Selection Algorithm 1: Randomly select a subset of patterns from the population to form matrix  $A$ . Solve the linear programming relaxation of (IP 2); round each variable to the nearest integer.

Pattern Selection Algorithm 2: Same as Algorithm 1 except use (IP 3).

Pattern Selection Algorithm 3: Randomly select a subset of patterns from the population to form matrix  $A$ . Solve the linear programming relaxation of (IP 2); round each variable to the nearest integer. Add the patterns corresponding to nonzero  $x_j$  to a cache (to use in successive invocations of this algorithm). Create a new matrix  $A'$  using a randomly selected subset of the

patterns in the cache. Solve (IP 2) by replacing  $A$  with  $A'$ .

Pattern Selection Algorithm 4: Same as Algorithm 3 except use (IP 3).

For small problems, we may solve (IP 2) and (IP 3) exactly, but exact solution becomes impractical for larger problems. We have several other algorithms for pattern selection, which follow similar approaches with small variations. For example, they modify the objective function weights, allow for positive deviations only, allow for negative deviations only, or alter  $b^{\min}$  and  $b^{\max}$  depending on order priority.

The  $A$  matrix, which defines the linear program solved by a pattern-selection algorithm, is predefined by the generated patterns. We do not use column generation to solve the linear program. The disadvantage of this approach is that, especially for large problems, generating all possible patterns up front may be impractical, and using only a limited number of patterns may result in suboptimal solutions. On the other hand, we have experienced better results with this approach in practice compared to the column-generation approach, due to the numerous restrictions on feasible patterns in real-world instances and due to our ability to use the generated collection of patterns in multiple pattern-selection agents.

The final step in creating a trim sheet is sequencing the trim patterns to try to meet the objectives. Each agent can be configured to apply a predefined pattern-grouping and sequencing strategy. Each strategy can perform grouping and multi-key sorting of the trim patterns. The grouping and sorting criteria include grouping trim patterns to keep similar patterns together (minimizing setups), grouping trim patterns to keep like diameter patterns together, sequencing trim patterns to produce high priority orders first, sequencing trim patterns to produce orders in the sequence of their due dates,

and sequencing trim patterns to minimize the number of open vehicles.

Minimizing open vehicles is an issue when the loading-dock or staging area is limited; the schedule should avoid the partial production of many orders at any moment. This can be achieved by sequencing trim patterns to complete orders soon after they are started.

### Load Planning

Load planning assigns rolls to vehicles while satisfying a set of loading constraints. The loading constraints include constraints on which orders can be combined in the same vehicle; constraints on the vehicle types that can be used for an order; the restriction that only trucks can do drop shipments; and constraints on the weight and volume of items that can be put into a vehicles. The primary objectives of load planning are to minimize transportation costs and minimize tardiness. Secondary objectives include minimizing loading dock disruptions, which include drop shipments and excessive numbers of open vehicles, and satisfying customer preferences for particular types of vehicles.

In forming loads, we use a divide and conquer strategy in which orders are first partitioned into groups and then loads are created for each group. We model the grouping problem as a set-partitioning problem in which we minimize the maximum distance between destinations within each set and ensure a minimum total weight in each set.

In defining the grouping problem, we use the following indices:  $j$  for orders and  $g$  for groups. The following quantities are input to the model: the number of rolls for order  $j$ , the weight of one roll for order  $j$  ( $\text{weight}(j)$ ), and the destination of order  $j$  ( $\text{destination}(j)$ ).

The following functions are used in the model:

$$\text{weight}(g) = \sum_{j \in g}$$

$\text{weight}(j)$ : the weight of the rolls for group  $g$

$\text{distance}(g)$ :  $\text{Max}_{j1, j2 \in g}$   
 $\text{distance}(\text{destination}(j1), \text{destination}(j2))$ .

We use  $G(j, g) \in (0,1)$  to indicate if order  $j$  is in group  $g$ . Each order is assigned to one group, that is,  $\sum_g G(j, g) = 1$  for each order  $j$ .

The objective is to find a partitioning of orders into a minimal set of groups of sufficient size ( $\text{weight}(g) > \text{Threshold}$ ; for all groups) that minimizes the maximum distance between destinations within a group (minimize  $\sum_g (\text{distance}(g))$ ).

To find a solution to this grouping problem, we use a merging algorithm, which coalesces orders to form groups. Initially, a group is formed for each order. Then groups with orders going to the same customer location are merged. The algorithm then greedily selects the smallest group and merges it with its nearest neighbor until all the undersized groups have been merged. Group distance is measured by the amount of increase in  $\sum_g \text{max}(\text{distance}(g))$  that merging two groups would create.

To avoid repeatedly performing the legality check, we first create a set of legal vehicle loads and transform the load-planning problem into the problem of selecting loads from this set of known legal vehicle loads. This is similar to the approach used for trimming subruns.

The only significant constraint for trucks is the weight constraint. Rail cars are not weight limited but volume limited. Insurance regulations are also much more stringent and dictate loading patterns required for load insurance against damage. The loading patterns for rail cars are based on floor spots and height limits. Rolls are shipped standing on end and stacked to a maximum height that depends on the size of the rail car. For a given roll diameter, there is a fixed pattern of floor spots on which rolls can be stacked and each spot has a maximum height. Only rolls of the

same diameter, but different widths, can be mixed in the same rail car.

To accurately determine whether a set of rolls could be legally loaded into a rail car could entail checking all possible combinations of roll stackings, which would prove prohibitive for a large set of roll widths. Instead, we use a conservative approximation. We first check to see that there are enough rolls to fill the minimum number of floor spots ( $\sum_j p(j) > \text{minFloorSpots}(v)$ ) and then check that the sum of the roll widths is less than the sum of the floor-spot heights ( $\sum_j \text{width}(j) p(j) \leq \sum_s \text{height}(s, v)$ ). If either of these tests fails, the loading pattern is rejected.

We then use a greedy packing algorithm to stack the rolls on the floor spots until all the rolls are loaded or until a roll cannot be loaded. If all the rolls are loaded, then the pattern is accepted. If the rolls are all of a single width, then we can conclude that there is no legal packing since rearranging the rolls would not allow more to be packed into the vehicle. If there is a mixture of roll widths, we do an additional check. We first create a set of stacks of rolls up to the height of the highest floor spot. We begin by generating all possible stacks of a single roll width, then all stacks of two-roll widths, all stacks of three-roll widths, and so on until the combinations are exhausted or the number of stacks reaches a threshold, usually 200. These stacks are then sorted by height, and a greedy algorithm is used to select stacks of rolls until the rolls or floor spots are exhausted. If a packing is found, then the load pattern is accepted. Otherwise, the pattern is rejected.

The set of legal patterns,  $P$ , can be represented as a two-dimensional matrix  $A = (a_{ij})$ , where  $a_{ij}$  denotes the number of rolls of order  $i$  in pattern  $j$ . Let  $x_j$  denote the number of repetitions of pattern  $j$  in the selected load plan  $x$ , and let  $b_i$  be the number of rolls in order  $i$ . The total cost is the

sum of the costs for each load pattern:

$\text{Cost}(x) = \sum_j c_j x_j = cx$  where  $c_j$  is the cost of pattern  $j$ .

Similarly,

$\text{Tardiness}(x) = \sum_j t_j x_j = tx$  where  $t_j$  is the tardiness of pattern  $j$ .

$\text{Disruptions}(x) = \sum_j d_j x_j = dx$  where  $d_j$  is the number of disruptions or mixed orders in pattern  $j$ .

The problem can be formulated as an integer-programming problem:

$$\text{Min } a_1 cx + a_2 tx + a_3 dx \quad (\text{IP } 4)$$

subject to  $Ax = b$   $x$  integer.

The coefficients,  $a_1$ ,  $a_2$ , and  $a_3$ , give the relative weight for each objective. By varying the coefficients, we can find solutions along the efficient frontier.

We use many of the same algorithms developed for selecting trim patterns to select load patterns as well, using (IP4) and its linear-programming relaxation. We also have the following greedy load-pattern-selection algorithm.

Use a greedy algorithm to select patterns. First sort the patterns by cost per ton, with secondary sorts based on tardiness and disruptions. Greedily select patterns until all the rolls are loaded.

Variations on the load-pattern-selection algorithms include modifying the objective function weights, restricting  $A$  to include only patterns that use a single mode of transportation, restricting  $A$  to include only patterns that have zero tardiness, zero disruptions, or both, and changing the sort criteria for the greedy algorithm.

Local load-plan-improvement algorithms attempt to improve a load plan by changing the set of selected patterns and possibly introducing new patterns and generally take the form of heuristics that have proven useful. These include swapping trucks for trains to reduce cost, swapping trains for trucks to reduce tardiness, resizing vehicles to reduce dead freight, and combining vehicles to reduce dead freight.

## References

- Abdul-Razaq, T. S.; Potts, C. N.; and Van Wassenhove, L. N. 1990, "A survey of algorithms for the single machine total weighted tardiness scheduling problem," *Discrete Applied Mathematics*, Vol. 26, No. 3, pp. 235–253.
- Akkiraju, R.; Keskinocak, P.; Murthy, S.; and Wu, F. 1998, "An agent-based approach for multi-machine scheduling," *Proceedings of the Tenth Annual Conference on Innovative Applications of Artificial Intelligence*, Menlo Park, California, July 27–29, pp. 1013–1019.
- Beck, J. C. and Fox, M. S. 1994, "Supply chain coordination via mediated constraint relaxation," *Proceedings of the First Canadian Workshop on Distributed Artificial Intelligence*, Banff, Alberta, May 15.
- Beck, K.; DeKing, N.; Garcia, D.; Goldsmith, P.; Keaton, D.; Marshall, C.; McLaren, J.; Miller, D.; Page, R.; Routson, J.; Rowland, J.; Rudder, G.; Tsai, K.; and Whittingham, T. eds., *Pulp and Paper 1998 North American Fact Book*, Miller Freeman Inc, San Francisco, California.
- Bennett, D. P. 1998, "Multicriteria optimization," <http://www.ieor.berkeley.edu/~bennett/multicrit.html>.
- Biermann, C. 1993, *Essentials of Pulp and Papermaking*, San Diego Academic Press, San Diego, California.
- Chen, C. L. 1992, "Bayesian nets and A-teams for power system fault diagnosis," PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Das, I. 1998, "Multi-objective optimization," <http://www.mcs.anl.gov/otc/Guide/OptWeb/multiobj/index.html>.
- deSouza, P. S. 1993, "Asynchronous organizations for multi-algorithm problems," PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Du, J. and Leung, J. Y. 1990, "Minimizing total tardiness on one machine is NP-hard," *Mathematics of Operations Research*, Vol. 15, No. 3, pp. 483–494.
- Dyckhoff, H. 1990, "A typology of cutting and packing problems," *European Journal of Operational Research*, Vol. 44, No. 2, pp. 145–159.
- Erman, L. D.; Hayes-Roth, F.; Lesser, V. R.; and Reddy, R. D. 1980, "The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty," *ACM Computing Surveys*, Vol. 12, No. 2, pp. 213–253.
- Ferreira, J. S.; Neves, M. A.; and Castro, P. F. 1990, "A two-phase roll cutting problem," *European Journal of Operational Research*, Vol. 44, No. 2, pp. 185–196.
- Gilmore, P. C. and Gomory, R. E. 1961, "A linear programming approach to the cutting stock problem," *Operations Research*, Vol. 9, No. 6, pp. 849–859.
- Gilmore, P. C. and Gomory, R. E. 1963, "A linear programming approach to the cutting stock problem: Part II," *Operations Research*, Vol. 11, No. 6, pp. 863–888.
- Golden, B. L. 1976, "Approaches to the cutting stock problem," *AIIE Transactions*, Vol. 8, No. 2, pp. 265–273.
- Goodwin, R. T.; Rachlin, J.; Murthy, S.; and Akkiraju, R. 1998, "Interactive decision support: Advantages of an incomplete utility model," *Proceedings of the AAAI Spring Symposium on Interactive and Mixed Initiative Decision-Theoretic Systems*, Stanford, California, March 1998, pp. 55–61.
- Goulimis, C. 1990, "Optimal solutions for the cutting stock problem," *European Journal of Operational Research*, Vol. 44, No. 2, pp. 197–208.
- Haessler, R. W. 1971, "A heuristic programming solution to a nonlinear cutting stock problem," *Management Science*, Vol. 17, No. 12, pp. 793–802.
- Haessler, R. W. 1988a, "Selection and design of heuristic procedures for solving roll trim problems," *Management Science*, Vol. 34, No. 12, pp. 1460–1471.
- Haessler, R. W. 1988b, "A new generation of paper machine trim programs," *Tappi Journal*, Vol. 71, No. 8, pp. 127–130.
- Haessler, R. W. and Sweeney, P. E. 1991, "Cutting stock problems and solution procedures," *European Journal of Operational Research*, Vol. 54, No. 2, pp. 141–150.
- Hinxman, A. I. 1979, "The trim-loss and assortment problems: A survey," *European Journal of Operational Research*, Vol. 5, No. 1, pp. 8–18.
- Ho, J. C. and Chang, Y. 1991, "Heuristics for minimizing mean tardiness for m parallel machines," *Naval Research Logistics*, Vol. 38, No. 3, pp. 367–381.

- Hoffman, T. 1996, "AI based software models help cut production costs," *Computer World*, September 2, pp. 59–60, <http://www.computerworld.com/home/print9497.nsf/all/SL35paper>.
- Holland, J. H. 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan.
- Holsenback, J. E. and Russell, R. M. 1992, "A heuristic algorithm for sequencing on one machine to minimize total tardiness," *Journal of the Operational Research Society*, Vol. 43, No. 1, pp. 53–62.
- Huberman, B.; Lukose, R.; and Hogg, T. 1997, "An economics approach to hard computational problems," *Science*, Vol. 275, No. 5296, pp. 51–54.
- Keskinocak, P.; Wu, F.; Goodwin, R. T.; Murthy, S.; Akkiraju, R.; Kumaran, S.; and Derebail, A. 1998, "Scheduling solutions for the paper industry," IBM research report RC21297 (94999).
- Koulamas, C. 1994, "The total tardiness problem: Review and extensions," *Operations Research*, Vol. 42, No. 6, pp. 1025–1041.
- Lee, H. S.; Murthy, S.; Haider, S. W.; and Morse, D. 1996, "Primary production scheduling at steel-making industries," *IBM Journal of Research and Development*, Vol. 40, No. 2, pp. 231–252.
- Lee, Y. H.; Bhaskaran, K.; and Pinedo, M. 1997, "A heuristic to minimize the total weighted tardiness with sequence dependent setups," *IIE Transactions*, Vol. 29, No. 1, pp. 45–52.
- Lee, Y. H. and Pinedo, M. 1997, "Scheduling jobs on parallel machines with sequence dependent setup times," *European Journal of Operational Research*, Vol. 100, No. 3, pp. 464–474.
- Liu, J. S. and Sycara, K. P. 1996, "Multiagent coordination in tightly coupled task scheduling," *Proceedings of the Second International Conference on Multiagent Systems (ICMAS '96)*, Kyoto, Japan, December.
- Martello, S. and Toth, P. 1990, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley and Sons, Chichester, United Kingdom.
- Murthy, S. 1992, "Synergy in cooperating agents: Designing manipulators from task specifications," PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Numao, M. and Morishita, S. 1991, "Cooperative scheduling and its application to steel-making processes," *IEEE Transactions on Industrial Electronics*, Vol. 38, No. 2, pp. 150–155.
- Panwalkar, S. S.; Smith, M. L.; and Koulamas, C. P. 1993, "A heuristic for the single machine tardiness problem," *European Journal of Operational Research*, Vol. 70, No. 3, pp. 304–310.
- Pickard, G. C. 1997, "Paper machine scheduling yields gains at James River Naheola Mill," *Pulp and Paper*, Vol. 71, No. 9, pp. 125–129.
- Pierce, J. F. 1964, *Some Large-Scale Production Scheduling Problems in the Paper Industry*, Prentice Hall, Englewood Cliffs, New Jersey.
- Reddy, R. 1996, "The challenge of artificial intelligence," *IEEE Computer*, Vol. 29, No. 10, pp. 86–98.
- Salman, F. S.; Kalagnanam, J.; and Murthy, S. 1997, "Cooperative strategies for solving the bicriteria sparse multiple knapsack problem," IBM research report RC 21059 (94164).
- Shaw, M. 1998, "Madison streamlines business processes with integrated information system," *Pulp and Paper*, Vol. 72, No. 5, pp. 73–81.
- Smith, F. S.; Ow, P. S.; Muscettola, N.; Potwin, J.; and Matthys, D. C. 1990, "An integrated framework for generating and revising factory schedules," *Journal of the Operational Research Society*, Vol. 41, No. 6, pp. 539–552.
- Stadtler, H. 1990, "A one-dimensional cutting stock problem in the aluminum industry and its solution," *European Journal of Operational Research*, Vol. 44, pp. 209–223.
- Steuer, R. E. 1989, *Multiple Criteria Optimization: Theory, Computation, and Applications*, Robert E. Krieger Publishing, Malabar, Florida.
- Swaminathan, J. M.; Smith, S. F.; and Sadeh, N. M. 1996, "A multi-agent framework for modeling supply chain dynamics," *Proceedings of the NSF Research Planning Workshop on Artificial Intelligence and Manufacturing*, Albuquerque, New Mexico, June.
- Talukdar, S. N.; Pyo, S. S.; and Mehrotra, R. 1983, "Distributed processors for numerically intense problems," Final report for Electric Power Research Institute (EPRI) Project, RP 1983-1764-3.
- Talukdar, S. N.; deSouza, P.; and Murthy, S. 1993, "Organizations for computer-based

- agents," *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, Vol. 1, No. 2, pp. 75–87.
- Tsen, C. K. 1995, "Solving train scheduling problems using A-teams," PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Vahrenkamp, R. 1992, "Random search in the one-dimensional cutting stock problem," *European Journal of Operational Research*, Vol. 95, pp. 191–200.
- Wagner, H. M. 1969, *Principles of Operations Research, with Applications to Managerial Decisions*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Wascher, G. 1990, "An LP-based approach to cutting stock problems with multiple objectives," *European Journal of Operational Research*, Vol. 44, pp. 175–184.
- Yu, P. L. 1985, *Multiple-Criteria Decision Making*, Plenum Press, New York.