MICRO-BOSS:
A MICRO-OPPORTUNISTIC FACTORY SCHEDULER

Norman Sadeh

CMU-RI-TR-91-22

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

November 1991

Also to appear in *"Expert Systems with Applications", special issue on "Scheduling Expert Systems and Their Performances".*

This research was supported, in part, by the Defense Advanced Research Projects Agency under contract #F30602-88-C-0001, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation.

Table of Contents

List of Figures

List of Tables

Abstract

Recent research in factory scheduling has demonstrated the benefits of building schedules by first optimizing the sequencing of bottleneck machines, namely machines whose utilizations are expected to be particularly high. Within this approach, two scheduling perspectives are generally adopted: a resource-centered perspective is used to help maximize the utilization of bottleneck machines and a job-centered perspective is later used to compactly complete each job schedule (i.e. reduce work-in-process inventory). Because new secondary bottlenecks may arise during the construction of the schedule, recent scheduling systems have been designed with an ability to switch back and forth between their resource-centered scheduling perspective and their job-centered scheduling perspective. This ability to dynamically revise the current scheduling strategy has been termed *opportunistic scheduling*. However, because these schedulers require scheduling large resource subproblems or large job subproblems before revising their scheduling strategy, we refer to them as *macro-opportunistic schedulers*. Instead, this paper describes MICRO-BOSS, a so-called *micro-opportunistic scheduler* that can revise its scheduling strategy each time an operation is scheduled. Experimental results suggest that the extra flexibility of a micro-opportunistic approach to scheduling often translates into important reductions in schedule costs.

Keywords: Production Scheduling, Bottleneck Analysis, Heuristic Search, Constraint Satisfaction Problems, Combinatorial Optimization, Probabilistic Reasoning, Intelligent Decision Support Systems.

1 Introduction

In an economy where competition continuously intensifies, the need for cost-efficient production scheduling is becoming more critical every day. Current production scheduling techniques have largely failed to provide such cost-efficient solutions because of their inability to account for the diversity of constraints, costs, and preferences typically encountered in the manufacturing domain. Numerous techniques have been developed that focus on meeting due dates without paying attention to inventory costs, or attempt to maximize machine utilization without taking care of meeting due dates. This paper presents an approach to scheduling that can explicitly account for a variety of costs, including tardiness costs, in-process inventory costs, and finished-goods inventory costs. These costs are used to continuously update demand profiles that reflect contention between unscheduled jobs for the allocation of machines over time. By closely monitoring the evolution of bottlenecks during the construction of the schedule, and continuously redirecting the scheduling effort toward the optimization of the bottlenecks that currently appear most critical, this new scheduling approach has yielded significant reductions in schedule costs.

The job shop scheduling problem requires scheduling a set of jobs on a finite set of resources (e.g. machines, human operators, etc.). Each job is a request for the scheduling of a set of operations according to a process routing that specifies a partial ordering among these operations, along with their resource requirements. Operations are atomic: once started they cannot be interrupted. This paper is concerned with the design of a factory scheduler that builds a detailed schedule for the jobs to be produced over a planning horizon that ranges from a couple of days to a couple of weeks. The jobs to be scheduled are provided by a master-scheduling module [20] along with due dates, earliest acceptable release dates, marginal tardiness costs, marginal in-process inventory costs (e.g. interests on raw material costs, marginal direct holding costs, interests on processing costs, etc.), and marginal finished-goods inventory costs.

Job shop scheduling belongs to the class of NP-complete problems [7]. At the exception of a couple of one-, two-, and three- machine job shop scheduling problems, for which there exist efficient algorithms [17], all attempts to guarantee an optimal solution have failed. Instead job shop scheduling problems have traditionally been solved using priority dispatch rules [2, 16, 6]. These are local decision rules of the greedy type that build schedules via a forward simulation of the shop. Because these rules lack a global view of the shop, they usually build up large amounts of inventory in front of bottleneck machines.

More recently, with the advent of more powerful computers, a couple of more sophisticated scheduling techniques have been developed [8, 12, 1, 13, 11]. The first and by far most publicized of these techniques is the one developed by Goldratt and his colleagues within the context of the OPT factory scheduling system [8, 9, 5]. Among other things, this system emphasized the need to distinguish between bottleneck and non-bottleneck machines [9, 5]. In OPT, bottlenecks drive the entire schedule as they determine the throughput of the plant. More specifically, a module called SERVE produces an initial infinite capacity schedule by working

backwards from the job due dates. This initial schedule helps detect potential bottlenecks. The OPT module itself is then called upon to generate a forward finite capacity schedule that optimizes the utilization of these bottlenecks. The resulting bottleneck schedules are passed back to the SERVE module, which schedules the non-bottleneck operations while trying to minimize inventory. The distinction between bottleneck and non-bottleneck machines was pushed one step further in the OPIS system [21, 13], as it was recognized that new bottlenecks can appear during the construction of the schedule. The OPIS scheduler combines two scheduling perspectives: a resource-centered perspective for scheduling bottleneck resources, and a job-centered perspective to schedule non-bottleneck operations on a job by job basis [4]. Rather than relying on its initial bottleneck analysis, OPIS typically repeats this analysis each time a resource or a job has been scheduled. This ability to detect the emergence of new bottlenecks during the construction of the schedule and revise the current scheduling strategy has been termed *opportunistic scheduling* [13]. Nevertheless, the opportunism in this approach remains limited in the sense that it typically requires scheduling an entire bottleneck (or at least a large chunk of it) before being able to switch to another one. For this reason, such scheduling techniques should in fact be called *macro-opportunistic*.

In reality, variations in the job mix often cause bottlenecks to wander over time. Bottlenecks also tend to shift or disappear before being entirely scheduled. A scheduler that can only schedule large resource/job subproblems will not be able to take such variations into account. It will overconstrain its set of alternatives before having worked on the subproblems that will most critically determine the quality of the entire schedule. This in turn will often result in poorer solutions. A more flexible approach would allow to quit scheduling a resource as soon as another resource is identified as being more constraining. In fact, in the presence of multiple bottlenecks, one can imagine a technique that constantly shifts attention from one bottleneck to another rather than focusing on the optimization of a single bottleneck at the expense of others. This paper presents a more flexible approach to scheduling, or a *micro-opportunistic* approach, in which the evolution of bottlenecks is continuously monitored during the construction of the schedule, and the problem solving effort constantly redirected towards the most serious bottleneck. In its simplest form, this micro-opportunistic approach results in an *operation-centered* view of scheduling, in which each operation is considered an independent decision point and can be scheduled without requiring that other operations using the same resource or belonging to the same job be scheduled at the same time[1].

Section 2 of this paper gives a formal definition of the job shop scheduling problem studied in this paper. Section 3 describes a micro-opportunistic factory scheduling system called **MICRO-BOSS** (MICRO-BOttleneck Scheduling System), concentrating on the heuristics to

---

[1]An alternative approach in which resources can be resequenced to adjust for resource schedules built further down the road is described in [1, 3]. This approach has been very successful at minimizing makespan. Attempts to generalize the procedure to account for due dates seem to have been less successful so far [19]. It should also be pointed out that the idea of continuously reoptimizing the current partial schedule is not incompatible with the micro-opportunistic approach.

identify critical operations and promising reservations for these operations. Section 4 describes an empirical study that compares MICRO-BOSS against several other scheduling techniques. Conclusions are presented in section 5.

2 The Job Shop Model

Formally, the problem requires scheduling a set of jobs $J=\{j_1,...,j_n\}$ on a set of physical resources $RES=\{R_1,...,R_m\}$. Each job j_l consists of a set of operations $O^l=\{O^l_1,...,O^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g. O^l_i BEFORE O^l_j).

Additionally a job j_l has an earliest acceptable release date, erd_l, a due-date, dd_l, and a latest acceptable completion date, lcd_l. $lcd_l \geq dd_l \geq erd_l$. All jobs need to be scheduled between their earliest acceptable release date and latest acceptable completion date. The earliest acceptable release date may correspond to the earliest possible arrival date of raw materials or to a rough release date provided by a master scheduling module. It is assumed that the actual release date will be determined by the schedule to be constructed. The latest acceptable completion date may correspond to a date after which the customer will refuse delivery. If such a date does not actually exist, it can always be chosen far enough in the future so that it is no longer a constraint.

Each operation O^l_i has a fixed duration, du^l_i, and a start time, st^l_i (to be determined), whose domain of possible values is delimited by an earliest start time, est^l_i, and a latest start time, lst^l_i (initially derived from the job's earliest acceptable release date erd_l and latest acceptable completion date lcd_l). In order to be successfully executed, each operation O^l_i requires a resource R^l_i (e.g. $R^l_i=R_1$, a milling machine).

COSTS:

Each job j_l has:

• a **marginal tardiness cost**, $tard_l$: the cost incurred for each unit of time that the job is tardy (i.e. finishes past its due date). Marginal tardiness costs generally include tardiness penalties, interests on delayed profits, loss of customer goodwill, etc[2]. The total tardiness cost of job j_l, in a given schedule, is:

$$TARD_l = tard_l \times Max(0, C_l - dd_l) \quad (1)$$

where $C_l = st^l_{n_l} + du^l_{n_l}$ is the completion date of job j_l in that schedule, assuming that $O^l_{n_l}$ is the last operation in job j_l.

• **marginal in-process and finished-goods inventory costs**: In our model, each operation O^l_i can incrementally introduce its own non-negative marginal inventory cost, inv^l_i. Typically the first operation in a job introduces marginal inventory costs that correspond to interests on the costs of raw materials, interests on processing costs (for that first operation), and marginal holding costs.

---

[2]In this model, inventory costs incurred past the due date are not accounted for in the tardiness costs. Instead they are accounted for in the inventory costs described below.

Downstream operations[3] introduce additional marginal inventory costs such as interests on processing costs or interests on the costs of additional raw materials required by these operations. The total inventory cost for a job j_l, in a given schedule, is:

$$INV_l = \sum_{i=1}^{n_l} inv_i^l \times [Max(C_l, dd_l) - st_i^l] \tag{2}$$

The total cost of a schedule is obtained by summing the cost of each job schedule:

$$Schedule\ Cost = \sum_{l=1}^{n} (TARD_l + INV_l)$$

The goal of the scheduler is to produce a feasible schedule that reduces this total cost as much as possible.

[3]An operation O_i^k is said to be downstream (upstream) of another operation O_j^k within the same job if O_i^k is a direct or indirect successor (predecessor) of O_j^k in that job, as defined by its process routing.

EXAMPLE:

Figure 1 depicts a small job shop problem with 4 jobs. Each square box represents an operation. Each box is labeled with the name of the operation that it represents (e.g. O_1^1), the duration of that operation (e.g. $du_1^1 = 2$), and its resource requirement (e.g. $R_1^1 = R_1$). The arrows represent precedence constraints. For instance, job j_1 requires 5 operations O_1^1, O_2^1,..., O_5^1. O_1^1 has to be performed before O_2^1, O_2^1 before O_4^1, etc. The other arcs in the graph represent capacity constraints, which require each resource to be allocated to only one operation at a time. There is a capacity constraint between each pair of operations that require the same resource. Notice that R_2 is the only resource required by four operations (one from each job). Notice also that, in three out of four jobs (namely j_1 , j_3 , and j_4), the operation requiring R_2 is one of the job's longest operations. Consequently, resource R_2 can be expected to be the main bottleneck of the problem. We will see that, to some extent, resource R_1 constitutes a secondary bottleneck.

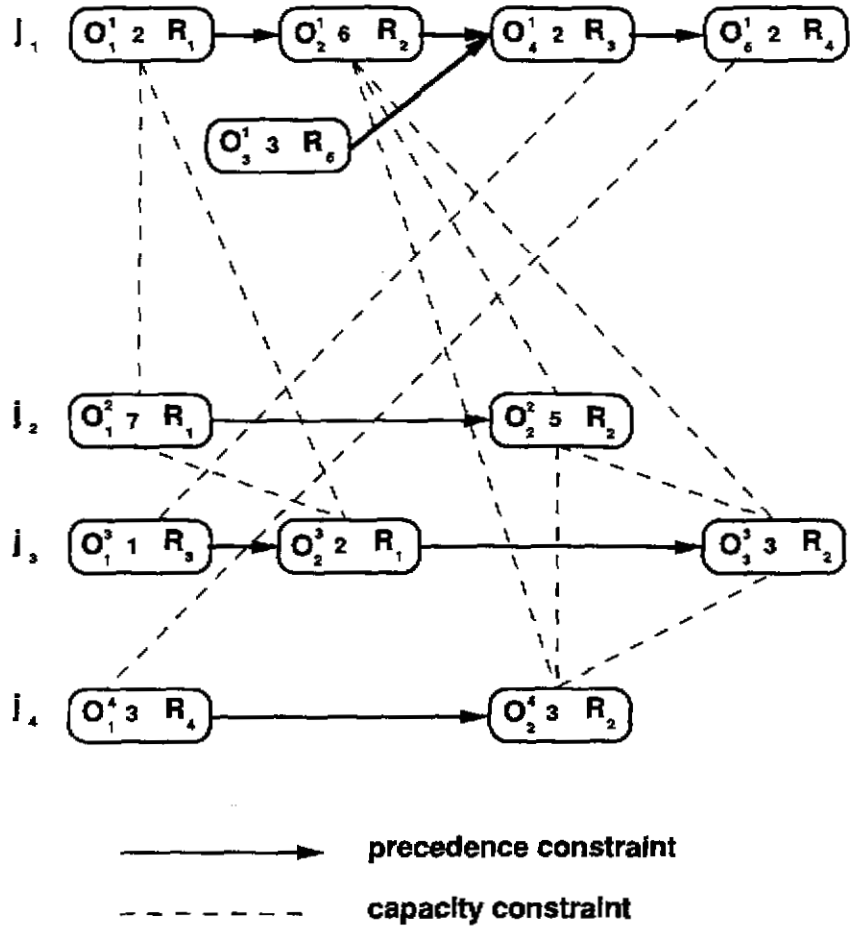

Figure 1: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents, its duration, and the resource that it requires.

The earliest acceptable release dates, due dates, and latest acceptable completion dates of the

jobs are provided in Table 1 along with the marginal tardiness and inventory costs of these jobs.

Earliest acceptable release dates, due dates, latest acceptable completion dates, and costs									
Job j_l	erd_l	dd_l	lcd_l	$tard_l$	inv_1^l	inv_2^l	inv_3^l	inv_4^l	inv_5^l
j_1	0	12	20	20	2	1	2	0	0
j_2	0	14	20	20	5	0	-	-	-
j_3	0	9	20	5	1	0	0	-	-
j_4	0	18	20	10	1	0	-	-	-

Table 1: Earliest acceptable release dates, due dates, latest acceptable completion dates and marginal costs.

3 A Micro-opportunistic Approach

In MICRO-BOSS, each operation is considered an independent decision point. Any operation can be scheduled at any time, if deemed appropriate by the system. There is no obligation to simultaneously schedule other operations upstream or downstream within the same job, nor is there any obligation to schedule other operations competing for the same resource.

MICRO-BOSS proceeds by iteratively selecting an operation to be scheduled and a reservation (i.e. start time) to be assigned to that operation. Every time an operation is scheduled, a new *search state* is created, where new constraints are added to account for the reservation assigned to that operation. A so-called consistency enforcing procedure is applied to the new state, that updates the set of remaining possible reservations of each unscheduled operation. If an unscheduled operation is found to have no possible reservations left, a *deadend state* has been reached: the system needs to *backtrack* (i.e. it needs to undo some earlier reservation assignments in order to be able to complete the schedule). If the search state does not appear to be a deadend, the system moves on and looks for a new operation to schedule and a reservation to assign to that operation.

In MICRO-BOSS, search efficiency is maintained at a high level by interleaving search with the application of consistency enforcing techniques and a set of look-ahead techniques that help decide which operation to schedule next (so-called *operation ordering heuristic*) and which reservation to assign to that operation (so-called reservation ordering heuristic).

1. **Consistency Enforcing (or Consistency Checking):** Consistency enforcing techniques prune the search space by inferring new constraints resulting from earlier reservation assignments [10, 18]. By constantly accounting for earlier scheduling decisions, these techniques reduce the chance of backtracking. Simultaneously, by allowing for the early detection of deadend states, these techniques limit the amount of work wasted in the exploration of fruitless alternatives.

2. **Look-ahead Analysis:** A two-step look-ahead procedure is applied in each search state, which first optimizes reservation assignments within each job, and then, for each resource, computes contention between jobs over time. Resource/time intervals where job contention is the highest help identify the critical operation to be scheduled next (operation ordering heuristic). Reservations for that operation are then ranked according to their ability to minimize the costs incurred by the conflicting jobs (reservation ordering heuristic). By constantly redirecting its effort towards the most serious conflicts, the system is able to build schedules that are closer to the global optimum. Simultaneously, because the scheduling strategy is aimed at reducing job contention as fast as possible, chances of backtracking tend

to subside pretty fast too.

The so-called opportunism in MICRO-BOSS results from the ability of the system to constantly *revise its search strategy and redirect its effort towards the scheduling of the operation that appears to be the most critical in the current search state.* This degree of opportunism differs from the one displayed by earlier approaches where scheduling entities were large resource/job subproblems [13], i.e. where large resource/job subproblems had to be scheduled before the system could revise its scheduling strategy.

Concretely, MICRO-BOSS starts in a search state in which no operation has been scheduled yet, and proceeds according to the following steps:

1. If all operations have been scheduled then stop, else go on to 2;
2. Apply the **consistency enforcing** procedure;
3. If a deadend is detected then **backtrack**, else go on to step 4;
4. Perform the **look-ahead** analysis: rank the possible reservations of each unscheduled operation according to how well they minimize the costs of the job to which the operation belongs (step 1), and evaluate job contention over time for each resource (step 2);
5. Select the next operation to be scheduled (so-called **operation ordering** heuristic);
6. Select a reservation for that operation (so-called **reservation ordering** heuristic)
7. Create a **new search state** by adding the new reservation assignment to the current partial schedule. Go back to 1.

The remainder of this section gives a more detailed description of the look-ahead analysis and the operation/reservation ordering heuristics. Further details on these techniques as well as other aspects of MICRO-BOSS can be found in [18].

3.1 Look-ahead Analysis in MICRO-BOSS

3.1.1 Optimizing Critical Conflicts First

If all jobs could be scheduled optimally (i.e. just-in-time), there would be no scheduling problem. Generally, this is not the case. Jobs typically have conflicting resource requirements. The look-ahead analysis carried out by MICRO-BOSS in each search state is meant to allow the scheduling system to focus its effort on those conflicts that currently appear most critical. A critical conflict is one that will require an important tradeoff, namely a tradeoff that will significantly impact the quality of the *entire* schedule. By first focusing on critical conflicts, MICRO-BOSS ensures that it has as many options as possible to optimize these conflicts. As illustrated by a trace provided at the end of this paper, once critical tradeoffs have been worked out, the remaining unscheduled operations tend to become more decoupled and hence easier to optimize[4]. As contention subsides, so does the chance of backtracking. In other words, by constantly redirecting search towards those tradeoffs that appear most critical, MICRO-BOSS is expected to produce better schedules and simultaneously reduce its chances of backtracking.

More specifically, a two-step look-ahead procedure is applied to each search state, which first optimizes reservation assignments within each job, and then, for each resource, computes contention between jobs over time. The so-called *demand profiles* produced by these computations help identify operations whose good reservations (within their jobs) conflict with the good reservations of other operations. These operations define the critical conflicts on which MICRO-BOSS works first.

The two-step look-ahead analysis carried out in MICRO-BOSS is further detailed below.

3.1.2 Step 1: Reservation Optimization Within a Job

In order to detect critical conflicts between the resource requirements of unscheduled operations, MICRO-BOSS keeps track of the best start times that remain available to each unscheduled operation within its job, and the marginal costs that would be incurred by the job if the operation was allocated another start time. More specifically, for each remaining possible start time τ of each unscheduled operation O_i^k, MICRO-BOSS (implicitly) keeps track of the minimum additional costs, $mincost_i^k(\tau)$, that would be incurred by job j_k (the job to which O_i^k belongs), if O_i^k was to start at $st_i^k=\tau$ rather than at one of its best remaining possible start times. By definition, if $st_i^k=\tau$ is one of the best start times that remain available to O_i^k within its job, $mincost_i^k(\tau)=0$.

Mincost functions are implicitly updated in each search state in order to account for earlier scheduling decisions. This is done by:

- identifying whether the choice of a start time for a given unscheduled operation still

---

[4]This generalizes the idea that a bottleneck schedule should be built first in order to drive other scheduling decisions, such as release decisions.

affects the tardiness costs of the job to which that operation belongs, and

- determining which inventory costs, if any, are still affected by that choice.

Once the *apparent* marginal tardiness and inventory costs of an unscheduled operation have been determined in the current search state, the system can identify which start time(s) among the ones that are still possible will be optimal for that operation (within its job). At that point, the system can also determine the minimum costs incurred by the job if suboptimal start times are instead selected. Efficient procedures to keep track of changes in apparent costs as the system moves from one search state to another are described in [18].

3.1.3 Step 2: Building Demand Profiles to Identify Highly Contended Resource/Time Intervals

In MICRO-BOSS, critical conflicts are identified as groups of operations whose good start times (within their jobs) conflict with each other. The importance of a conflict (and the criticality of the operations participating in that conflict) depends on the number of jobs that are competing for the same resource, the amount of temporal overlap between the requirements of these jobs, the number of alternative reservations (i.e. start times) still available to the conflicting operations and the differences in cost between these alternative reservations (as determined by the *mincost* functions computed in step 1).

In order to identify critical conflicts, MICRO-BOSS uses a probabilistic framework, in which each remaining possible start time τ of an unscheduled operation O_i^l is assigned a *subjective probability* $\sigma_i^l(\tau)$ to be selected for that operation. Possible start times with lower *mincost* values are simply assigned a larger probability, thereby reflecting our expectation that they will allow for the production of better schedules. Using these start time distributions, the system builds, for each unscheduled operation O_i^l, an *individual demand profile* $D_i^l(t)$ that indicates the subjective probability that the operation will be requiring its resource as a function of time (i.e. also a measure of the reliance of the operation on the availability of its resource as a function of time). By aggregating the individual demand profiles of all unscheduled operations requiring a given resource, R_k, the system obtains an *aggregate demand profile*, $D_{R_k}^{aggr}(t)$, that indicates contention between (all) unscheduled operations for the resource as a function of time.

Figure 2 represents $\sigma_2^2(\tau)$, the start time distribution of operation O_2^2 in the problem defined in Figure 1. This start time distribution is depicted in the initial search state, where all operations still have to be scheduled. In this search state, start time $st_2^2=9$ is the best possible start time for O_2^2: it corresponds to a just-in-time schedule of job j_2. Later start times have a lower subjective probability as they would force the job to finish past its due date. Earlier start times are also suboptimal since they would produce additional inventory. In this example, the marginal tardiness cost of job j_2, $tard_2=20$, is five times larger than the marginal inventory cost introduced by operation O_1^2, $inv_1^2=5$. Accordingly $\sigma_2^2(\tau)$ has a steeper slope for $\tau > 9$ than for $\tau < 9$. Additional details on how these distributions are constructed can be found in [18].

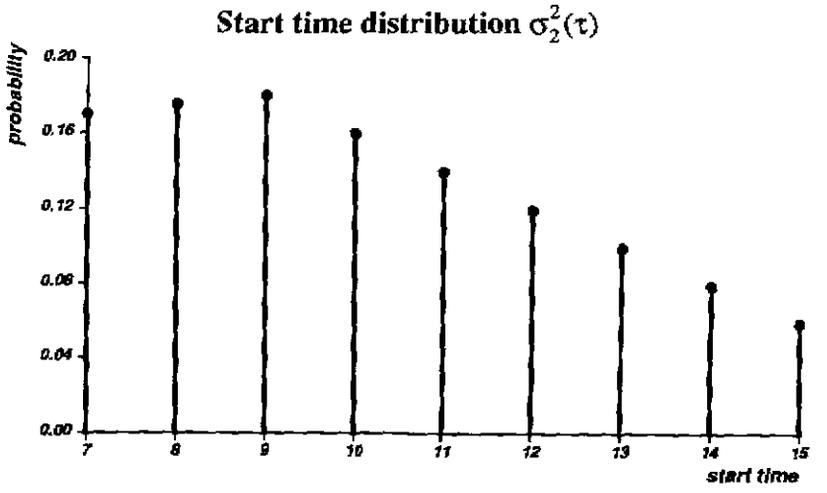

Figure 2: Start time distribution $\sigma_2^2(\tau)$ for operation O_2^2 in the initial search state for the problem defined in Figure 1.

Figure 3 displays the individual demand profiles of the four operations requiring resource R_2. These demand profiles represent the subjective probability that each one of these operations uses resource R_2 as a function of time. The aggregate demand for resource R_2 is obtained by summing these four individual demands over time. The individual demands of operations O_3^3 and O_2^4 are quite uniform since these two operations have relatively low apparent marginal costs. In contrast, operations O_2^1 and O_2^2, which have larger apparent marginal costs, have individual demands that are concentrated around their best reservations.

Similar computations can be performed for each of the five resources in the problem. The resulting aggregate demands (in the initial search state) are displayed in Figure 4. As expected, resource R_2 appears to be the most contended for. The aggregate demand for that resource is well above 1.0 over a large time interval, with a peak at 1.49. Resource R_1 appears to be a potential bottleneck at the beginning of the problem, with a demand peaking at 1.20. Whether R_1 will actually be an auxiliary bottleneck or not cannot be directly determined from the curves displayed in Figure 4. Instead the system needs to update these curves in each search state to account for earlier decisions. It could be the case that, as operations requiring R_2 are scheduled, the aggregate demand for R_1 becomes smoother. In this example, this is not the case. On the contrary, after only a portion of the operations requiring resource R_2 have been scheduled, MICRO-BOSS will redirect its effort to the scheduling of resource R_1 (see section 4).

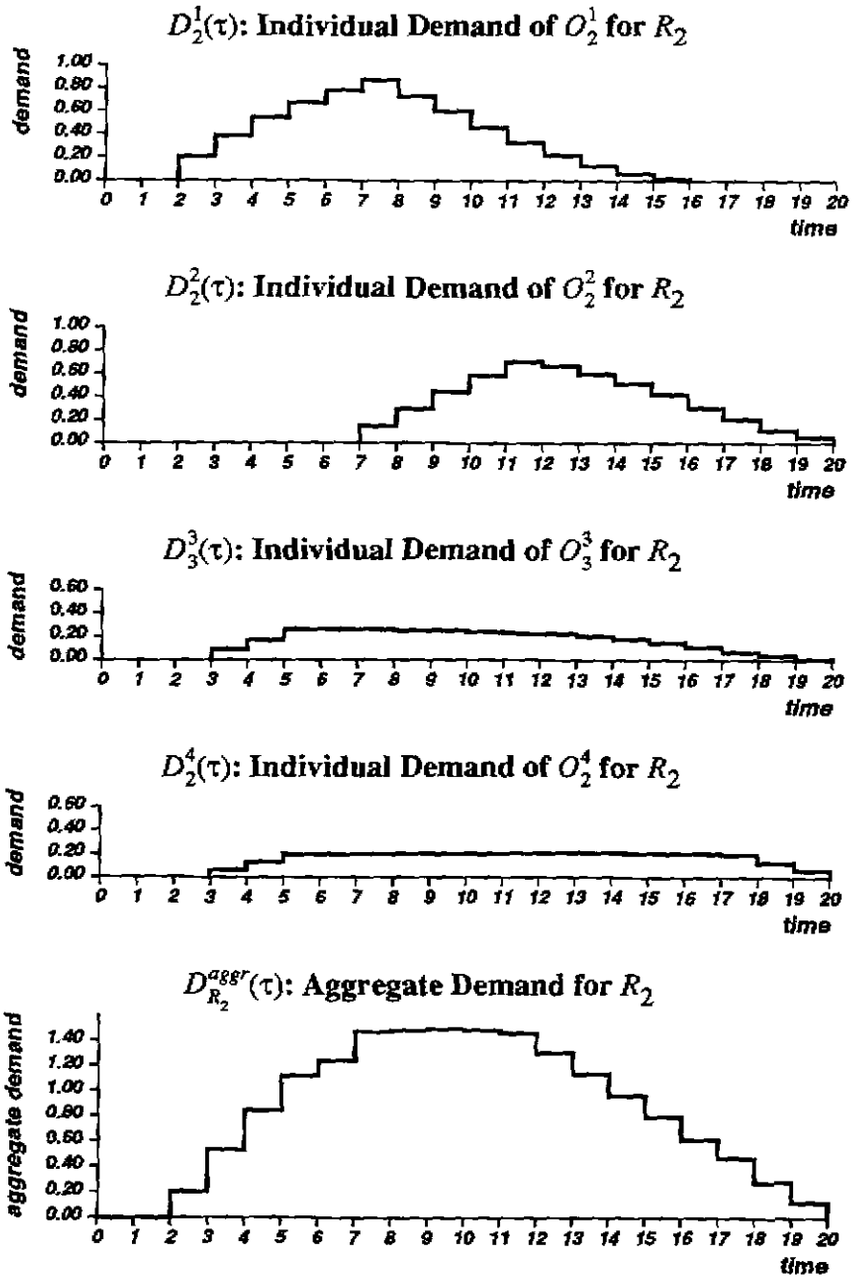

Figure 3: Building R_2's aggregate demand profile in the initial search state.

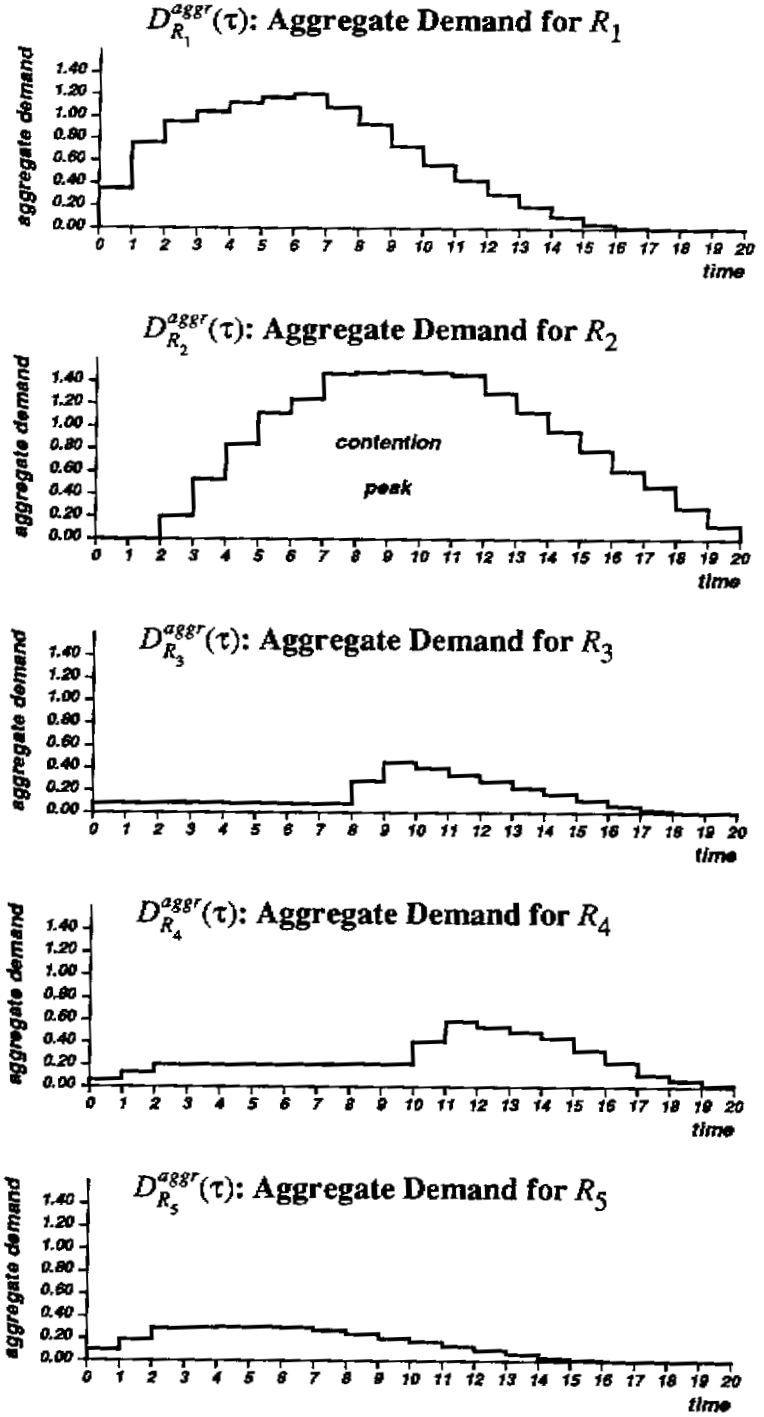

Figure 4: Aggregate demands in the initial search state for each of the five resources.

3.2 Operation Selection

Critical operations are identified as operations whose good reservations conflict with the good reservations of other operations. The largest peak in the aggregate demand profiles determines the next conflict (or micro-bottleneck) to be optimized, and the operation with the largest reliance on the availability of that peak (i.e. the operation with the largest individual contribution to the peak) is selected to be scheduled next. Indeed, this operation is the one whose good reservations are the most likely to become unavailable if other operations contending for the current micro-bottleneck were scheduled first.

In the example introduced earlier, the most contended demand peak is the one for resource R_2 over interval [7,12[. Figure 5 displays the aggregate demand for resource R_2 together with the individual demands of the four operations requiring this resource. The operation with the largest contribution to the demand peak is O_2^1. Therefore this operation is selected to be scheduled next. This is no real surprise: O_2^1 belongs to one of the two jobs in the problem that have a high marginal tardiness cost ($tard_1 = 20$). While any delay in starting job j_1 will cause this job to be late, job j_2 (i.e. the other job with a high marginal tardiness cost) can tolerate a small amount of delay without ending up late.

3.3 Reservation Selection

Once it has selected an operation, MICRO-BOSS attempts to identify a reservation (for that operation) that will reduce as much as possible the costs incurred by the job to which that operation belongs and the other competing jobs. This is equivalent to solving a one-machine early/tardy problem in which operations scheduled past their best start times incur penalties determined by their apparent marginal tardiness costs, while operations scheduled before their best start times incur earliness penalties determined by their apparent marginal inventory costs (as determined by the *mincost* functions).

MICRO-BOSS uses a hybrid reservation ordering heuristic that adapts to the amount of contention for the critical resource/time interval. When contention is particularly high, MICRO-BOSS successively runs several variations of a one-machine early/tardy procedure developed by Ow and Morton [15, 18]. The one machine schedule that reduces most the costs of the jobs competing for the critical resource determines the reservation assigned to the critical operation. When contention is lower, the system dynamically switches to a greedy reservation ordering heuristic, in which reservations are simply rated according to their apparent costs (i.e. according to their *mincost* values). Indeed, in situations where contention is not too high, a sizable proportion of the good start times of non critical operations tend to remain available after more critical operations have been scheduled. When this is the case, a greedy reservation ordering tends to produce high quality solutions. In particular, it inserts idle time as required by the operation it is scheduling[5].

---

[5]See [18] for further details.

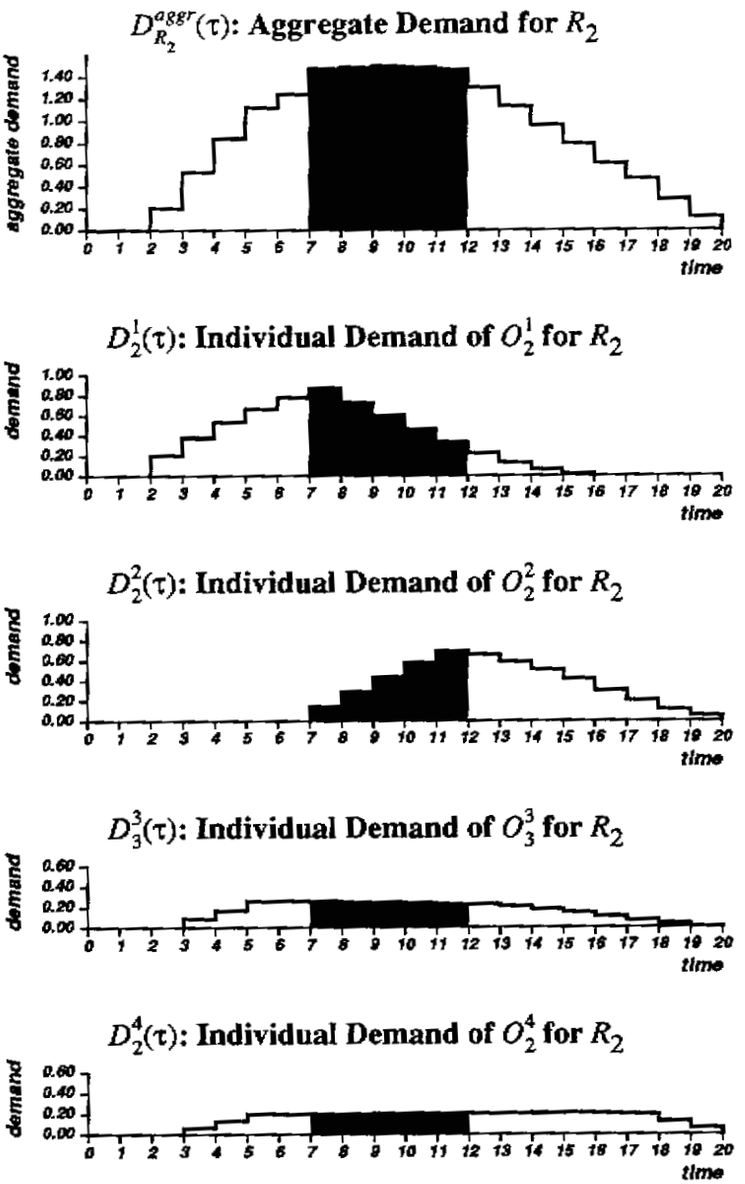

Figure 5: Operation selection in the initial search state.

4 A Small Example

The current version of MICRO-BOSS has been implemented in Knowledge Craft, a frame-based language that runs on top of Common Lisp. The system runs on a DECstation 5000 under Mach UNIX. The small example used throughout this paper requires a little over 1 second of CPU time in the current implementation. An edited trace of this example is given in Figure 6.

Observe that, rather than entirely scheduling the main bottleneck resource, namely resource R_2, MICRO-BOSS started to work on resource R_1 only after two out of the four operations requiring R_2 had been scheduled. The average expected demand displayed in each search state is the average demand for the critical demand peak, and the average contribution is that of the critical operation for the demand over that peak. The decoupling effect of the operation ordering heuristic is very clear in this example. In particular, the average demand over the critical peak consistently decreases from one search state to the next, thereby indicating a regular decrease in contention as the schedule is constructed (remember that the demand peak corresponds to the interval of highest contention in the current search state). This observation is correlated by the average contribution of the critical operation to the demand for the peak in each search state. As the schedule is constructed, the contribution of the critical operation to the peak becomes a larger proportion of the total demand for that peak. This indicates that there are fewer and fewer operations contending with each other. After half of the operations have been scheduled (depth 7), contention has totally disappeared: the critical operation is the only one to contribute to the demand for the peak. In other words, the problem has been totally decoupled. The resource requirements of the operations that still need to be scheduled no longer interact with each other. This phenomenon is not specific to this example, but can be observed in all the problems that we have run. This suggests that the operation ordering heuristic implemented in MICRO-BOSS is indeed very effective at redirecting search towards the most serious conflicts.

Notice also that no backtracking was necessary to schedule this problem. The resulting schedule is displayed in Figure 7.

```
MON NOV 12 1990 --- 17:04:49 EST

>> Depth: 0, Number of states visited: 0
   Critical demand peak:
   R2 between 7 and 12, Avg. expected demand: 1.48
   Critical Operation: O2^1, Avg. contrib.: 0.60
   Using early/tardy reservation ordering heuristic:
   O2^1 scheduled between 2 and 8 on R2

>> Depth: 1, Number of states visited: 1
   Critical demand peak:
   R2 between 10 and 15, Avg. expected demand: 1.33
   Critical Operation: O2^2, Avg. contrib.: 0.64
   Using early/tardy reservation ordering heuristic:
   O2^2 scheduled between 9 and 14 on R2

>> Depth: 2, Number of states visited: 2
   Critical demand peak:
   R1 between 0 and 4, Avg. expected demand: 1.35
   Critical Operation: O1^2, Avg. contrib.: 0.75
   Using early/tardy reservation ordering heuristic:
   O1^2 scheduled between 2 and 9 on R1

>> Depth: 3, Number of states visited: 3
   Critical demand peak:
   R2 between 14 and 19, Avg. expected demand: 1.13
   Critical Operation: O3^3, Avg. contrib.: 0.58
   Using early/tardy reservation ordering heuristic:
   O3^3 scheduled between 17 and 20 on R2

>> Depth: 4, Number of states visited: 4
   Critical demand peak:
   R2 between 14 and 19, Avg. expected demand: 0.60
   Critical Operation: O2^4, Avg. contrib.: 0.60
   Using greedy reservation ordering heuristic:
   O2^4 scheduled between 14 and 17 on R2
```

Figure 6: An edited trace

>> Depth: 5, Number of states visited: 5
Critical demand peak:
R_4 between 10 and 13, Avg. expected demand: 0.57
Critical Operation: O_5^1, Avg. contrib.: 0.34
Using greedy reservation ordering heuristic:
O_5^1 scheduled between 10 and 12 on R_4

>> Depth: 6, Number of states visited: 6
Critical demand peak:
R_3 between 8 and 10, Avg. expected demand: 1.08
Critical Operation: O_4^1, Avg. contrib.: 1.0
Using greedy reservation ordering heuristic:
O_4^1 scheduled between 8 and 10 on R_3

>> Depth: 7, Number of states visited: 7
Critical demand peak:
R_5 between 4 and 7, Avg. expected demand: 0.55
Critical Operation: O_3^1, Avg. contrib.: 0.55
Using greedy reservation ordering heuristic:
O_3^1 scheduled between 5 and 8 on R_5

>> Depth: 8, Number of states visited: 8
Critical demand peak:
R_1 between 0 and 4, Avg. expected demand: 0.50
Critical Operation: O_1^1, Avg. contrib.: 0.50
Using greedy reservation ordering heuristic:
O_1^1 scheduled between 0 and 2 on R_1

>> Depth: 9, Number of states visited: 9
Critical demand peak:
R_4 between 5 and 8, Avg. expected demand: 0.44
Critical Operation: O_1^4, Avg. contrib.: 0.44
Using greedy reservation ordering heuristic:
O_1^4 scheduled between 7 and 10 on R_4

Figure 6, continued

```
>> Depth: 10, Number of states visited: 10
   Critical demand peak:
   R1 between 12 and 16, Avg. expected demand: 0.31
   Critical Operation: O_2^3, Avg. contrib.: 0.31
   Using greedy reservation ordering heuristic:
   O_2^3 scheduled between 15 and 17 on R1

>> Depth: 11, Number of states visited: 11
   Critical demand peak:
   R3 between 13 and 15, Avg. expected demand: 0.14
   Critical Operation: O_1^3, Avg. contrib.: 0.14
   Using greedy reservation ordering heuristic:
   O_1^3 scheduled between 14 and 15 on R3

>> Depth: 12, Number of states visited: 12
   Schedule Completed
```

Figure 6, concluded

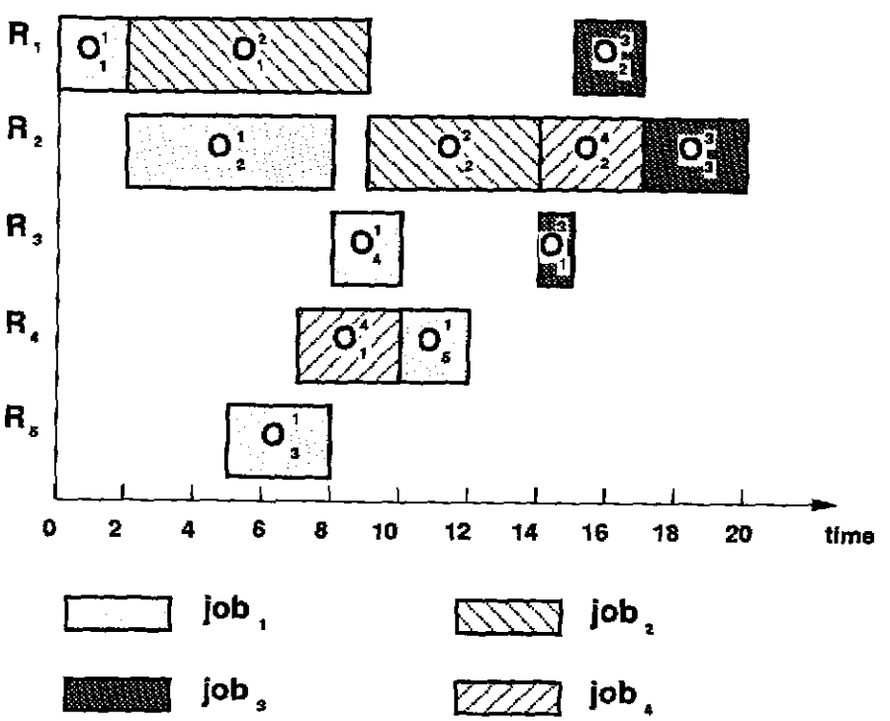

Figure 7: The final schedule produced by MICRO-BOSS.

5 Performance Evaluation

Several experimental comparisons were carried out in order to evaluate the performance of MICRO-BOSS. These comparisons were performed under a variety of scheduling conditions and different cost assumptions. They included comparisons against several combinations of popular priority dispatch rules and release policies, comparisons against a macro-opportunistic scheduler that could dynamically switch between a job-centered perspective and a resource-centered perspective, comparisons against coarser variations of MICRO-BOSS, and a comparison against a variation of MICRO-BOSS using unbiased demand profiles. This section briefly summarizes some of these experiments. A more comprehensive description of these experiments can be found in [18].

Problem Sets			
Number of Bottlenecks	Due Date Range	Avg. Due Date	Problem Set
1	loose	wide	**1**
1	loose	narrow	**2**
1	tight	wide	**3**
1	tight	narrow	**4**
2	loose	wide	**5**
2	loose	narrow	**6**
2	tight	wide	**7**
2	tight	narrow	**8**

Table 2: Characteristics of the eight problem sets.

Eight sets of scheduling problems were randomly generated to cover a wide variety of scheduling conditions. Scheduling conditions were varied by adjusting three parameters: an average due date parameter (tight/loose), a due date range parameter (narrow/wide), and a parameter controlling the number of major bottlenecks (in this case one or two). A set of 10 scheduling problems was randomly generated for each parameter combination (see Table 2), resulting in a total of 80 scheduling problems (10 problems x 2 average due date values x 2 due date ranges x 2 bottleneck configurations).

Each problem involved 20 jobs and 5 resources for a total of 100 operations. Marginal tardiness costs in the experiments presented in this paper were set to be *on the average* 5 times larger than marginal inventory costs in order to model a typical make-to-order production environment[6].

Figure 8 displays the average costs of the schedules produced by MICRO-BOSS and four priority dispatch rules that have been reported to be particularly good at reducing tardiness under

[6]Experiments under different cost assumptions are also reported in [18].

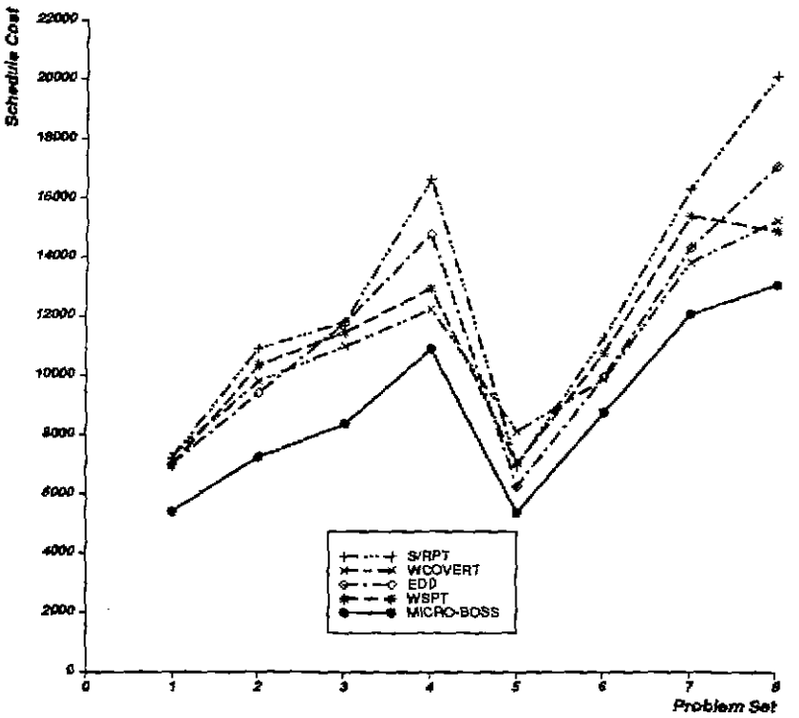

Figure 8: Comparison of the costs of schedules produced by MICRO-BOSS, the WSPT, EDD, S/RPT and WCOVERT dispatch rules. Each problem set includes 10 problems with 20 jobs and 5 machines.

different scheduling conditions [22]: the Weighted Shortest Processing Time (WSPT) rule, the Earliest Due Date (EDD) rule, the Slack per Remaining Processing Time (S/RPT) rule, and the Weighted Cost OVER Time (WCOVERT) parametric rule. In the experiments reported here, these four dispatch rules were coupled to the Average Queue Time release policy (AQT) described in [11]. AQT is a parametric release policy that estimates queuing time as a multiple of the average job duration. The release of a job is determined by offsetting its due date by the sum of the total job duration and the estimated queuing time. Look-ahead parameters in AQT and WCOVERT were optimized for each problem set.

Remarkably enough, MICRO-BOSS consistently outperformed all four dispatch rules under all eight conditions of the study. A more detailed analysis of the results indicates that, while performing at a level comparable to the dispatch rules with respect to tardiness, MICRO-BOSS yielded significant reductions in inventory (between 15 and 50 percent depending on the scheduling situation) [18]. The most important reductions in inventory were observed on the most difficult problems, namely those with tight average due dates and narrow due date ranges. Overall, MICRO-BOSS reduced the average schedule cost by 18% compared to its closest competitor, WCOVERT.

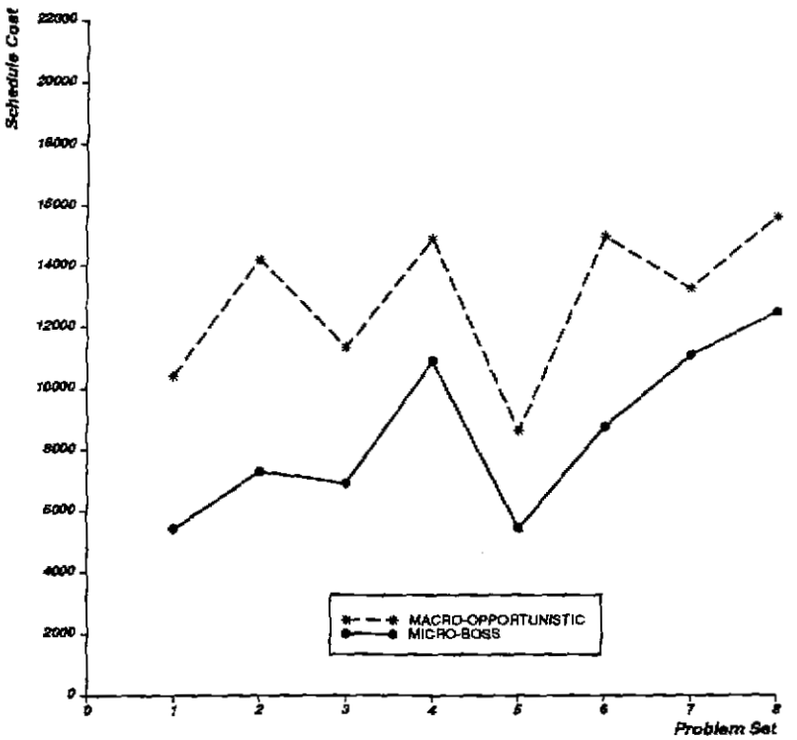

Figure 9: Comparison of the costs of schedules produced by MICRO-BOSS and the macro-opportunistic scheduler.
Each problem set includes 10 problems with 20 jobs and 5 machines.

MICRO-BOSS was also compared against a macro-opportunistic scheduler that dynamically combined both a resource-centered perspective and a job-centered perspective, like in the OPIS scheduling system [13]. However, while OPIS relies on a set of repair heuristics to recover from inconsistencies [14], the macro-opportunistic scheduler of this study was built to use the same consistency enforcing techniques and the same backtracking scheme as MICRO-BOSS[7]. The macro-opportunistic scheduler also used the same demand profiles as MICRO-BOSS. When average demand for the most critical resource/time interval was above some threshold level (a parameter of the system that was empirically adjusted), the macro-opportunistic scheduler focused on scheduling the operations requiring that resource/time interval, otherwise it used a job-centered perspective to identify a critical job and schedule some or all the operations in that job. Each time a resource/time interval or a portion of a job was scheduled, new demand profiles were computed to decide which scheduling perspective to use next[8].

---

[7]An alternative would have been to implement a variation of MICRO-BOSS using the same repair heuristics as OPIS. Besides being quite time-consuming to implement, such a comparison would have been affected by the quality of the specific repair heuristics currently implemented in the OPIS scheduler.

[8]Additional details on the implementation of the macro-opportunistic scheduler can be found in [18].

Figure 9 summarizes the results of the comparison between MICRO-BOSS and the macro-opportunistic scheduler[9]. The macro-opportunistic scheduler was consistently outperformed by MICRO-BOSS under all eight scheduling conditions. Additional performance measures indicate that the savings achieved by MICRO-BOSS correspond to reductions in tardiness (Figure 10), flowtime (i.e. work-in-process) (Figure 11) and in-system time (i.e. total inventory, including finished-goods inventory)(Figure 12). These results clearly indicate that the additional flexibility of a micro-opportunistic approach to scheduling generally yields important reductions in schedule costs.

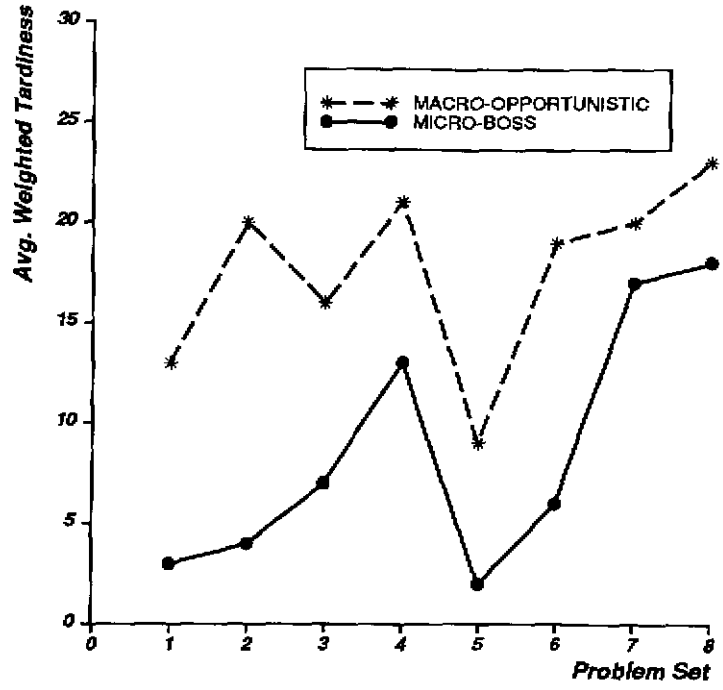

Figure 10: Tardiness performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

On most problems, MICRO-BOSS achieved a search efficiency of 100% (computed as the ratio of the number of operations to be scheduled over the number of search states that were visited), and required about 10 minutes of CPU time to schedule each problem (on a DECstation 5000). Preliminary experiments in C seem to indicate that this time could be reduced to under 1 minute, on the same machine.

[9]While MICRO-BOSS had no problem scheduling any of the 80 scheduling problems, the macro-opportunistic scheduler could only solve 69 problems out of 80 in less than 1,000 search states. The results presented here correspond to these 69 experiments.

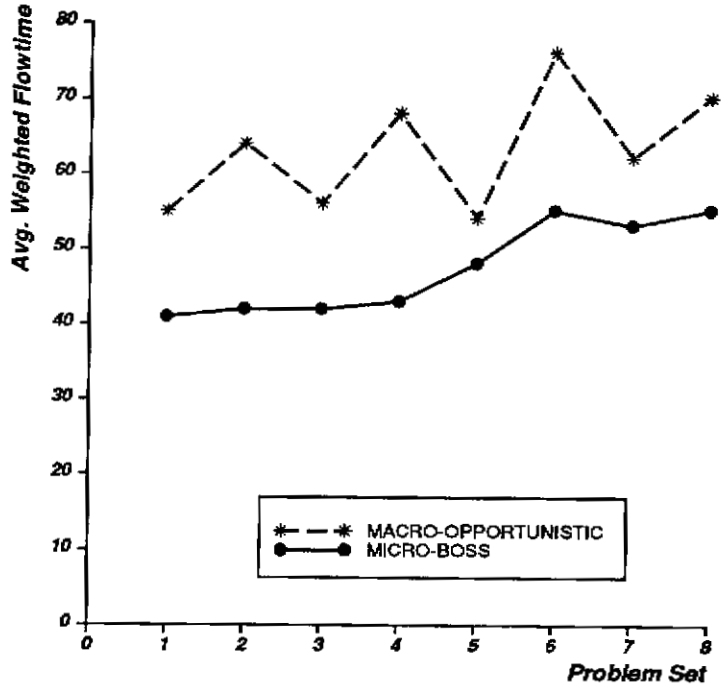

Figure 11: Flowtime performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

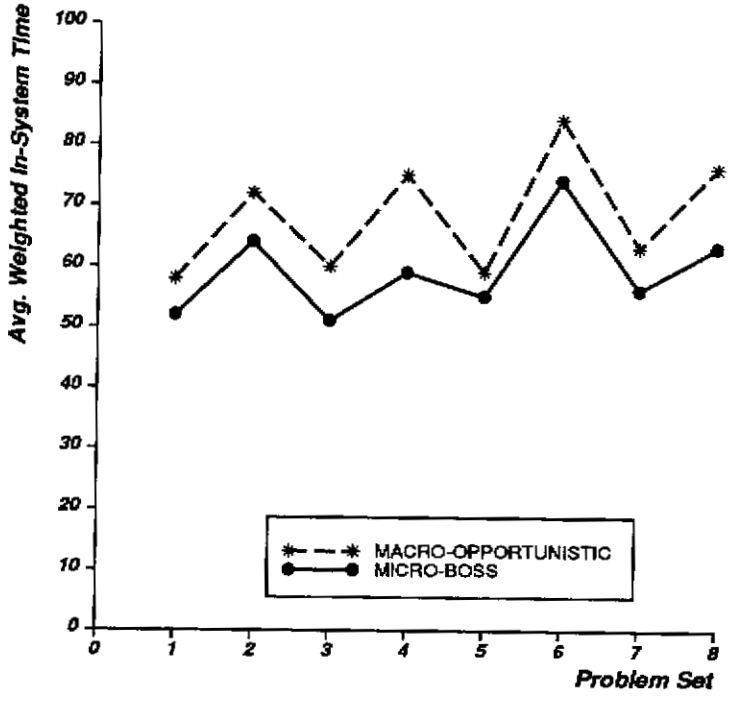

Figure 12: In-system time performance of MICRO-BOSS and the macro-opportunistic scheduler on eight different problem sets.

6 Conclusions

In this paper, a micro-opportunistic approach to factory scheduling was described that closely monitors the evolution of bottlenecks during the construction of the schedule, and continuously redirects search towards the operation that appears to be most critical. This approach differs from earlier opportunistic approaches, as it does not require scheduling large resource subproblems or large job subproblems before revising the current scheduling strategy. This micro-opportunistic approach has been implemented in the context of the MICRO-BOSS factory scheduling system [18]. An experimental study indicates that the flexibility of this approach has enabled MICRO-BOSS to outperform combinations of popular priority dispatch rules and release policies described in the Operations Research literature as well coarser opportunistic scheduling approaches proposed in the Artificial Intelligence literature.

Acknowledgment

The author wishes to thank Mark Fox for his help and support while he was performing the research reported in this paper.

References

[1] J. Adams, E. Balas, and D. Zawack.
The Shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34(3):391-401, 1988.

[2] K.R. Baker.
Introduction to Sequencing and Scheduling.
Wiley, 1974.

[3] S. Dauzere-Peres and J.B. Lasserre.
A Modified Shifting Bottleneck Procedure for Job Shop Scheduling.
Technical Report LAAS 90106, Laboratoire d'Automatique et d'Analyse des Systemes, 7, Av. du Colonel Roche, 31077 Toulouse Cedex, France, 1990.

[4] Mark S. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1983.

[5] R.E. Fox.
OPT: Leapfrogging the Japanese.
Just-in-time Manufacture.
In C.A. Voss,
IFS Ltd, Springer Verlag, 1987.

[6] S. French.
Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.
Wiley, 1982.

[7] M.R. Garey and D.S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[8] Eliyahu M. Goldratt.
Optimized Production Timetable: Beyond MRP: Something Better is finally Here.
October, 1980
Speech to APICS National Conference.

[9] F. Robert Jacobs.
OPT Uncovered: Many Production Planning And Scheduling Concepts Can Be Applied With Or Without The Software.
Industrial Engineering 16(10):32-41, October, 1984.

[10] A.K. Mackworth and E.C. Freuder.
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[11] T.E. Morton, S.R. Lawrence, S. Rajagopolan, S Kekre.
SCHED-STAR: A Price-Based Shop Scheduling Module.
Journal of Manufacturing and Operations Management :131-181, 1988.

[12] Peng Si Ow.
Focused Scheduling in Proportionate Flowshops.
Management Science 31(7):852-869, 1985.

[13] Peng Si Ow and Stephen F. Smith.
Viewing Scheduling as an Opportunistic Problem-Solving Process.
Annals of Operations Research 12:85-108, 1988.

[14] P.S. Ow, S.F. Smith, and A. Thiriez.
Reactive Plan Revision.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 77-82. 1988.

[15] Peng Si Ow and Thomas Morton.
The Single Machine Early/Tardy Problem.
Management Science 35(2):177-191, 1989.

[16] S.S. Panwalkar and Wafik Iskander.
A Survey of Scheduling Rules.
Operations Research 25(1):45-61, January-February, 1977.

[17] A.H.G. Rinnooy Kan.
Machine Scheduling Problems: Classification, complexity, and computations.
PhD thesis, University of Amsterdam, 1976.

[18] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, Carnegie Mellon University, March, 1991.

[19] P. Serafini, W. Ukovich, H. Kirchner, F. Giardina, and F. Tiozzo.
Job-shop scheduling: a case study.
Operations Research Models in FMS.
In F. Archetti, M. Lucertini, and P. Serafini,
Springer, Vienna, 1988.

[20] Edward E. Silver and Rein Peterson.
Decision Systems for Inventory Management and Production Planning.
Wiley, 1985.

[21] S. Smith, M. Fox, and P.S. Ow.
Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.
AI Magazine 7(4):45-61, Fall, 1986.

[22] Ari P.J. Vepsalainen and Thomas E. Morton.
Priority Rules for Job Shops with Weighted Tardiness Costs.
Management Science 33(8):1035-1047, 1987.