Three Stage Generalized Flowshop: Scheduling Civil Engineering Projects

Moshe Dror and Paul A. Mullaseril†

October 25, 1995

Abstract

We model the working of a civil engineering firm concerned with land development as a three stage flexible flowshop with weak chain precedence constraints and where preemption is allowed. The scheduling objective is to minimize the total tardiness for all the projects. Since solving this problem optimally is very hard, we propose a number of heuristic scheduling procedures which are evaluated extensively on real-life data and artificial problem instances.

MIS Department, College of Business, The University of Arizona, Tucson, Arizona 85721.

Journal of Global Optimization 9: 321–344, 1996.

1 Introduction

The problem of project design for land development and public works in a civil engineering consulting firm is examined from the perspective of machine scheduling methodology. The engineering design operations, like jobs in a manufacturing environment, must follow a prescribed sequence. Scheduling engineering jobs becomes very similar to regular job scheduling if engineers are viewed as processing machines, and distinct stages of engineering work are equated with manufacturing work centers. In this paper we describe the operation of an engineering firm specializing in land development and public work design and cast it as a flexible flowshop with three work centers composed of uniform machines. To our knowledge, such a modeling approach has not been examined in the literature in the past and in this paper we show how fitting this approach is for engineering consulting operations.

The engineering practices described below are taken from a local civil engineering firm. Project data for six months taken from that firm is used to examine and validate the analysis presented in this paper. The objective in scheduling such projects is customer satisfaction expressed here as minimizing total tardiness. Though the problem description is confined to the firm observed in this study, it represents a very common setting in land development project planning and the analysis can be easily extended to any number of similar engineering consulting activities. We start by describing in some detail the operation of the firm in this study, followed by a mapping of the firm's operations into a three stage flexible flowshop scheduling terminology.

A civil engineering firm specializing in land development and public works receives requests (customer orders) to design land development projects. The project requests are received dynamically in time and the engineering firm commits itself to a delivery date(s) for each project. A common land development project consists of four main design stages: (i) an initial field study involving a field crew for on site project assessment and measurements, (ii) development of the project design by a design engineer, (iii) transfer of the design specifications into drafting format by a drafting engineer, and (iv) another field study conducted once again by the field crew to lay out draft construction parameters. There are two phases to a contract. A contractual agreement between the customer and the engineering firm specifies the due date for the first phase, after which the customer examines the drafting layout of the project in question and comments on the design. After that, a second due date is set for the project wherein the company completes the field study to layout construction parameters and delivers the final product to the customer. In many cases, both due dates are set at the initial project acceptance but the second due date might be reset when customer examines and comments on the project design. The initial release date for each project is obtained dynamically in time as the projects are received by the firm. Similarly the release date for the final field study for

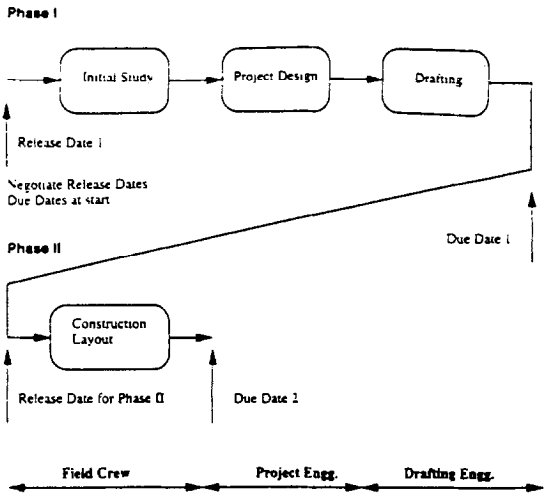

Figure 1: A systems view of the operations

each project is obtained after the drafting format is reviewed by the customer. For most of the projects, the initial field study (stage (i)), and the final field study (stage (iv)) are conducted by a team from the field crew.

We view the process of designing civil engineering projects as a three stage flexible flowshop with uniform machines at each stage. We start by breaking up each project into two separate projects (two jobs) with precedence relationship. Thus, we establish a *chain* precedence relationship of length at most two for the modified set of projects, derived from a given set of projects. Stages (i) - (iii) are viewed as one project with its due date, and stage (iv) is viewed as a separate project with its own due date. The final field study (the original stage (iv) activity) is represented as a new project which requires processing only in stage (i) and has zero processing times in stages (ii) and (iii) of the generalized flowshop. Note that the projects/jobs can be broken up in this way because this Generalized Flowshop is Null-continuous (Hefetz and Adiri, 1982), and since in reality there are two separate due dates for each project – one for the completion at stage (iii) and one for the final project completion (stage(iv)). A systems view of the operations is given in fig. 1.

We now cast the problem in machine scheduling terminology. We will use the term projects and jobs interchangeably in this paper.

2 Problem Formulation

Following the notation in Sriskandarajah and Sethi (1989), a flexible flowshop consists of a set of $k \geq 2$ machine centers $[Z_1, Z_2, \ldots, Z_k]$, with center Z_i having $m_i \geq 1$ parallel machines. There are n projects - (parts) $\{P_j | 1 \leq j \leq n\}$ to be processed in the flowshop. The vector $\pi(P_j) = [p_{j1}, p_{j2}, \ldots, p_{jk}]$ denotes the processing times of the various tasks required by P_j in centers $[Z_1, Z_2, \ldots, Z_k]$, respectively. Task p_{ji} may be processed on any of the m_i machines in machine center Z_i. In the literature (Sriskandarajah and Sethi, 1989, Wittrock, 1988, Blazewicz et al. 1991, Blazewicz, et al. 1994, and Kouvelis and Vairaktarakis, 1994) the common assumption is that the machine centers are composed of identical parallel machines.

In the case of the civil engineering firm in this study, the machine centers are composed of uniform machines. More specifically, in stage (i) there are four crews of field engineers available, each crew can be viewed as a machine and the four crews are considered identical. In stage (ii), one personal computer is available to 5 engineers to assist in the development of a plan for a project. Only one out of the five planning engineers can use the computer at any given time. The computer assisted planning is about 25% faster than project planning without the computer. Thus, stage (ii) is viewed as having 5 machines; four identical machines and one machine 25% faster. Stage (iii) has 3 computers available to five drafting engineers. Hence, stage (iii) consists of three high speed identical machines and two low speed machines (manual drafting). Consistent with the usual scheduling assumptions, no engineer may work on more than one project at a time, and no project can be assigned to more than one engineer at a time. Also, if a task is assigned to a fast machine in any stage, in case of preemption it has to complete its processing on a fast machine. The same holds for the tasks processed in some stage on a slow machine. We cannot switch the processing of a task at any stage from a fast machine to a slow machine and vice versa. We use time units corresponding to half days (reflecting the practice of the civil engineering company in this study) and we allow for preemption which respects machine type. A diagram for the civil engineering project planning operations is given in Figure 2.

Following the notational classification of scheduling problems (see for instance Blazewicz, et al. 1993), the problem described above is represented by $F3|m_1 = P4, m_2 = Q5, m_3 = Q5, r_j, pmtn, weak\ chain \leq 2|\sum_j T_j$, where $F3$ represents the flowshop environment with 3 machine centers, $m_1 = P4$ represents that the first machine center has four identical parallel machines, $m_2 = Q5$ represents that the second machine center has five parallel uniform machines, and $m_3 = Q5$ represents that the third machine center has five parallel uniform machines. We allow for task preemption (the tasks' processing time is measured in half day units). The precedence relationship

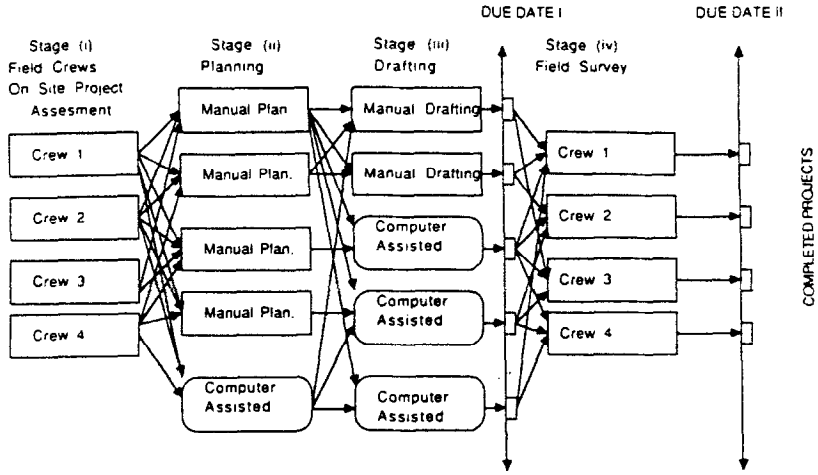

Figure 2: Layout of the facilities

between tasks is that of *weak chain* (see Dell'Olmo et al, 1993) and each chain has at most two tasks. In the third field $\gamma = \sum_j T_j$, which represents that the objective in this problem is that of minimizing the total tardiness. In terms of complexity results, the classical $F2||\sum_j C_j$ problem is NP-hard in the strong sense (Garey, et al. 1976). Thus, $F2||\sum_j T_j$ is NP-hard in the strong sense as well. The flexible flowshop represents a generalization of the traditional one, thus, its complexity classification is no easier than that in the corresponding traditional flowshop cases. In terms of heuristic algorithms, Blazewicz, et al. (1994), present a simple (Johnson like) algorithm which has a worst case behavior of 2 for a two stage flexible flowshop $F2|m_1 \geq 2, m_2 = 1|C_{max}$ and in Kouvelis and Vairaktarakis (1994) an improved worst case bound of $2 - 1/(max\{m_1, m_2\})$ is presented for the more general case. Sriskandarajah and Sethi (1989), present a several similar algorithms with bounded worst case behavior for different variants of two stage flexible flowshop problems where the objective is to minimize makespan.

For the three stage flexible flowshop with tardiness related objective, to our knowledge, no bounded worst case behavior algorithms have been described in the literature. Wittrock (1988), describes a decomposition based heuristic which was tested for sequencing printed circuit board cards on insertion machines grouped into families of one, two, and three machines respectively in a three stage flexible flowshop setting. Wittrock's objective was to minimize the makespan and thereby reduce the auxiliary storage requirements in the system. He reports good results for his algorithm as compared to a lower bound on the schedule length when using real life industrial data for a weekly production. In this paper, we present an algorithmic approach which constructs schedules for a set of projects which the engineering firm received over a course of six months. The projects' data presented here

is real-life data and the tardiness for the projects is computed in half day units which closely approximates the operational units in our company.

3 Description of the Proposed Scheduling Procedures

In this section we introduce three procedures for scheduling tasks in our three stage flexible flowshop, The Forward Heuristic, The Backward Heuristic and The Earliest Due-Date Heuristic. We also discuss a post optimization swap heuristic called the Interchange Improvement procedure. An example problem to illustrate the mechanics of each of these heuristics is also presented.

Currently, the firm's practice is to schedule projects for the entire week each Monday morning. Since preemption of tasks is allowed, a task whose processing was started in the previous week (at any one of the three processing stages) but has not yet been completed is regarded as a new task and where its processing time is appropriately adjusted. The only continuity requirement is that the same machine type (slower or faster) is necessary for the completion of a started task, so that tasks started on a computer will remain on it while manual tasks are done manually later. The projects are scheduled according to an intuitive perception of job priority relative to potential tardiness. In most cases this strategy results in a near-term Earliest Due Date (EDD) heuristic. It should be noted that, because of a problem in accurately estimating project's requirements or the firm's capacity, for some projects a due date specified at the initial project acceptance time may introduce unavoidable tardiness (i.e., the minimum processing time required for a project already results in some tardiness for that project given the due date).

Project data for a six month of firm's operation was made available to us, and hence it was easy to calculate the total tardiness as experienced by the firm during that period. In real life the projects are scheduled on line each Monday morning from a pool of projects available at that time. Since our project information for the testing period was in some sense more complete, we opted to assume that the EDD heuristic would be the appropriate candidate to represent the firm's present performance and serve as a bench mark for comparison instead of just reporting the current operational tardiness. We will first describe the EDD heuristic as applied to this civil engineering project planning and disregard for the time being the *weak chain* precedence relations. This precedence relation is imposed directly in the heuristic procedure.

3.1 Earliest Due Date (EDD) Heuristic

The *earliest due date* (EDD) algorithm (known as Jackson's rule, Jackson, 1955), simply schedules the projects in order of non-decreasing due dates. For this $F3|m_1 = P4, m_2 = Q5, m_3 = Q5, r_j, prmtn|\sum_j T_j$ problem, this EDD heuristic is extended in an attempt to reduce the mean flow time for the projects, always favoring the faster machines in each stage whenever there is an assignment option. We completely schedule the tasks of a project in increasing order of the stages, before scheduling tasks from the next project. As soon as the a phase 1 project is completely scheduled, the corresponding phase 2 project is added to the list of tasks to be scheduled. A description of the modified EDD heuristic for the three stage Generalized Flow Shop is given below.

Step 1 Construct a list L of projects sorted according to non-decreasing due dates d_j.

Step 2 Remove the project at the head of the list.

Step 3 Starting with stage $i := 1$, we schedule each task of the project on the fastest available machine, so that we do not violate constraints imposed by the release date r_j or by the processing times of a task in an earlier stage .

Step 4 As soon as we completely schedule all the stages of a phase 1 project, the phase 2 project in the chain precedence constraint is added to the list and the list re-sorted.

Step 5 We repeat step 2,3 and 4 until the list L is empty.

3.2 Forward Scheduling Heuristic

Emmons (1969) showed that in a one machine environment, the shortest processing time (SPT) is optimal if it yields a sequence where all the tasks are tardy. He also showed that the earliest due date heuristic (EDD) sequence is optimal if it yields a sequence where at most one job is tardy. We note that the engineering firm in this study practices accepting as many projects as possible even if it initially results in tardiness for some or all the projects. Thus, it is important for our heuristics to schedule a family of projects even when none of the projects can be completed on time. Following Emmons single machine SPT-EDD observations, our first heuristic can be characterized as a combination of SPT and EDD approach. We favor the faster machine in each stage whenever there is an assignment option. However, since preemption of tasks is allowed, in the Forward and Backward heuristics we reschedule the available tasks each time a new release date is reached. In other words, we apply the two heuristics to the subset of tasks available between any two consecutive release dates. We break ties using the SPT rule.

Let S be the number of stages (in this case 3) and m_i be the number of available machines in stage i where $i \in (1, ..., S)$. Let R be the list of release dates sorted in non-decreasing order. Identified with each project J_j are: a release date r_j, a due date d_j, a vector of tasks $[J_{j1}, ... J_{ji}, ..., J_{jS}]$, a vector of processing times $[P_{j1}, ..., P_{ji}, ... P_{jS}]$, and a vector of available times (defined later) $[t_{j1}, ... t_{ji}, ..., t_{jS}]$. The available times are determined dynamically while assigning tasks. Associated with each project J_j is a project J'_j reflecting the second phase of the original project. J'_j's release date is determined by the completion date of project J_j, even though its due date d'_j is usually known in advance. Note that J'_j has processing time zero in all stages except the first stage. A set of available tasks in any time interval $[t_i, t_j]$ consists of those tasks of a project whose associated tasks in an earlier stage and phase (if applicable) have been completely assigned. Thus the set of available tasks can be started in the time interval $[t_i, t_j]$ without conflicting with the processing of its predecessors. The set of predecessor tasks for a task J_{ji} includes all tasks that in previous stages of the project j and portions of the task already scheduled in the same stage. The heuristic is as follows:

Step 1 Consider the release date at the head of the list R, say r. Preempt already scheduled tasks in all machines (in all stages as well) at this release date r. Preemption of a task in phase 1 project will result in the removal of all scheduled tasks of the project in subsequent stages and the entire removal of the corresponding phase 2 project (if scheduled).

Step 2 Start with stage $i := 1$. Construct a list L of available tasks J_{ji} in the interval $(r, r + 1)$, where $r + 1$ is the next distinct release date in the list R after release date r. Set the available time for the first task in the three stages of a project J_j in the list as $t_{ji} = r$, the available times of tasks in subsequent stages will be determined by the completion time of its predecessor tasks. This also applies to projects that are the phase 2 project in the weak chain precedence relationship. To this list add the set of unassigned pre-preempted tasks in stage i from step 1. The available times for the first of these tasks in a project are set as r. The available times for tasks in stage i that are not yet completely scheduled are determined from the completion times of its predecessors. Sort the list L according to nondecreasing available times t_{ji}.

Step 3 Schedule each successive task from this list, preemptively so as to minimize its completion time. Apply the *shortest processing time* (SPT) rule for tasks with the same due date, always favoring the faster machines in each stage whenever there is an assignment option. If any task J_{ji} is completely scheduled, update the available time of the task in the next stage of that project according to $t_{ji} + P_{ji} \leq t_{ji+1}$ such that it does not conflict with its predecessors. Update the available times of tasks in stage i whose predecessors have changed.

Project	*Release*	*Phase I*				*Phase II*	
	Date	*I*	*II*	*III*	*Due date*	*I*	*Due Date*
Job 1	0	30	30	20	60	6	70
Job 2	0	20	20	10	60	10	70
Job 3	10	16	20	20	60	10	100
Job 4	20	10	10	30	70	10	90

Table 1: Processing times in the example problem

Step 4 Repeat step 2 and 3 with $i := i + 1$ until the last stage. If at stage 3 a project J_j is completely scheduled, then insert the next phase J_j' of that project into list L.

Step 5 Repeat steps 2,3 and 4 until the list L is empty.

Step 6 Repeat steps 1, 2 ,3 ,4 and 5, until the list R is empty.

If there are n tasks, the time complexity of this algorithm is $O(n^2 \log n)$. We illustrate this procedure in the example below.

Example 1: Consider the set of projects in Table 1 to be scheduled on a flow shop with three stages and two machines in each of the stages. The second machines is twice as fast as the other machine in each stage. The numbers in the columns marked *I, II, III* are the processing times for the tasks in each of those stages. In first iteration, we schedule projects with release time in the interval [0, 10), which are projects 1 and 2. In stage 1, since the available times for both projects are the same, we use the SPT rule to break ties. Project 2 has tasks with the shortest processing time, hence, we schedule it first, on machine 2 (the faster one). Project 1 is also scheduled on this machine. In stage 2 once again project 2 has the earliest available time and is scheduled first on machine 2, thereafter project 1 on machine 2. At the end of stage 3, on completion of phase 1 of projects 1 and 2, we add the phase 2 projects to the list, labeled as tasks 1001 and 1002. Since the processing time of task 1001 is shorter than the processing time of task 1002, it is assigned first to machine 2, which is idle. At the end of this iteration we have the assignment as shown in figure 3a.

In the second iteration we consider tasks in the interval [10, 20), which consists of only task 3. Since the release date of this job is 10 all tasks of machine 2 in stage 1 after this date are preempted. Similarly all tasks in stages 2, 3 and Phase 2 tasks in stage 1, namely tasks 1001 and 1002 are preempted. When assigning tasks in each stage,

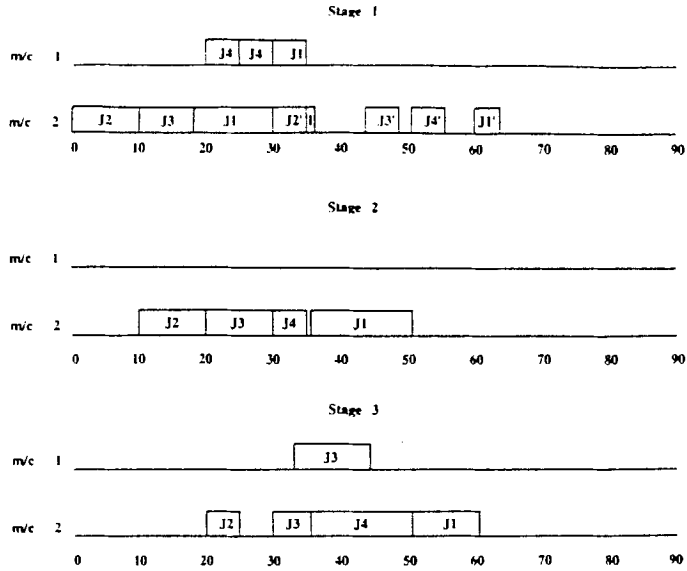

Fig 3. Example showing the job assignment using the forward scheduling heuristic.

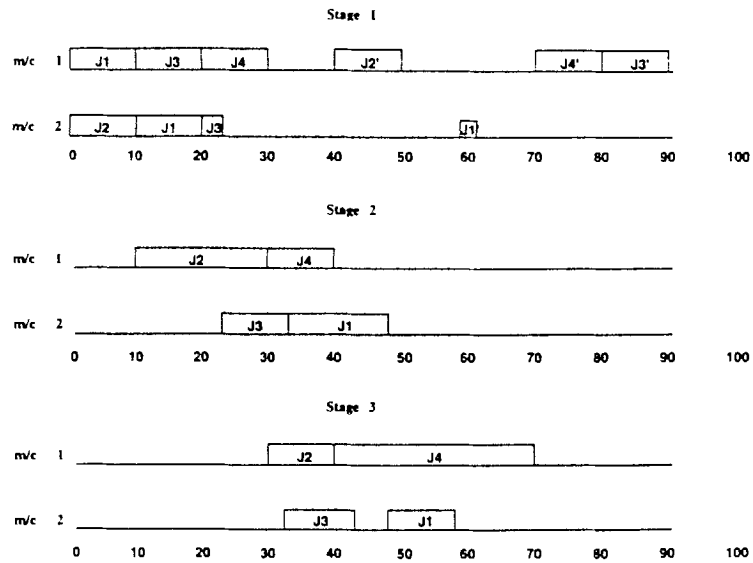

Fig 4. Example showing the assignment job assignments using the backward scheduling heuristic.

Forward Scheduling Heuristic

Field-Crew 1

*** No Assignments are made on this machine ***

Field-Crew 2

Job	Start	Finish	Phase 2	Tardiness
2	0.00	10.00	N	0.00
1	10.00	25.00	N	0.00
1002	25.00	30.00	Y	0.00
1001	50.00	53.00	Y	0.00

Project-Planning 1

*** No Assignments are made on this machine ***

Project-Planning 2

Job	Start	Finish	Tardiness
2	10.00	20.00	0.00
1	25.00	40.00	0.00

Drafting-Engineer 1

*** No Assignments are made on this machine ***

Drafting-Engineer 2

Job	Start	Finish	Tardiness
2	20.00	25.00	0.00
1	40.00	50.00	0.00

Figure 3a: Assignments for after iteration 1

Field-Crew 1

Job	Start	Finish	Phase 2	Tardiness
1002	25.00	30.00	Y	0.00
1002	30.00	33.00	Y	0.00

Field-Crew 2

Job	Start	Finish	Phase 2	Tardiness
2	0.00	10.00	N	0.00
3	10.00	18.00	N	0.00
1	18.00	33.00	N	0.00
1002	33.00	34.00	Y	0.00
1003	40.00	45.00	Y	0.00
1001	58.00	61.00	Y	0.00

Project-Planning 1

*** No Assignments are made on this machine ***

Project-Planning 2

Job	Start	Finish	Tardiness
2	10.00	20.00	0.00
3	20.00	30.00	0.00
1	33.00	48.00	0.00

Drafting-Engineer 1

*** No Assignments are made on this machine ***

Drafting-Engineer 2

Job	Start	Finish	Tardiness
2	20.00	25.00	0.00
3	30.00	40.00	0.00
1	48.00	58.00	0.00

Figure 3b: Assignments for after iteration 2

Backward Scheduling Heuristic

Field-Crew 1

Job	Start	Finish	Phase 2	Tardiness
1	-20.00	10.00	N	0.00
1002	60.00	67.00	Y	0.00
1002	67.00	70.00	Y	0.00

Field-Crew 2

Job	Start	Finish	Phase 2	Tardiness
2	30.00	40.00	N	0.00
1001	67.00	70.00	Y	0.00

Project-Planning 1

Job	Start	Finish	Tardiness
2	0.00	20.00	0.00
1	20.00	50.00	0.00

Project-Planning 2

Job	Start	Finish	Tardiness
2	50.00	50.00	0.00

Drafting-Engineer 1

Job	Start	Finish	Tardiness
2	50.00	60.00	0.00
1	60.00	60.00	0.00

Drafting-Engineer 2

Job	Start	Finish	Tardiness
1	50.00	60.00	0.00

Figure 4a: Assignments after iteration 1

Field-Crew 1

Job	Start	Finish	Phase 2	Tardiness
1	0.00	30.00	N	0.00
1002	50.00	57.00	Y	0.00
1002	57.00	60.00	Y	0.00

Field-Crew 2

Job	Start	Finish	Phase 2	Tardiness
2	0.00	10.00	N	0.00
1001	80.00	83.00	Y	0.00

Project-Planning 1

Job	Start	Finish	Tardiness
2	10.00	30.00	0.00
1	40.00	70.00	0.00

Project-Planning 2

*** No Assignments are made on this machine ***

Drafting-Engineer 1

Job	Start	Finish	Tardiness
2	40.00	50.00	0.00

Drafting-Engineer 2

Job	Start	Finish	Tardiness
1	70.00	80.00	0.00

Figure 4b: Assignments after move left and move right operation.

the preempted tasks are added to the appropriate lists. In case of stage 1 phase 1 tasks, the available time is 10. In case of other stages and for phase 2 the available times are computed dynamically. The assignment at the end of this iteration is shown in figure 3b.

A *gantt chart* showing the final schedule for the example in Table 1, is presented in *figure 3*. Tasks that pertain to phase II of a project J_j, *i.e.* the second project in the chain precedence constraint is denoted as J'_j.

3.3 The Backward Scheduling Heuristic

The *backward scheduling* (BS) heuristic is an alternative method for scheduling the projects in this civil engineering context. The BS procedure for scheduling the projects starts by assigning the project whose due date is the latest and then sequentially assigns the tasks with earlier due dates. Its emphasis is on flow time at each stage by assigning the longer tasks to the slower machines. The time adjustments are initially made by updating the starting times for each project at each stage so that it can start as soon as possible.

For any given pool of projects, schedule the project with the latest due date (break ties according to largest processing time in the last stage) so that it will be completed at or before its due date. After scheduling all the projects, adjust the times by moving the starting times to the left whenever it is possible as long as they are non-negative. In case of a negative starting time, move the project to the right by the least amount so the starting time is at least the release date. Adjust all the projects whose schedule is affected by moving their processing to the right by the same amount. Whenever a choice is possible, assign the longer task on the slower machine when assigning from right to left in time (and stages). In this case note that when we schedule preemptively, it is done so backwards. This means that while scheduling task backwards we may remove portions of already scheduled tasks prior to the due date of the job being scheduled upto the release date of the project.

Once again, let S be the number of stages and m_i be the number of available machines in stage i where $i \in (1, \ldots S)$. Let R be the list of release dates sorted in non decreasing order. We now define a related quantity, *target completion time* t_{ji} of any task i as the latest date that the task must be completed in stage i for the project J_j to be on time. As discussed before, identified with each project J_j are: release date r_j, due date d_j, a vector of tasks $[J_{j1}, \ldots, J_{ji}, \ldots, J_{jS}]$, a vector of processing times $[P_{j1}, \ldots, P_{ji}, \ldots, P_{jS}]$, and target completion times $[t_{j1}, \ldots, t_{ji}, \ldots, t_{jS}]$. Associated with each project J_j is a project J'_j reflecting the second phase of the project, whose release date is the same as that of J_j but whose available time for the first stage is determined by the completion date of project J_j and whose processing times are zero in all stages except the first stage. A set of available tasks

in any time interval $[t_i, t_j]$ consists of those tasks of a project whose associated tasks in an later stages and phase 1 (if applicable) have been completely assigned. The set of succesors of a task is the tasks in subsequent stages of the project and portions of the task already scheduled. The algorithm is as follows:

Step 1 Start with the release date at the head of the list R, say r. Preempt all tasks in all machines in all stages at this release date r. Set the available time of the pre-empted tasks (if there are any) as r. Preemption of a task in phase 1 project will result in the removal of all scheduled tasks of the project in subsequent stages and the removal of the corresponding phase 1 project if a phase 2 task is preempted.

Step 2 Starting with the last stage $i := S$. Construct a list L of available tasks J_{ji} in the interval $(r, r+1)$. To this list add the set of unassigned pre-empted tasks in stage i from step 1. Set the completion time for the last stage of project J_j as $t_{j1} = d_j$ for all tasks in the list L. Sort the list L according to non increasing target completion times t_{ji}.

Step 3 Remove the first available task in stage i from the list L and schedule the task with the latest time t_{ji} preemptively (break ties according to *longest processing time* in the last stage) so that it will be completed at or before its due target completion time t_{ji}, favoring the slowest machine whenever there is an assignment option. If a task does not fit (too long), we need to move tasks to the right on one of the machines in order to schedule this task. Determine the machine where the smallest such movement can be made and move all tasks on that machine to fit the task in question. Determine all other tasks affected by such movement of tasks and move them also, so that the schedule made thus far remains feasible.

Step 4 If any task J_{ji} is completely scheduled, update completion time of the task in the previous stage of that project according to $t_{ji-1} \leq t_{ji} - P_{ji}$ such that it does not conflict with its successor tasks. Update the target completion times of any task in stage i whose set of successors tasks has changed. If at stage 1, introduce into the list L any task that becomes available as a result of completely scheduling the phase 2 project in a weak chain precedence.

Step 5 Repeat step 2,3 and 4 with $i := i - 1$ until $i = 1$.

Step 6 Repeat steps 2, 3, 4 and 5, until the list L is empty.

Step 7 After scheduling all the projects in the interval $(r, r+1)$, adjust the times by moving the starting times to the left whenever it is possible as long as they are non-negative and the schedule remains feasible. In case of a negative starting time, move the project to the right by the least amount so the starting times become

non-negative. Adjust all the projects whose schedule is affected by moving their processing to the right by the same amount.

Step 8 Repeat step 1 to 7 until R is empty.

The time complexity of this algorithm is $O(n^2 \log n)$ as well.

Example 2: We illustrate the BS heuristic using the same example as before. Start with the interval [0, 10), with tasks 1 and 2. Assign first the Phase 2 tasks, in this case phase 2 task 2 (label 1002) on machine 1 starting at time 60 and ending at time 70. Phase 2 task 1 (label 1001) is assigned on machine 2 from 67 to 70. Tasks are then assigned on stage 3, 2, and 1. We note that task 1 on machine 1 stage 1 is assigned from -20 to 10. Hence we need to move the tasks to the right. The assignments at the end of step 3 are shown in figure 4a. Next we move the tasks as far left as possible in step 7. The assignment after this phase is shown in figure 4b. The procedure is repeated for intervals [10, 20) and [20, ...]. We preempt all tasks at time 10 and 20. The final assignments are displayed in the *gantt chart* in *figure 4*.

3.4 Interchange Improvement Procedure

We now present a project interchange heuristic which receives as its input a schedule between two consecutive release dates. The objective of the heuristic is to effect the maximum reduction of the total tardiness by a series of pairwise swaps of the constituent tasks (belonging to the same stage) of pairs of projects. Suppose, we define for all tasks i in any project j, a quantity

$$T_{ji} = \max(0, (d_j - C_{ji}))$$

C_{ji} being the completion time of task i in project j, then

$$T_{j1} \leq T_{j2} \leq ..... \leq T_{ji} \leq ..... \leq T_{jS}$$

where $i \in (1, ..., S)$. So, using T_{ji} as a measure to select candidate pairs will always result in one selecting the tasks in the last stage, whereas the project may be delayed due to a delay in any of the tasks in other stages. Hence we introduce the notion of *delay*. The delay of a task is defined as the maximum of the difference between the target completion date and the actual completion date, and zero, where the target completion date t_{ji} of any task J_{ji} in stage i of project J_j is calculated as

$$t_{ji} = \max((r_j + \sum_{k=1}^{i} P_{jk}), (d_j - \sum_{k=i+1}^{S} P_{jk}))$$

where $i, k \in (1, .., S)$. Thus, delay D_{ji} for a task belonging to project j in stage i is given by:

$$D_{ji} = \max(0, (t_{ji} - C_{ji}))$$

For any stage i, evaluate interchanging a task with the highest delay with all other tasks in the same stage. The pair of tasks that decrease the total tardiness the most is selected for a pairwise interchange. We continue until we can no longer decrease total tardiness by pair-wise interchange of tasks. The heuristic is formally stated below.

Step 1 Starting with stage $i = 1$, construct a list L of all tasks in stage i. Evaluate the delay of each task in the list L. Sort the list according to non-increasing delay.

Step 2 Remove the first task on the list L and evaluate interchanging this task with other tasks in the list by calculating the potential decrease (savings) in the total tardiness. Select the pair that effects the highest positive savings and implement the interchange. When making a pairwise interchange of tasks with unequal processing times, move tasks if necessary to the right to accommodate a larger task and move tasks to the left to fill a void left by a larger task. We use the same routines for moving left and right as done in the backward scheduling heuristic. In the case of moving tasks to the right we may need to move tasks in subsequent stages so that is no conflict in the processing of tasks.

Step 3 Repeat steps 1 and 2 for all other stages.

Step 4 Repeat steps 1 to 3 until no more improvement in total tardiness can be made.

If there are n tasks, the time complexity of the interchange procedure is $O(n^2)$.

4 Computation Results

For the six month project data obtained for the engineering firm, we have the release times, completion times, and the processing requirements for each project. We rerun the two heuristics (FS, BS) each Monday updating the corresponding pool of projects by adding all the new projects released since last Monday (i.e, on line scheduling). The EDD heuristic was used for benchmarking the performance of our heuristics. The results are tabulated in *Table.2*. We refer to the Forward Scheduling, Backward Scheduling and the Earliest Due Date heuristics as *Alg.1*, *Alg.2* and *Alg.3* respectively.

	With Improvement		*Without Improvement*		
	Alg.1	**Alg.2**	**Alg.1**	**Alg.2**	**Alg.3**
Total Tardiness	9.00	2174.36	568.32	2466.21	5850.50
Average Tardiness	0.14	34.51	9.02	39.15	92.87
Maximum Tardiness	6.00	105.67	63.01	136.50	251.50

Table 2: Computation results for six month scheduling horizon

In an effort to study the performance of these algorithms under varied circumstances, we generated problems of size 10 to 60 tasks in increments of 10, based on the data given to us, generating 20 problems in each size. The tasks were generated using a uniform distribution, since it was found to best fit the processing times of typical tasks encountered by the firm. The inter arrival time between tasks arriving to the system was modeled using an exponential distribution. We use *Unifit II*, a software package developed by Averill Law and Associates, Tucson and the methodology prescribed by Law and Kelton (1991) to determine the suitability of the probability distributions. The upper and lower bounds of the distribution were determined from the data given. It was observed that tasks arrived to the system at the frequency of 3 to 4 tasks every ten days. We used a direct proportion of this arrival rate (called *congestion factor*) to determine the arrival rate for the different data sets. This is done, so that the scheduling horizons for each data set is reduced appropriately, so that the tardiness results are comparable. The results of these computations are tabulated in *table 3*, while *table 3a-c* give a comparison of the results for different values of the congestion factor.

We observe that the Forward heuristic (Alg.1) tends to dominate the others by generating a solution where all machines were kept occupied as much as possible (Table 2). However this strategy tended to produce many preemptions which may be detrimental to other objectives of the firm. The backward scheduling heuristic (Alg.2) allocates larger tasks to the slower machines whenever there is an assignment option. This produced fewer preemptions at the cost of increased total tardiness. The number of preemptions produced by both these heuristics for 20 sample problems in each category of size is compared in table 2a.

It is also observed (Table 3) that the pairwise interchange heuristic does not improve the initial tardiness results, by a significant amount unless the congestion to the system is high. In table 3, on actual data from the firm the congestion is high because the firm accepts as many projects as possible (high congestion) even if it results in tardiness for some or all of the projects. If the congestion is low the interchange procedure does not significantly

	Forward Heuristic		*Backward Heuristic*	
	Min.	**Max.**	**Min.**	**Max.**
Size 10	74	136	13	38
Size 20	250	433	51	87
Size 30	350	726	87	155
Size 40	465	1137	154	211
Size 50	522	1161	183	285
Size 60	673	1356	225	329

Table 2a Comparison of the number of preemptions

Table 3: Computation results of 20 experiments at each size of simulated input data

	Average Mean Tardiness Phase 1			*Average Mean Tardiness Phase 2*			*Mean Maximum Tardiness*		
Input Data	**Alg 1**	**Alg 2**	**Alg 3**	**Alg 1**	**Alg 2**	**Alg 3**	**Alg 1**	**Alg 2**	**Alg 3**
without Improvement heuristic									
Size 10	0.67	26.61	8.89	0.37	45.48	6.92	3.92	131.36	34.56
Size 20	4.45	22.20	32.48	3.84	39.93	30.36	19.98	106.97	93.64
Size 30	4.74	14.92	42.76	5.32	31.64	120.65	24.04	120.65	120.65
Size 40	5.41	14.43	57.70	6.76	30.95	52.97	29.89	97.78	143.54
Size 50	2.14	10.68	56.98	2.13	21.40	53.52	18.25	82.73	142.21
Size 60	1.85	9.32	69.17	1.57	22.56	65.20	21.94	100.21	180.29
with Improvement heuristic									
Size 10	0.67	25.17	8.80	0.37	45.27	6.85	3.92	148.14	34.56
Size 20	4.45	21.21	32.16	3.84	36.05	30.04	19.98	92.18	93.04
Size 30	4.74	16.55	42.29	5.32	47.76	39.44	24.04	155.88	119.73
Size 40	5.41	14.20	56.21	6.76	28.89	51.50	29.89	92.26	141.21
Size 50	2.14	11.28	55.95	2.13	22.78	52.55	18.25	105.43	140.96
Size 60	1.84	9.34	67.71	1.55	21.00	63.81	21.59	87.99	178.69

Note:- Alg 1 - Forward Heuristic, Alg 2 - Backward Heuristic and Alg 3 - EDD Heuristic.

The figures in the first two columns (Average mean tardiness) is the average of the mean tardiness from each simulation run.

Sample Size is 20 in each experiment

Table 3a. The performance of the forward heuristic at various values of the congestion factor.

Input Data	*Cong = 0.25*		*Cong = 0.50*		*Cong = 1.00*		*Cong = 1.25*		*Cong = 1.50*	
	Phase 1	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*
Size 10	2.96	1.93	1.48	1.03	0.67	0.37	0.49	0.30	0.44	0.30
Size 20	26.22	25.35	14.92	14.24	4.45	3.84	2.68	2.15	1.76	1.38
Size 30	51.90	57.25	27.67	34.88	6.30	7.11	3.23	3.42	1.19	1.25
Size 40	66.91	84.98	35.65	48.51	5.10	5.59	2.03	2.00	0.88	0.81
Size 50	85.15	111.82	38.53	52.61	2.14	2.13	1.09	1.01	0.46	0.35
Size 60	101.93	140.49	47.02	70.52	2.09	1.98	0.98	0.88	0.43	0.32

Table 3b. The performance of the backward heuristic at various values of the congestion factor.

Input Data	*Cong = 0.25*		*Cong = 0.50*		*Cong = 1.00*		*Cong = 1.25*		*Cong = 1.50*	
	Phase 1	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*
Size 10	50.80	61.27	41.24	66.06	26.61	45.48	20.90	35.56	18.59	40.28
Size 20	68.31	89.83	53.15	86.17	22.20	39.93	15.82	31.85	14.13	27.58
Size 30	102.80	140.15	51.74	74.10	17.47	34.91	15.02	32.99	11.31	20.56
Size 40	113.11	169.49	48.32	78.77	14.17	30.46	11.36	23.99	8.74	17.64
Size 50	107.50	151.25	43.56	87.34	10.68	21.40	8.95	16.84	7.69	12.80
Size 60	126.55	185.42	59.31	117.71	10.38	22.82	9.17	17.19	7.13	11.46

Table 3c. The performance of the EDD heuristic at various values of the congestion factor.

Input Data	*Cong = 0.25*		*Cong = 0.50*		*Cong = 1.00*		*Cong = 1.25*		*Cong = 1.50*	
	Phase 1	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*	*Phase 1*	*Phase 2*
Size 10	23.40	18.98	16.53	13.45	8.89	6.92	8.05	6.24	7.04	5.57
Size 20	73.17	68.23	57.00	53.36	32.48	30.36	28.25	25.99	17.96	16.57
Size 30	121.84	116.02	90.63	86.35	44.37	41.10	33.42	30.67	21.89	19.51
Size 40	171.16	164.63	123.53	118.22	56.03	51.91	40.99	37.49	25.99	23.38
Size 50	213.63	207.89	149.13	144.31	56.98	53.52	38.14	35.36	17.93	16.00
Size 60	258.34	253.97	185.61	178.98	71.53	66.62	46.53	42.66	20.11	17.68

Note:- The figures in each of the two columns (Average mean tardiness) is the average of the mean tardiness from each simulation run. Sample Size is 20 in each experiment

Table 4a: Results of paired t-test comparing the performance of three heuristics (without improvement heuristic)

	Phase 1	*Phase 2*	*max. tard*
	p-value	p-value	p-value
Size 10			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.003	0.000	0.003
Forward vs. Backward	0.000	0.000	0.000
Size 20			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.004	**0.145**	**0.409**
Forward vs. Backward	0.000	0.000	0.000
Size 30			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.123	0.775
Forward vs. Backward	0.000	0.000	0.000
Size 40			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.000
Forward vs. Backward	0.000	0.000	0.000
Size 50			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.000
Forward vs. Backward	0.000	0.000	0.000
Size 60			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.001
Forward vs. Backward	0.000	0.000	0.000

Table 4b: Results of paired t-test comparing the performance of three heuristics (with improvement heuristic)

	Phase 1	*Phase 2*	*max. tard*
	p-value	p-value	p-value
Size 10			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.001	0.001	0.003
Forward vs. Backward	0.000	0.000	0.000
Size 20			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.001	0.071	**0.920**
Forward vs. Backward	0.000	0.000	0.000
Size 30			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	**0.559**	**0.491**
Forward vs. Backward	0.001	0.012	0.023
Size 40			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.000
Forward vs. Backward	0.000	0.000	0.000
Size 50			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.042
Forward vs. Backward	0.000	0.000	0.000
Size 60			
Forward vs. EDD	0.000	0.000	0.000
Backward vs. EDD	0.000	0.000	0.000
Forward vs. Backward	0.000	0.000	0.000

decrease total tardiness. In some cases the pairwise swaps actually produced worse results because they were applied only to the schedule generated between two adjacent release dates and thereafter the schedules were not undone.

5. Summary

In this paper we study a common crew scheduling problem in a civil engineering firm concerned with land design and development. The problem is modeled as a three stage generalized flowshop problem with weak chain precedence constraints and where preemption of the tasks are possible in each of the stages. The modeling aspects are interesting because :- (a) it represents a very common setting in land development project planning and the analysis can be extended to any number of similar engineering consulting activities. (b) we establish a weak chain precedence relationship of length at most two, to model the fact that each contract actually consists of two separate phases, each with its own due date. (c) we extend the frame of generalized flowshop to cover uniform machines in each stage. (d) to our knowledge there is only one other paper on three stage flexible flowshop scheduling (Wittrock, 1988). In that paper, the issue of precedence constraints and preemption are not considered.

We examine alternative scheduling approaches for minimizing tardiness in the context of the generalized flow-shop. In view of the problem complexity we consider only heuristic solution methods. The impact of post-heuristic optimization techniques such as pairwise swaps of tasks are evaluated. From the computation results, one of the heuristics (forward) performed very well, generating in many cases schedules with total tardiness very close to zero.

The proposed heuristic procedures are compared on real-life data from a local civil engineering form. One of the heuristic procedures (the forward heuristic) tends to dominate the others by generating a solution where all machines were kept occupied as much as possible (Tables 2 & 3). However this strategy tended to produce lots of preemptions which may be detrimental to other objectives of the firm. The backward scheduling heuristic allocates larger tasks to the slower machines whenever there is an assignment option. This produced less number of preemptions at the cost of increased total tardiness. Both heuristics performed considerably better than the existing solution strategy (Tables 2&3). In an extensive computational study, we compare these methods with existing practice at the firm and extensions of others that have been proposed in past research and show their effectiveness under a variety of problem scenarios (Table 2a-c). A statistical study (table 4a&b.) using *One-way Anova* with blocking on data size and *paired t-tests* shows that the forward heuristic out performed the backward heuristic, while the Backward heuristic was better than the extended EDD heuristic for large problem sizes. Those comparisons that did not show statistical significance are marked in bold. We also studied the effect of congestion in the system by varying the

rate that tasks arrive to the system. It was seen that the above results hold irrespective of the congestion, i.e. there is no observed interaction between the congestion factor and the problem size in case of the tested heuristics.

In the future we plan to study the worst case behavior of these algorithms. Analytical results were derived by Sriskandarajah and Sethi (1989) for the two stage generalized flowshop problem where the objective is to minimize the makespan. However the tardiness problem is considerably more difficult and analytical results in the area are few. From the computation results of the forward heuristics, it appears that the average performance of his heuristic is good, however this does not guarantee the same results for the worst case performance. Similarly deriving a good lower bound for such problems will aid in solving these problems to optimality by a branch & bound algorithm, as the upper bounds given by the heuristics are fairly tight.

5 References

Blazewicz, J., Eiselt, H., Finke, G., Laporte, G., and Weglarz, J. (1991). "Scheduling jobs and vehicles in flexible manufacturing systems", *The International Journal of FMS*, 4, 5-16.

Blazewicz, J., Ecker, K., Schmidt, G., and Weglarz, J., (1993). *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, Berlin 1993.

Blazewicz, J., Dror, M., Pawlak, G., and Stecke, K.E., (1994). "A note on flexible flowshop scheduling with two-stages", *Foundation of Computing and Decision Science* 19, 159-172.

Dell'Olmo, P., Dror, M., and Kubiak, W., (1993). " 'Strong'-'Weak' Chain Constrained Scheduling." working paper, MIS dept., University of Arizona.

Emmons, H. (1969). "One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17, 701 - 715.

Garey, M.R., Johnson, D.S., and Sethi, R., (1976). "The complexity of flowshop and jobhop scheduling", *Mathematics of Operations Research*, 117-129.

Hefetz, N. and Adiri, I., (1982). "A note on the influence of missing operations on scheduling problems", *Naval Research Logistics Quarterly* 29, 535-539.

Jackson, J.R. (1955). "Scheduling a production line to minimize maximum tardiness", Research Report 43, Management Science Research Project, University of California, Los Angeles.

Kouvelis, P. and Vairaktarakis, G.L., (1994). "A two stage flexible job shop scheduling problem", *Operations Research Letters*, (in press).

Law, Averill M. and Kelton, David W. (!991). "Simulation Modeling & Analysis", cond Edition, *McGraw-Hill, Inc.* New York.

Sriskandarajah, C. and Sethi, S.P., (1989). "Scheduling algorithms for flexible flowshops: Worst and average case performance", *EJOR*, 43, 143-160.

Wittrock, R.J., (1988). "An adaptable scheduling algorithm for flexible flow lines", *Operations Research*, 36, 445-453.