A COMPARISON OF HEURISTICS FOR PREEMPTIVE RESOURCE-CONSTRAINED PROJECT SCHEDULING

Lori K. Richter
Candace Arai Yano
Department of Industrial and
Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 86-39

October 1986

A COMPARISON OF HEURISTICS FOR PREEMPTIVE RESOURCE-CONSTRAINED PROJECT SCHEDULING

Lori K. Richter
Candace Arai Yano
Department of Industrial and
Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

October 1986

A COMPARISON OF HEURISTICS FOR PREEMPTIVE RESOURCE-CONTRAINED PROJECT SCHEDULING

ABSTRACT

We consider a general resource-contrained project scheduling problem where activity preemption is allowed. The objective is to minimize project duration subject to network precedence constraints and resource availability constraints. We develop two-pass procedures, based on existing heuristic project scheduling techniques, for this particular type of problem. We then evaluate these various heuristics on the basis of several performance criteria, including the length of schedule generated and the number of preemptions caused by each heuristic.

A COMPARISON OF HEURISTICS FOR PREEMPTIVE RESOURCE-CONTRAINED PROJECT SCHEDULING

I. INTRODUCTION

Although a substantial amount of research has been done concerning resource-contrained project scheduling problems, relatively little has focused on the situation where activities are preemptible. Allowing jobs to be interrupted and restarted certainly complicates formulation of the problem, and greatly increases the number of scheduling possibilities. This in turn makes it difficult to determine an "optimal," or even a "good" schedule.

However, in many practical situations, it may be possible (or in fact *desirable*) to allow preemption of activities in progress. Many manufacturing companies produce a customized (or *configured*) product to order. In this situation, production scheduling may involve both *preemptive job-shop scheduling* (due to the nature of the jobs) and *project scheduling* (due to the one-shot nature of each order). In a standard project context (e.g. construction or shipbuilding), allowing job preemption might result in better utilization of resources when resource levels vary from period to period.

The enormous increase in combinatorial complexity resulting from job preemption means that standard integer programming techniques are not applicable to realistic problems. Many "rule of thumb" heuristics (perhaps modified) can be used on large-scale projects, but the relative effectiveness of these heuristics has not yet been determined for the preemptive case. This paper is concerned with the latter problem--evaluating heuristic methods in *preemptive* project scheduling.

In Section II, we give a general formulation of the preemptive project scheduling problem. Section III contains a brief review of existing optimal and heuristic solution techniques. We discuss in some detail our experimentation methodology and results in Section IV. Finally, Section V summarizes our results and points to further topics of interest.

II. PROBLEM DESCRIPTION AND FORMULATION

In general, resource-constrained project scheduling problems are characterized by three things: a *set of activities*, a *set of resources*, and one or more *scheduling objectives*. Below is a more detailed outline of our particular type of problem:

Activities (Jobs)

- We allow *one mode of operation* for each activity. (A *mode of operation* is a set of resources needed to complete an activity and an associated duration time.)

- There may be precedence relationships among the various activities.

- We allow preemption of all of the activities. (There may be a time penalty associated with preemption of a particular activity).

Resources

- In this paper, we consider only *renewable* resources--any unused resources in one period cannot be applied to an activity in a later period. Manpower, machines, and electric power are examples of renewable resources.

- There are fixed limits on the amounts of resources available each period.

Scheduling Objective

- Minimize the project duration given resource limits.

The following formulation is a generalization of those used by Davis and Heidorn [2], Stinson, et al. [12] and Patterson and Roth [10]:

Let

$x_{jt} = \begin{cases} 1 \text{ if job j is active at time t} \\ 0 \text{ otherwise} \end{cases}$

p_j = number of times j is preempted in schedule (x_{jt}),

ρ_j maximum number of times job j may be preempted ($\rho_j \leq d_j$),

r_{jk} = amount of resource k needed by job j each period j is active,

R_{kt} = amount of resource k available at time t,

d_j = duration of job j (when not preempted),

τ_j = time penalty (increase in job j duration cuased by each preemption of job j),

P(j) = set of predecessors of job j,,

N = number of jobs in project (also the number of the unique last job in project),

K = number of resources,

HP = known (heuristic) completion time for project.

Then the problem is to

$$minimize\ [\ max\ (t x_{Nt}: t = 1\ to\ HP\}\ ]$$

subject to

(2.1) $$(d_t + \pi_i p_i) x_{jt} \leq \sum_{t'=1}^{t-1} x_{it},$$ j = 1 to N, i ∈P (j), t = 1 to HP

(2.2) $$\sum_{t=1}^{HP} x_{jt} = d_j + \tau_j p_j,$$ j = 1 to N

(2.3) $$\sum_{j=1}^{N} r_{jk} x_{jt} \leq R_{kt},$$ k = 1 to K, t = 1 to HP

(2.4) $$p_j \leq \rho_j,$$ j = 1 to N

$$x_{jt} = 0 \text{ or } 1,$$ ∀ j, t

$$p_j \text{ integer},$$ ∀ j.

Constraints (2.1) are the precedence constraints. Equations (2.2) give the actual durations of the jobs in a given schedule, while the inequalities (2.4) ensure that no job is preempted more then ρ_j times. Resource limitations at each period t are satisfied by inequalities (2.3). If we set $\tau_j = 0$, and/or eliminate ρ_j, then this formulation becomes somewhat simpler. However, the large number of 0-1 variables x_{jt} makes this a difficult problem to solve optimally.

III. LITERATURE REVIEW

Most of the optimal solution procedures available were developed to solve the problem of minimizing project duration subject to strict resource limits and *non-preemption* of jobs. We discuss two optimal algorithms below which either were designed for or can be adapted to a preemptive scheduling problem.

Davis and Heidorn [2] adapt a procedure developed by Gutjahr and Nemhauser [5] for the assembly line balancing problem. This solution approach is (theoretically) readily applicable to problems which allow preemption of jobs. In addition, it is not necessary to assume constant per period resource usage for all jobs in the project. This is the only optimal technique developed for the non-preemptive case that can easily handle relaxation of these assumptions.

The major disadvantage of the Davis and Heidorn optimal procedure is that both computer core storage and CPU time increase rapidly as the problem size grows. The method is highly sensitive to the degree of precedence interrelationships in the original project network, and to a lesser extent to the length of the job durations.

Slowinski [11] considers a class of project scheduling problems which allows *arbitrary preemption* of activities with no time or cost penalty. He presents two solution techniques, a *one-stage approach* and a *two-stage approach*. Both of these procedures involve solving a large scale linear program, and both can be generalized to handle problems with renewable, non-renewable, and doubly contrained resources, and several modes of operation for the activities.

Although Slowinski derives theoretical results concerning the number of operations required for both the one- and two-stage approaches, no actual

computational results are reported. Therefore, it is not known how well these methods work on real problems. In addition, since his procedures are based on solving *linear* programs (i.e., processing times are continuous) with no penalties for job preemption, an optimal solution may require a large number of preemptions for one or more jobs.

The size and complexity of resource-contrained project scheduling problems (in both the preemptive and non-preemptive situations) has precluded to date the use of optimal procedures on all but the smallest problems ($\leq$ 50 jobs). Thus we must use heuristic techniques to handle realistic problems.

SIMPLE HEURISTICS

Many "rule of thumb" heuristics have been developed for the non-preemptive project scheduling situation. These simple heuristics are, for the most part, priority dispatch rules (i.e., when applied to a set of jobs, each heuristic specifies the *order* in which the jobs should be scheduled). The one exception is the *Greatest Resource Utilization* (GRU) heuristic, which will be discussed in more detail below. These heuristics can be roughly classified into three types: *CPM-Based*, *Resource-Based*, and *Others*. Although these heuristics have generally been applied to non-preemptive scheduling problems, they can be used for projects with preemptible activities if the cost of preemption is negligible. [Note: In the following discussion, "ESj", "LSj", "EFj", "LFj" correspond to Critical-Path Early Start, Late Start, Early Finish, and Late Finish times, respectively.]

CPM-BASED HEURISTICS

Minimum Job Slack (MINSLK) Heuristic:

To resolve resource conflicts using MINSLK, a slack measure is computed for each job involved:

$$SLACK_j = LS_j - ES_j - delay,$$

where *delay* represents the number of periods job j is currently delayed from its Early Start Time.

Resource Scheduling Method (RSM) (Brand, Meyer, Schaffer):

Let T represent the set of jobs that can be scheduled at the current time. For all i,j in T, calculate

d_{ij} = increase in project duration resulitng when activity j follows activity i

= max (0, EF_i - LF_j).

Then give priority to the job i with minimum d_{ij}.

Minimum Late Finish Time (LFT) Heuristic:

In the event of resource conflicts, the LFT heuristic schedules the jobs with smaller Critical Path *Late Finish Time* first.

Minimum Least Total Float (LTF) Heuristic:

This heuristic is the same as MINSLK except that no account is taken of possible delays of jobs past the Critical Path Early Start Times.

RESOURCE-BASED HEURISTICS

Greatest Resource Demanded (GRD) Heuristic:

This heuristic assigns priority to activities in conflict on the basis of some measure of total resource needed by the activities. The amount of resources needed by activity j is computed as:

$$\text{PRIORITY}_j = d_j \sum_{i=1}^{K} r_{ij},$$

and jobs with higher priority are scheduled first. As one can see, this gives a rather crude measure of resource usage. The units of different resource types may not be

compatible. Furthermore, some resources may be more constrained than others; this heuristic does not take this situation into consideration.

Greatest Resource Utilization (GRU) Heuristic:

Instead of assigning priorities to the activities in conflict, the GRU heuristic solves an integer program for each period *t* to find the *best* subset of these activities to schedule ("*best*" in the sense that this subset maximizes the utilization of resources in period *t*). This rule explicitly requires that the units of resource r_{jk} be compatible. Note that this rule maximizes resource usage *locally* (i.e., only for the period in question). It does not take into consideration the durations of the various activities.

OTHER HEURISTICS

Shortest Imminent Operation (SIO) Heuristic:

The SIO heuristic assigns higher priority to activities with shorter durations. Thus this rule is merely the application of the SPT (Shortest Processing Time) job shop scheduling rule to the *project* scheduling situation.

Random (RAN) Heuristic:

This rule schedules conflicting jobs randomly. Although not widely used as a scheduling technique, RAN does serve several purpoes:

- it is used as a basis of comparison for other heuristics
- it serves as a tie-breaker for other heuristics

Reschedule (RES) Heuristic:

The RES heuristic is applied in the following manner:

1) Obtain a feasible schedule using another heuristic.
2) Find the resource with the *least total amount of idle time*.
3) Reschedule, giving priority to activities using greater amounts of this resource.

RES thus attempts to determine the "bottleneck" resource and schedule around it.

There have been several studies to date which attempt to compare the various simple heuristics on a particular type of scheduling problem [1][6][7]. These studies indicate that *some* heuristic generally performed better than the others on the problem in question, but the superior heuristic is different for different problems. Less has been done to compare different heuristics for a range of problems, or for various performance criteria (see [4],[8]).

Patterson [9] performs a statistical analysis which tests several simple heuristics using one "real-life" multi-project scheduling problem and various performance criteria. His results indicate that CPM-based heuristics (in particular LTF) and SIO outperform resource-based heuristics when the objective is to minimize total project delay. However, when the goal is to minimize the Total Resource Idle Time, one should choose resource-based heuristics over other types.

Davis and Patterson [3] compare different heuristics with the optimal schedule for a set of eighty-three test problems. These problems are small enough in size (twenty to twenty-seven activities, one to three resource types) to be solved optimally by the Davis and Heidorn procedure. The optimal project duration serves as the basis of comparison for the various heuristics. As one would expect, the authors conclude that CPM-based heuristics yield better schedules (in terms of *percent increase over optimum project duration*) than resource-based or other heuristics.

Unfortunately, we know of no studies to date which compare these heuristics in a preemptive scheduling environment. One might expect that *all* of the heuristics will perform better in this situation (as long as the actual cost and/or time penalties for preemption are small enough). In the next section, we discuss results of our own experimentation with several simple heuristics in a preemptive scheduling environment.

IV. ADAPTATION OF HEURISTICS TO PREEMPTIVE PROJECT SCHEDULING

Implementation of Heuristics:

We have adapted four heuristics to handle problems involving job preemption: Minimum Slack (MS), Resource Scheduling Method (RSM), Shortest Imminent

Operation (SIO), and Greatest Resource Demanded (GRD). Each of these is implemented in a two-pass procedure. The first (forward) pass schedules jobs according to the respective heuristic, while the second (backward) pass uses *local* and/or *global right-shifting* (see Wiest [13]) to reschedule the project. Working backwards from the end of the project, *right-shifting* moves portions of jobs to later times while maintaining resource feasibility. The objective is to compress the schedule by freeing up whole periods' worth of resources. Time penalties for preemption can be handled by both passes of the procedure. The goal of our experimentation is to examine (and hopefully answer) several questions:

- Does any *one* heuristic perform consistently better than the others on a given set of problems?
- What are the effects of the *second* pass on the schedules generated--is it worthwhile including it in a scheduling procedure?
- How do resource tightness, the degree of precedence interrelationships, and the existence of preemption penalties affect the performance of the various heuristics?

In order to test the heuristics as fairly as possible, we designed a simple program to generate "random" projects. Given the number of jobs in a project, the *maximum* allowable job duration, the average number of predecessors per job, the amount of each resource available per period, and the "tightness" of each resource, we can randomly generate any number of projects with these characteristics. The tightness of each resource is reflected in a resource utilization factor, w_k. If resource k utilization factor is given by w_k, then the amount of resource k needed by each job (r_{jk}) is uniformly distributed on $[0, w_k R_{kt}]$.

Each experimental project consists of 50 jobs, 2 resources. The maximum duration of any job is 10 time periods, and there are 15 units of each resource available each period. All resources are *renewable*, i.e. unused resources from one period cannot be stored for later use. In this paper, we only consider the case where there is *no* time penalty for job preemption. We will discuss the amount of preemption which occurs in this situation. The variable parameters and their values are given below:

average number of predecessors per job:

2 (low degree of precedence relationship in the network)
6 (high degree of precedence relationship)

resource utilization factor, w_k,

0.33 (resources relatively unconstrained)
0.60 (resources fairly constrained)
0.93 (resources tightly constrained)

Fifteen projects were generated for each combination of the variable parameters. All computer programs were written in Turbo Pascal (version 3.0) for the IBM-PC/XT.

Experimental Results:

Our goal was to minimize the project duration; therefore, we evaluated the four heuristics on the basis of these criteria:

1. Percentage deviation of each first-pass heuristic schedule length from the *best first-pass heuristic* schedule for each project,
2. Percentage deviation of each second-pass heuristic schedule length from the *best second-pass heuristic* schedule for each project,
3. Percentage decrease in the length of each heuristic schedule caused by the second (right-shifting) pass.

Results from One-Pass Procedure:

The Minimum Slack (MS) heuristic consistently outperforms the other three heuristics in a one-pass procedure (see Table 1). When the degree of resource tightness becomes very large, the Resource Scheduling Method (RSM) approaches the effectiveness of MS (see Table 2). The same behavior is detected as the degree of precedence relationship in the network increases (see Table 3). Note that both the Shortest Imminent Operation (SIO) and Greatest Resource Demanded (GRD) heuristics perform significantly worse in all cases than the other two methods.

MS	RSM	SIO	GRD
0.57	2.74	14.18	11.37

Table 1. Mean % Deviation from Best Schedule over All Projects (1-pass).

resource tightness	MS	RSM	SIO	GRD
0.33	0.00	2.19	13.40	9.66
0.60	0.43	4.24	17.56	13.59
0.93	1.27	1.81	11.58	10.76

Table 2. Mean % Deviation from Best Schedule vs. resource tightness (1-pass).

ave. # of pred.	MS	RSM	SIO	GRD
2	0.32	3.54	16.02	12.82
6	0.81	1.95	12.34	9.72

Table 3. Mean % Deviation from Best Schedule vs. Precedence Relationship (1-pass).

Results from Two-Pass Procedure:

As we see from Tables 4-6, the general behavior of the heuristics in the two-pass procedure is the same as before. The MS rule continues to yield the best schedules on average in most situations. We note, however, that the SIO and GRD rules (although still producing the worst schedules on average) perform considerably better with the extra right-shifting than they do alone.

MS	RSM	SIO	GRD
0.73	2.31	6.25	6.15

Table 4. Mean % Deviation from Best Schedule over All Projects (2-pass).

resource tightness	MS	RSM	SIO	GRD
0.33	0.00	2.19	6.51	4.99
0.60	0.62	3.72	7.12	6.56
0.93	1.57	1.04	5.11	6.89

Table 5. Mean % Deviation from Best Schedule vs. Resource Tightness (2-pass).

Ave. # of pred.	MS	RSM	SIO	GRD
2	0.48	3.04	7.24	7.40
6	0.98	1.59	5.26	4.90

Table 6. Mean % Deviation from Best Schedule vs. Precedence Relationship (2-pass).

Thus the right-shifting of schedules, while having little effect on *good* first-pass schedules, dramatically improves schedules which were *poor* to begin with. This is verified in Table 7, which shows the mean percent decrease in schedules caused by the right-shifting algorithm.

		MS	RSM	SIO	GRD
resource tightness	0.33	0.00	0.00	5.92	4.08
	0.60	0.21	0.88	9.14	6.19
	0.93	0.65	1.69	6.56	4.31
ave. # of predecessors	2	0.22	0.84	7.84	5.03
	6	0.35	0.87	6.57	4.69
over all projects		0.29	0.86	7.20	4.86

Table 7. Mean % Decrease in Schedules caused by right-shifting.

Unfortunately, this improvement in schedule length comes at the expense of a significant increase in the number of jobs preempted in the schedule. We examined a subset of 30 of the test projects (5 from each parameter combination). While the

one-pass Minimum Slack heuristic schedules contained an average of 6 preemptions per schedule, the *right-shifted* (two-pass) Minimum Slack schedules averaged 19 preemptions per schedule. Similar results were true for all of the other heuristics except the Greatest Resource Demanded (GRD) routine (see Tables 8, 9). Since the GRD heuristic generated schedules with an excessive number of preemptions, it is not surprising that right-shifting reduces the number of preemptions in a schedule.

MS	RSM	SIO	GRD
6	8	4	86

Table 8. Average Number of Preemptions in One-Pass Heuristic Schedules.
(30 test projects)

MS	RSM	SIO	GRD
19	16	18	29

Table 9. Average Number of Preemptions in Two-Pass Heuristic Schedules.
(30 test projects)

V. CONCLUSIONS

On average, the Minimum Slack project scheduling heuristic performed better (in terms of schedule duration) than the other heuristics on a variety of problems. Our results are similar to those of Davis and Patterson for the non-preemptive case [3]. The Resource Scheduling Method of Brand, Meyer, and Schaffer [1] performs almost as well on average, and in some instances better, than the Minimum Slack rule. The other heuristics (Shortest Immiment Operation, Greatest Resource Demanded) produced worse schedules on average, but occasionally outperformed the former rules.

The use of a right-shifting second pass as part of a heuristic procedure is generally effective in shortening the schedule length. In particular, this kind of procedure is most beneficial when the initial schedule is poor. Thus right-shifting serves to bring all heuristic solutions closer together. In general no heuristic is *guaranteed* to work well on a given problem; therefore the use of a right-shifting

algorithm can only increase the likelihood of obtaining a reasonably good schedule, whatever the initial heuristic.

Our next step is to include the possibility of time penalties for preemption of jobs. This would correspond to a set-up time incurred each time a job is preempted. We expect that imposing time penalties for preemption will not significantly alter the relative performance of the various heuristics, but will result in longer schedule length for all heuristics. Hopefully, penalties will reduce the number of preemptions which occur in right-shifted schedules.

More serious issues are the inability of project scheduling routines to obtain optimal solutions for realistic problems and the lack of optimization-based heuristics. Thus one has no way of determining *how good* a particular heuristic solution is. It is not very satisfying to compare various "rule-of-thumb" solutions with each other. We are currently working on the development of a procedure to obtain good lower bounds for the resource-constrained project scheudling problem. Ideally, such a procedure would be the foundation for a *new* heuristic scheduling routine, as well as a basis for evaluating existing heuristics.

REFERENCES

1. Brand, J.D., W.L. Meyer, and R. Schaffer, "The Resource Scheduling Problem in Construction," Civil Engineering Studies Report No. 5, University of Illinois (1964).

2. Davis, E.W. and G. E. Heidorn, "Optimal Project Scheduling Under Multiple Resource Constraints," Management Science, Vol. 17, No. 12 (1971), B803-816.

3. David, E.W. and J. H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," Management Science, Vol. 21, No. 8 (1975), 944-955.

4. Fendley, L. G., "Toward the Development of a Complete Multiproject Scheduling System," Journal of Industrial Engineering, Vol. 19, No. 10 (1968).

5. Gutjahr, A.L. and G.L. Nemhauser, "An Algorithm for the Line-Balancing Problem," Management Science, Vol. 11, No. 2 (1964), 308-315.

6. Knight, R.M., Resource Allocation and Multi-Project Scheduling in a Research and Development Environment," Unpublished M.S. Thesis, M.I.T. (1966).

7. Mize, J.H., A Heuristic Scheduling Model for Multi-Project Organizations, Unpublished Ph.D. Thesis, Purdue University (1964).

8. Pascoe, T.L., An Experimental Comparison of Heuristic Methods for Allocating Resources, Unpublished Ph.D. Thesis, Cambridge University (1965).

9. Patterson, J.H., "Alternate Methods of Project Scheduling with Limited Resources," Naval Research Logistics Quarterly, Vol. 20, No. 4, (1973), 767-784.

10. Patterson, J.H., and G. Roth,"Scheduling a Project Under Multiple Resource Constraints: A Zero-One Programming Approach," AIIE Transaction, Vol. 8, No. 4, (1976), 447-456.

11. Slowinski, R. "Two Approaches to Problems of Resource Allocation Among Project Activities--A Comparative Study," J. Opl. Res. Soc., Vol. 31, (1980), 711-723

12. Stinson, J.P., E.W. Davis, and B. Khumawala, "Multiple Resource-Constrained Scheduling Using Branch and Bound," AIIE Transactions, Vol. 10, No. 3, (1978), 252-259.

13. Wiest, J.D., "Some Properties of Schedules for Large Projects with Limited Resources," Operations Research, Vol. 12, No. 3 (1964), 395-418.