Resource-Constrained Project Scheduling: Past Work and New Directions[1]

Bibo Yang • Joseph Geunes
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL
William J. O'Brien
Department of Civil and Coastal Engineering, University of Florida, Gainesville, FL

April 2001

Abstract

This report summarizes past work in resource-constrained project scheduling problems (RCPSP) and also presents a new RCPSP with a specialized minimum cost objective function. This new RCPSP model focuses on single-resource problems with resource consumption and late delivery costs. This model applies, for example, to a general contractor or sub-contractor in the construction industry facing project deadlines with limited resources and penalties for late completion. We develop a new bin-packing based algorithm to provide good solutions for this problem and describe some computational experience with this algorithm. This paper is separated into two distinct parts. Part 1 (Sections 1 through 5) summarizes the vast literature on RCPSPs and categorizes this literature. Part 2 (Section 6) presents our new RCPSP variant and our heuristic algorithm.

1. Introduction and Classification of RCPSP Problems

Resource constrained project scheduling problems (RCPSPs) involve assigning jobs or tasks to a resource or set of resources with limited capacity in order to meet some predefined objective. As we will see, many different objectives are possible and these depend on the goals of the decision maker, but the most common of these is to find the minimum makespan, i.e., the minimum time to complete the

[1] Research Report 2001-6, Department of Industrial and Systems Engineering, University of Florida. This work was supported by NSF Grant #CMS-0122193.

entire project. We will discuss several varieties of the RCPSP, beginning with so-called single-mode scheduling problems and moving on to multi-mode problems. Single-mode problems imply that each project (activity) has a single execution mode: both the activity duration and its requirements for a set of resources are assumed fixed. In multi-mode problems, each activity has to be processed in one of several modes. Each mode implies a different option in terms of (possibly) cost and activity duration for performing the activity under consideration. We broadly separate the RCPSP into the following 6 different classes:

1. Basic Single-Mode RCPSP
2. Basic Multi-Mode RCPSP
3. RCPSP problems with Non-regular objective functions
4. Stochastic RCPSP
5. Bin-packing-related RCPSP problems[2]
6. Multi-resource-constrained project scheduling problems (MRCPSP)

We next consider the essential elements of each of these classes of project scheduling problems.

1.1 Basic Single-Mode RCPSP

As we have noted, in the single-mode RCPSP, each project (activity) has a single execution mode: both the activity duration and its requirements for a set of resources are assumed to be fixed, and only one execution mode is available for any activity. Single mode RCPSPs generally contain 4 sub-classes [16], which we characterize next. Throughout our discussion we will assume that a project is represented as a network of activities. The network can be represented as a graph, $G(N, A)$, where the nodes in the graph correspond to activities, and the arcs in the graph specify precedence relationships: if

[2] While the bin-packing-related RCPSP is not a formally recognized class of RCPSPs, we use this classification to characterize RCPSPs that can be thought of as general versions of the classical bin-packing problem. We later present a bin-packing-based algorithm for the new RCPSP we define in Section 6. See [12] for more on the relationship between resource-constrained

arc (*i*, *j*) appears in the graph, then activity *i* must be completed prior to performing activity *j*. In stating precedence conditions, we can consider two types: cases in which activity *j* can start at any time following completion of activity *i*, or cases in which activity *j* must start within some time window following the completion of activity *i* (we call these latter restrictions General Precedence Relationships, or GPR). We will take an approach of characterizing resources in terms of their availability (capacity) to perform tasks in each of a set of time periods. Resource availability for a given resource may be the same for all periods, or it may vary from period to period. Additionally, if a task is assigned to a resource in multiple periods, the task might require the same amount of processing time in all periods, or we might allow the resource consumption to vary in different periods. Finally, we can consider cases that require us to complete a task once it is started on a machine versus cases in which we can stop a job in the middle of processing and return to the job later, which is termed preemption. As we have noted, the typical approach in project scheduling minimizes the total project makespan, or the total completion time of all tasks from start to finish. However, a broad array of other objectives is possible. We define a *regular* objective as any objective with the following property: the objective function value for a given schedule is not made any worse if the completion time of a single activity is reduced without changing the activity's mode or changing the completion time of any other task [23]. Table 1 summarizes our classification of different single-mode RCPS problems.

Table 1: Single-Mode RCPS Problem Classification

	Basic SM-RCPSP	**SM-RCPSP with Preemption**	**General Precedence SM-RCPSP**	**General SM-RCPSP**
Objective	Min Makespan	Min Makespan	Min Makespan	Regular
Precedence Relations	P	P	GPR	GPR
Resource Availability	Constant	Constant	Constant or Time-Varying	Constant or Time-Varying

scheduling problems and the bin-packing problem.

Per Period				
Resource Requirements Per Period	Constant	Constant	Constant or Time-Varying	Constant or Time-Varying
Preemption	No	Yes	No	No

1.2 Multi-Mode RCPSP

In a multi-mode RCPSP (MM-RCPSP), given the estimated work content for an activity, a set of allowable execution modes can be specified for the activity's execution. Each mode is characterized by a processing time and amount of a particular resource type for completing the activity. For example, one worker might finish a job in 10 hours (mode 1), whereas 2 workers might finish the same activity in 5 hours (mode 2). The product of the duration of the activity and the amount of the resource type needed is called the activities *work content*. In our previous example, in modes 1 and 2 the activity had a work content of 10 worker-hours. Resources available for completing tasks can be classified as either renewable or non-renewable. Non-renewable resources are depleted after a certain amount of consumption (or number of periods), while renewable resources typically have the same amount of availability in every period for an unlimited number of periods. As in the single-mode RCPSP, we classify different types of MM-RCPSPs in Table 2. Additional variants of the MM-RCPSP exist, such as the MM-RCPSPs with setup times; see [19]

Table 2: Multi-Mode RCPS Problem Classification

	Nonrenewable Resource MM-RCPSP	**Renewable Resource MM-RCPSP**	**General MM- RCPSP**
Objective	Min Makespan	Min Makespan	Regular
Resource Type	Nonrenewable	Renewable	Nonrenewable or Renewable
Precedence Relation	P	P	GPR
Resource Availability Per Period	Constant Within a Mode	Constant Within a Mode	Constant Within a Mode
Resource Requirements Per Period	Constant Within a Mode	Constant Within a Mode	Constant Within a Mode

Preemption	No	No	No
Trade Offs	Time/Resource	Time/Resource	Time/Resource, Time/Cost, Resource/Resource

1.3 RCPSP problems with non-regular objective functions

We next provide some examples of non-regular objective functions. Recall that a regular objective function is one in which the objective function is never made worse by reducing the completion time of a job without increasing the completion time of any other job. A non-regular objective function violates this property. Specific problem types include the resource-leveling problem (RL-RCPSP), the maximum net present value (NPV-RCPSP) problem, and the discounted cash flows problem (DC-RCPSP). Specific non-regular objective functions include:

1) Maximize NPV (unconstrained resource problem)
2) Maximize discounted cash flows (resource-constrained problem)
3) Minimize total (weighted) resource consumption
4) Maximize smoothness of resource usage (resource leveling)

Because of the time dependent nature of these objective function costs, the objective function value can actually increase by reducing the completion time of an activity (all else being equal), and so these are non-regular objective functions.

1.4 Stochastic RCPSP

In a stochastic RCPSP, the processing time of any activity is a random variable, which follows some probability distribution [16]. This problem class, while often more realistic, leads to much greater complexity in analysis. Instead of minimizing makespan, we consider objectives such as minimizing the *expected* makespan. Since many interdependent activities are often represented in a project graph, and activity completion times are highly interdependent, the probability distribution of the total makespan is often extremely difficult or impossible to characterize, often leading to activity

independence assumptions for tractable analysis. Such independence assumptions, however, can provide extremely misleading results in practice. We will not explore stochastic versions of the RCPSP problem in detail since we will focus mainly on activities with predictable resource consumption times.

1.5 Bin-packing-related RCPSP problems

A bin-packing-related RCPSP refers to a special kind of resource-constrained project scheduling problem that can be seen as analogous to a bin-packing problem. A bin-packing problem instance is specified by a standard bin size and a set of items, each of some (possibly) unique size. The objective is to pack every item in the set into a bin while minimizing the total number of bins used. Many resource-constrained scheduling problems can be viewed as generalizations (or even special cases) of the basic bin-packing problem. The analogy between the bin packing and RCPSP problems can be explained as follows. The resource capacity represents the bin size, while a task's resource consumption requirement represent an item size. In the RCPSP context, we can view each time period as a bin into which we can pack different tasks (of course in many of these problems, we can pack an item into consecutive bins, or equivalently schedule jobs across consecutive days). If we suppose, however, that each task takes less than one period of resource consumption and that every task must be fully completed within a single day, then minimizing makespan of this RCPSP is equivalent to minimizing the number of bins used in an equivalent bin-packing problem. We can adapt extremely successful heuristic algorithms used for solving bin-packing problems to the RCPSP with relatively minor modifications. This is our initial approach to solving a RCPSP with a unique objective function that we describe later. Table 3 characterizes different kinds of bin-packing related RCPSPs.

Table 3: Bin-Packing-Related RCPS Problem Classification

	No Precedence BP-RCPSP (Multidimensional BP)	General BP-RCPSP
Objective	Minimize Latest Scheduled Time	Minimize Latest Scheduled Time
Resource Type	Renewable	Renewable
Precedence Relation	No	GPR
Resource Availability Per Period	With Upper Bounds	With Upper Bounds
Resource Requirements	Less Than Resource Upper Bound	Less Than Resource Upper Bound
Preemption	No	No
Trade Offs	No	No

1.5.1 Multi-resource-constrained project scheduling problems (MRCPSP)

In each of the above problem classes, projects/jobs may choose from several resources, but only one operation is required for each job. By contrast, in multi-resource-constrained project scheduling problems, a job may require a set of operations, or a set of successive resources. For a given operation, several resources may be in parallel, which means the job can select any one of these resources for processing. A job might also need to complete processing on one resource before it begins processing on another resource, where successive resources are needed in series. These problems are often called machine-scheduling problems, since in manufacturing we often see machines and workstations in parallel and in series. In the manufacturing context, a resource might be a machine that can only process one job at any time. Machine and job scheduling characteristics can be broken down to several categories:

1) Identical machines in parallel;
2) Machines in parallel with different speeds: the speed of machine i is v_i for all jobs;
3) Unrelated machines in parallel: speed of machine i for job j is v_{ij};
4) Flow shop: machines are in series; every job has to be processed on each of the machines in the

same sequence.

5) Job shop: a job can visit any subset of the machines in any order.

See [21] for more details on past research on more general machine scheduling problems.

2. Literature Review: Past Research on Single-Mode RCPSPs

2.1 Basic SM-RCPSP

This section discusses some prior modeling approaches for SM-RCPSP problems found in the literature. We assume a set of tasks must be performed and that tasks are numbered 1 to n, where tasks 1 and n represent "dummy" tasks denoting the beginning and the end of a project. Let p_i denote the duration or process time of task i, for $i = 1, \ldots, n$ and let s_i and f_i denote decision variables for the starting and finish time for task i, for $i = 1, \ldots, n$. Suppose task i is ready for processing no sooner than time g_i and must be completed by its deadline d_i. Let r_{ik} denote the fixed resource requirement of task i for resource type k, for $i = 1, \ldots, n$ and $k = 1, \ldots, K$, and let R_k denote the fixed availability of resource type K in any period. Our goal is to minimize the total time it takes to process all tasks, i.e., to minimize the finish time of task n. This problem can be formulated as follows (see [16]):

$$\textit{minimize} \quad f_n \tag{1}$$

subject to:

$$\textit{Startup:} \quad f_1 = 0 \tag{2}$$

$$\textit{Precedence Type 1:} \quad s_j \geq s_i + SS_{ij}, \quad i, j \in H_1, i \neq j, \tag{3}$$

$$\textit{Precedence Type 2:} \quad f_j \geq s_i + SF_{ij}, \quad i, j \in H_2, i \neq j, \tag{4}$$

$$\textit{Precedence Type 3:} \quad s_j \geq f_i + FS_{ij}, \quad i, j \in H_3, i \neq j, \tag{5}$$

$$\textit{Precedence Type 4:} \quad f_j \geq f_i + FF_{ij}, \quad i, j \in H_4, i \neq j, \tag{6}$$

$$\textit{Task Processing:} \quad f_i \geq g_i + p_i, \quad i = 1, \ldots, n, \tag{7}$$

$$\textit{Due Date:} \quad f_i \leq d_i, \quad i = 1, \ldots, n, \tag{8}$$

Resource Capacity: $\sum_{i \in S_t} r_i \le R_k$, $t = 1, \ldots, T, k = 1, \ldots, K.$ (9)

H_1, H_2, H_3, H_4 are sets of pairs of tasks indicating precedence relations of the type SS_{ij}, SF_{ij}, FS_{ij}, and FF_{ij}, respectively (defined below), S_t denotes the set of projects in progress in time interval $[t-1, t]$, i.e.,

$$S_t = \{i \mid f_i - p_i < t \le f_i\}, \tag{10}$$

and T denotes some upper bound on the processing time of all tasks. (Note that the above formulation is more of a "conceptual" formulation, in that we implicitly assume that we know which jobs are in process in the time interval $[t-1, t]$ in constraint (9); the set S_t is, however, a function of the decision variables s_i and f_i. Many of the formulations that follow are expressed similarly for brevity of exposition.) The following describe the sets of precedence relation types:

H_1: If $(i, j) \in H_1$, then the start time of task j must follow the start time of task i plus some constant, SS_{ij}.

H_2: If $(i, j) \in H_2$, then the finish time of task j must follow the start time of task i plus some constant, SF_{ij}.

H_3: If $(i, j) \in H_3$, then the start time of task j must follow the finish time of task i plus some constant, FS_{ij}.

H_4: If $(i, j) \in H_4$, then the finish time of task j must follow the finish time of task i plus some constant, FF_{ij}.

The remaining constraints ensure that we cannot start a job before it is ready, and that we finish each job before its due date, while respecting resource capacity limits. We can further specify time windows for every project, $[EF_j, LF_j]$, which denote the earliest and latest finishing time for task j, and $[ES_j, LS_j]$ which denote the earliest latest starting time for task j.

This problem has been extensively studied in the literature. Previous research on algorithms for providing optimal solutions typically involves the use of mathematical programming techniques and

implicit enumeration, e.g., dynamic programming and branch-and-bound. A variety of branch-and-bound algorithms have been developed for this problem. Most of these use partial feasible schedules as a starting point. The branching process extends the partial schedule until finding a complete schedule. Different branching and pruning methods are used in the process; see [16]. A depth-first branching strategy resolves resource capacity violations by delaying a minimal subset of activities. Two dominance rules are used to prune the search tree. One is a left-shift dominance rule, the other is a cutset dominance rule. Under a left-shift dominance rule, if the current decision activity, *i*, can be left-shifted (scheduled earlier in time) without violating precedence or resource constraints, the corresponding partial schedule is dominated. Under a cutset dominance rule, a cutset is defined as the set of unscheduled activities for which all predecessor activities belong to the current partial schedule. Consider a cutset C_m at time *m*, which contains the same activities as a cutset C_k that was previously saved during the search of another path in the search tree. If time *k* is not greater than time *m* and if the activities in progress at time *k* did not finish later than the maximum of *m* and the finish time of the corresponding activities in the partial schedule, then the current partial schedule is dominated. The cutset dominance rule ranks among the most effective dominance pruning rules. A multitude of priority-rule based scheduling heuristics have been proposed for finding good solutions for the problem; see [16].

2.2 SM-RCPSP with Preemption

This problem allows tasks to be preempted during processing at integer points in time, i.e., the fixed integer processing time p_i of project *i* may be split into $j = 1, 2, \ldots, p_i$ process units. Suppose we have only FS type precedence relation constraints and let f_{ij} equal the integer finish time for processing unit *j* of task *i*. f_{i0} is the earliest time that task *i* can be started, where f_{i0} equals the latest finish time of all

the predecessors of task i. Let S_t denote the set of tasks in progress in the interval $[t - 1, t]$ (a task i is in progress if one of its process units $j = 1, 2, \ldots, p_i$ is in progress at time t). Let H denote the set of pairs of tasks with precedence relations, i.e., the pair (i, j) is in H if either task i must fully finish before task j can start or vice versa. The minimum makespan problem with preemption can be formulated as follows (see [7]):

$$\textit{minimize} \qquad f_{n0} \tag{11}$$

subject to:

$$\textit{Startup:} \qquad f_{10} = 0 \tag{12}$$

$$\textit{Precedence:} \qquad f_{j0} \geq f_{ip_i} + 1, \qquad i, j \in H,\ i \neq j, \tag{13}$$

$$\textit{Task ordering:} \qquad f_{i,j-1} + 1 \leq f_{ij}, \qquad i = 1, \ldots, n,\ j = 1, \ldots, p_i, \tag{14}$$

$$\textit{Resource capacity:} \qquad \sum_{i \in S_T} r_{ik} \leq R_k, \qquad t = 1, \ldots, T,\ k = 1, \ldots, K. \tag{15}$$

Davis and Heidorn [5]developed an implicit enumeration scheme based on splitting projects into unit length process tasks. Kaplan [17] presented a dynamic programming formulation for this problem. Several branch-and-bound algorithms have also been proposed; see [16].

2.3 General Precedence Constrained SM-RCPSP

This problem specifies a minimal and maximal time lag between tasks. A minimal time lag specifies that a project can only start or finish when the predecessor project has already started (finished) for a certain time period. A maximal time lag specifies that a project can only start or finish at the latest a certain number of time periods beyond the starting or finish of another project. Letting S_i denote the start time of task i and letting l_{ij} equal the required lag time between tasks i and j, these different precedence relations can be reduced to a single constraint form [16]:

$$S_i + l_{ij} \leq S_j \tag{16}$$

The minimum makespan problem can be formulated as follows:

$$\textit{minimize} \quad f_n \tag{17}$$

subject to:

$$\textit{Startup:} \quad S_1 = 0 \tag{18}$$

$$\textit{Precedence:} \quad S_i + l_{ij} \le S_j, \quad i, j \in E \tag{19}$$

$$\textit{Resource capacity:} \quad \sum_{i \in S_T} r_{ik} \le R_k, \quad t = 1, \ldots, T, k = 1, \ldots, K, \tag{20}$$

where E is the set of pairs of tasks with precedence relationships. This problem is strongly NP-hard. The only optimal solution procedures are the branch-and-bound approaches. Heuristic procedures such as truncated branch-and-bound techniques and priority-rule methods have also been developed; see [16].

2.4.1 General SM-RCPSP

The General SM-RCPSP considers regular performance measures in its objective function, including:

1) minimization of the makespan
2) minimization the weighted delay
3) minimization the total number of tardy activities
4) minimization the weighted mean flow time

Branch-and-bound algorithms used in general precedence SM-RCPSP can be used for a regular objective problem as well; however, papers on this research are scarce.

3. Past Research on Multi-Mode RCPSP

In the MM-RCPSP problem, a set of processing modes is available for each task. Using mode m for task i requires a processing time p_{im} and requires a constant amount r_{im} of resource m during each period while task i is in progress. Once a mode is selected, the task continues in this mode until finishing. We

next discuss past research within this class of problems.

3.1 Nonrenewable Resource MM-RCPSP

A nonrenewable resource MM-RCPSP allocates a nonrenewable resource to tasks. A cost of resource m while processing task i, c_{im}, is associated with this mode. If the mode set for every project can be represented as a closed interval and c_{im} is an affine and decreasing function of p_i, we have a linear time-cost tradeoff problem. If the modes are a discrete set and c_{im} is decreasing in p_i, we have a discrete time-cost tradeoff problem.

There are two related problems within this class: the budget problem (given a nonnegative budget, minimize the makespan), and the deadline problem (given a deadline for each task, minimize the total cost), which are both in the class of non-regular objective RCPSPs. For the linear time-cost tradeoff, Kelly [18] and Fulkerson [11] developed an algorithm that iteratively calculates a sequence of cuts in the AOA (activity on arc) network to compute the project cost curve. The cut can be determined by a maximum flow computation in which the capacities are derived from the slopes of the linear cost functions of the critical activities.

For the discrete case, Hindelang and Muth [15] proposed dynamic programming, and Harvey and Patterson [14] proposed an enumeration algorithm. The currently best known algorithms still rely on dynamic programming, but additionally exploit the decomposition structure of the underlying network (It is known as modular decomposition or substitution decomposition). Skutella [22] further developed approximation algorithms for this class of problems.

3.2 Renewable Resource MM-RCPSP

The renewable resource MM-RCPSP problem considers the processing time of a task as a discrete, non-increasing function of the amount of a renewable resource. We let $x_{imt} = 1$ if task i is performed in

mode m and started at time t, and zero otherwise. Let e_i (l_i) represent the critical-path-based earliest (latest) start time of task i based on the modes with the smallest process time and define T as an upper bound on the project duration. We also create a dummy start node 1 and dummy end node n, each with a single execution mode. The problem can be formulated as follows [16]:

$$\textit{minimize} \quad \sum_{t=e_n}^{l_n} t x_{nt} \tag{21}$$

subject to:

$$\textit{Task start windows:} \quad \sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1, \quad i = 1, 2, ..., n \tag{22}$$

$$\textit{Precedence:} \quad \sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} (t + p_{im}) x_{imt} \le \sum_{m=1}^{M_j} \sum_{t=e_j}^{l_j} t x_{jmt}, \quad (i, j) \in H \tag{23}$$

$$\textit{Resource capacity:} \quad \sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{im} \sum_{s=\max(t-p_{im}, e_i)}^{\min(t-1, l_i)} x_{ims} \le R, \quad t = 1, 2, ..., T \tag{24}$$

$$\textit{Integrality:} \quad x_{imt} \in \{0,1\}, i = 1, 2, .., n; m = 1, 2, .., M_i; t = 0, 1, .., T \tag{25}$$

The objective function once again minimizes the makespan of the project (such that the project ends within some predefined time window [e_n, l_n]). The first constraint requires that we start task i within its earliest and latest start time window in one of the available modes for the task. The second constraint enforces precedence relations, defined by the set H. The third constraint set enforces obeying resource limits in every period.

Demeulemeester, Reyck, and Herroelen presented a branch and bound procedure based on the concept of maximal activity-mode combination; see [8]. An activity-mode combination is a subset of activities executed in a specific mode; it is maximal if no other activity can be added without causing a resource conflict. Sprecher and Drexl developed a branch-and-bound procedure that relies on an enumeration scheme based on the precedence tree concept; see [24]. Several heuristic algorithms have

been developed for this problem as well. The most effective is the recent generalized version of a genetic algorithm approach by Hartmann [13].

3.3 General MM-RCPSP

The general MM-RCPSP includes time/cost, time/resource and resource/resource tradeoffs, renewable and nonrenewable and double constrained resources (resources limited on both a per period and a total project basis) and a variety of regular objective functions. The modes reflect alternative combinations of resource types and resource quantities to fulfill the activities. The activities can be accelerated by raising the resource quantities (time/resource trade off or time/cost trade off), or by raising the quantities of some resources and reducing the quantities of other resources (resource/resource trade off). Sprecher and Drexl [24] extend their algorithm for the renewable resource MM-RCPSP to some regular objective problems, while Bottcher, Drexl, abd Kolisch [1] provide a branch-and-bound algorithm for a partially nonrenewable resource.

4. Past Research on RCPSP with Non-regular objective functions

4.1 Resource leveling problem

In the resource leveling problem, the main objective is to limit the amount of resource usage variation from period to period, which may be costly in certain contexts. Suppose we have a set K of resources. Let c_k denote the cost per unit of resource k and let $r_k(S, t)$ denote the total usage of resource k at time t under some schedule S. The objective function can be written as:

$$\text{minimize} \sum_{k \in K} c_k f(r_k(S,t)) \tag{26}$$

$f(r_k(S, t))$ is generically referred to as the usage function. Three kinds of usage functions have been considered in the literature: a resource investment function, a resource consumption deviation function, and a variation of resource utilization over time function, respectively, see [16]. Exact algorithms for

this problem are based upon enumeration, integer programming or dynamic programming. Zimmermann and Engelhardt [26]devised a time-window based branch-and-bound algorithm. Several heuristic procedures based on a priority-rule method have been presented [16].

4.2 Net Present Value RCPSP

The deterministic unconstrained maximum Net Present Value (NPV) problem is sometimes referred to as the payment scheduling problem where processing each task involves a series of cash flow payments and receipts. Recently the most efficient algorithm is due to Demeulemeester [9] and is based on the concept of an early tree (which spans all activities scheduled at their earliest completion times and which corresponds to a feasible solution with a project duration equal to the critical path length); see [16].

4.3 Minimum Total Cost Problem (Deadline RCPSP)

Given a deadline for all tasks, consider the availability of an additional resource to finish the tasks on time. Deckro and Hebert [6] proposed such a problem. Assuming that an additional unit of the renewable resource $r \in R$ (where R is a set of available resources) provided in period t is available at a cost of C_r and is limited to a percentage of the regular period capacity, K_r, the objective is to find a feasible schedule with the least *additional* cost. Let O_{rt} denote the total overflow (or overtime) usage of resource r in period t. The problem can be formulated like the MM-RCPSP problem (see [19]) with a few extensions. The resource availability for resource r, K_r, is augmented by the additional capacity O_{rt} in every period, and O_{rt} can be no more than o_r times K_r in any period. The problem is formulated as follows:

$$minimize \quad \sum_{r\in R} c_r \sum_{t=1}^{l_n} O_{rt} \tag{27}$$

subject to:

$$\sum_{m=1}^{M_i}\sum_{t=e_i}^{l_i} x_{imt} = 1, i = 1,2,...,n \tag{28}$$

$$\sum_{m=1}^{M_i}\sum_{t=e_i}^{l_i} (t + d_{im}) x_{imt} \le \sum_{m=1}^{M_j}\sum_{t=e_j}^{l_j} t x_{jmt}, (i,j) \in H \tag{29}$$

$$\sum_{i=1}^{n}\sum_{m=1}^{M_i} r_{im} \sum_{s=\max(t-d_{im},e_i)}^{\min(t-1,l_i)} x_{ims} \le K_r + O_{rt}, t = 1,2,...,T, r \in R \tag{30}$$

$$O_{rt} \le o_r K_r, r \in R, t = 1,..,T \tag{31}$$

$$x_{imt} \in \{0,1\}, i = 1,2,..,n; m = 1,2,..,M_i; t = 0,1,..,T \tag{32}$$

This formulation is very similar to the previous one, except that here we minimize total overtime usage, which has some upper limit, and which is proportional to the regular time in a given period. Based on the idea of Burgess and Killebrew [3], a heuristic for the single-mode case is proposed. This algorithm uses two lexicographically ordered priority rules, using left shifts (starting an activity earlier) and right shifts (delaying an activity) until no improvement is available in the objective function. In the multi-mode case, heuristics generally assign tasks to the lowest labeled mode, calculate the slack (the latest finishing time minus the earliest finishing time) of each project, and implement a mode-changing scheme until finding no improvement in cost [19].

5. Bin-packing-related RCPSP

We consider two types of bin-packing related scheduling problems. One is called the integral problem, in which a task must be completely put into a single bin. The other is called the fractional problem, where a fraction of a task can be put into different bins. This latter problem (with preemption allowed) can be solved by Linear programming. Research efforts are therefore typically focused on the

more difficult integer programming versions of the problem.

5.1 BP-RCPSP (multidimensional BP) Without Precedence Relations

In this problem each task requires one unit of time for a subset of resources $i = 1, 2, \ldots, s$. The resource requirements for tasks $j = 1, 2, \ldots, n$ can be viewed as vectors $v_1, v_2, \ldots, v_n \in [0,1]^s$ (vector element v_{ij} equals 1 if task j requires resource i and zero otherwise). Resource i has a capacity l_i per time period, and we let $l = (l_1, l_2, ..., l_s)$. Processing task j in a time period means putting vector v_j into a bin. We wish to pack all vectors into a minimum number of bins (number of time periods) such that in each bin B and for each coordinate i, $i = 1, \ldots, s$, we satisfy the capacity restrictions for each resource, i.e., $\sum_{v_j \in B} v_{ij} \le l_i$. Let L_R denote the maximum percentage capacity utilization among all resource types, i.e., $L_R = \left\lceil Max_{1 \le i \le s} (1/l_i) \sum_{j=1}^{n} v_{ij} \right\rceil$. The objective of this problem is to minimize L_R such that all items are packed in a bin.

When $s = 1$ this is the classic one-dimensional bin packing problem, which has been studied extensively. Garey, Graham, Johnson and Yao [12] adapted the First Fit (FF) and First Fit Decreasing (FFD) bin packing heuristics to the multi-dimensional case. Fernandez and Lueker [10] show that for any asymptotic worst-case ratio, there is a linear-time algorithm. A polynomial time approximation algorithm with asymptotic worst-case ratios equal to one exists for this problem [4].

5.2 General BP-RCPSP

General BP-RCPSP can have precedence constraints and/or starting time constraints. The problem can be described as follows: Consider a task set $L = \{x_1, x_2, \ldots, x_n\}$, a partial order defined on L (precedence constraints), a finite resource set $\{R_i: i = 1, ..., s\}$ and a function $R_i(x)$ that specifies the consumption of resource i under the assignment of tasks x to that resource. The problem is to find an ordered partition $P = \{B_1, B_2, \ldots, B_m\}$ (m is latest scheduling time) of task set L into nonempty sets such

that:

1) for all i and j, $\sum_{x \in B_j} R_i(x) \leq 1$

2) for all $x \in L$, the sequence in the bins satisfies all precedence constraints.

3) for all partitions P' satisfying 1) and 2), $|P| \leq |P'|$

See [12] for more details on this problem.

Garey, Graham, Johnson and Yao [12] analyzed FF and FFD algorithms for this problem and proposed an FFL (first fit level) algorithm, which reorders L according to precedence levels. Srivastav, Stangier [25] developed a tight approximation algorithm for the problem with start time constraints. This method starts with a fractional solution for the system with tighter constraints and randomly rounds the fractional solution to an integral one.

Jansen and Ohring [20] considered the minimum number of machines problem (which belongs in the category of time constrained problem, and the total execution time of a machine is 1). They proposed several coloring methods (the algorithm uses coloring of a conflict graph where the nodes in a color set have no conflict.) to partition jobs into independent sets before using bin-packing heuristics.

6. A New Model for a Minimum Cost Resource Constrained Problem

In this section we discuss a new version of the RCPSP with an objective function based on combined resource usage and lateness penalty costs. We consider the problem of scheduling a set of tasks on a single resource during some finite time horizon of length T and assume that time is separated into a set of discrete periods of equal length and we index periods by t, where $t = 1, \ldots, T$. Let j index the set J of tasks where $j = 1, \ldots, M$. Task j has a due date D_j and a required processing time of P_j. The resource has an available processing time of R_t on day t during regular time, plus an additional O_t for overtime. We assume for clarity of explanation that $R_t = RT$ and $O_t = OT$ for $t = 1, \ldots, T$. Processing

during regular time incurs cost at a rate of c_1 per unit time, while processing during overtime incurs cost at a rate of c_2 per unit time, where $c_2 > c_1$, i.e., overtime is more expensive.

Task j incurs a cost of L_j per period late, and so if F_j denotes the finish period of project j, then its lateness cost equals $L_j[F_j - D_j]^+$, where $[x]^+ = \max\{x, 0\}$. We initially assume that once a task is started it can be stopped and restarted at a later time, i.e., preemption is allowed, although we later consider an alternate formulation that does not allow preemption. Let w_{jt} denote a decision variable for the amount of work performed on task j in period t. The total amount of work performed in period t then equals $W_t := \sum_{j=1}^{M} w_{jt}$, which cannot be greater than $RT + OT$ in any given day. The regular time cost for period t will equal $c_1 \cdot \min\{RT, W_t\}$, while the overtime cost for period t will equal $c_2 \cdot [W_t - RT]^+$. Let x_{jt} equal 1 if we work on task j in period t and 0 otherwise. Note that $F_j \geq t x_{jt}$ and $w_{jt} \leq (RT + OT)x_{jt}$. We can formulate the problem of minimizing total resource usage plus lateness cost as follows:

$$\textit{minimize} \quad \sum_{t=1}^{T}\left[c_1 \min\{RT, W_t\} + c_2 \left(W_t - RT\right)^+\right] + \sum_{j \in J} L_j \left(F_j - D_j\right)^+ \tag{33}$$

subject to:

$$\textit{Work Content:} \quad W_t = \sum_{j=1}^{M} w_{jt}, \quad t = 1, \ldots, T, \tag{34}$$

$$\textit{Capacity Limits:} \quad W_t \leq RT + OT, \quad t = 1, \ldots, T, \tag{35}$$

$$\textit{Task Requirements:} \quad \sum_{t=1}^{T} w_{jt} = P_j, \quad \text{for all } j \in J, \tag{36}$$

$$\textit{Task Forcing:} \quad w_{jt} \leq (RT + OT)x_{jt}, \quad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{37}$$

$$\textit{Task Finish:} \quad F_j \geq t x_{jt}, \quad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{38}$$

$$\textit{Nonnegativity:} \quad w_{jt} \geq 0, \quad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{39}$$

$$\textit{Integrality:} \quad x_{jt} \in \{0, 1\}, \quad \text{for all } j \in J,\ t = 1, \ldots, T. \tag{40}$$

The objective function (33) minimizes total work-related and lateness costs. The work content constraints (34) track the total work performed in a period, and the capacity limits (35) restrict the total work content in a period to be less than available resource capacity. The task requirements constraint (36) ensures that each task completes its processing time. The task forcing constraints (37) do not allow work to be performed on a job in a period if the indicator variable for that job in the period equals zero. The task finishing constraints (38) keep track of the finish time of each job in order to keep track of lateness. Note that we can formulate this as a mixed integer linear program using extra variables, eliminating the nonlinearity in the objective function. Consider the following reformulation:

$$\textit{minimize} \qquad \sum_{t=1}^{T}\left[c_1 y_t + c_2 u_t\right] + \sum_{j\in J} L_j v_j \tag{41}$$

subject to:

$$u_t + y_t = \sum_{j=1}^{M} w_{jt}\,, \qquad t = 1, \ldots, T, \tag{42}$$

$$u_t \geq \sum_{j=1}^{M} w_{jt} - RT, \qquad t = 1, \ldots, T, \tag{43}$$

$$v_j \geq F_j - D_j, \qquad \text{for all } j \in J, \tag{44}$$

$$\sum_{j=1}^{M} w_{jt} \leq RT + OT, \qquad t = 1, \ldots, T, \tag{45}$$

$$\sum_{t=1}^{T} w_{jt} = P_j\,, \qquad \text{for all } j \in J, \tag{46}$$

$$w_{jt} \leq (RT + OT)x_{jt}, \qquad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{47}$$

$$F_j \geq t x_{jt} \quad , \qquad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{48}$$

$$w_{jt},\ y_t,\ u_t,\ v_j \geq 0, \qquad \text{for all } j \in J,\ t = 1, \ldots, T, \tag{49}$$

$$x_{jt} \in \{0, 1\}, \qquad \text{for all } j \in J,\ t = 1, \ldots, T. \tag{50}$$

These two formulations are equivalent, but in this new formulation, under the relaxation of the integrality restriction on the x_{jt} variables, we have a linear program which is easy to solve and will provide a lower bound on the optimal solution value.

6.1 Non-preemption case: mixed integer programming formulation

We next consider the case in which preemption of a task is not allowed—once an operation is started it cannot be interrupted. Here we assume that each task contains a release period denoted by SS_j. Let w_{jt} denote amount of the resource used for project j in period t. Let z_{ij} denote a binary *sequence* indicator variable for projects i and j, i.e. $z_{ij} = 1$ implies that task j precedes task i, while $z_{ij} = 0$ implies task i precedes task j. Additionally, let l_{ij} denote the required lag time between the start period of projects i and j, where tasks i and j belong to a set H of pairs of jobs with precedence relationships (we use the notation (i, j) to denote the pair of tasks i and j).

Resource costs are a piecewise linear convex function of $\sum_{j=1}^{n} w_{jt}$ (the total resource usage in period t) with slope c_1 for $\sum_{j=1}^{n} w_{jt} \le RT$ and slope c_2 for $RT < \sum_{j=1}^{n} w_{jt} \le OT$, where $c_2 \ge c_1$. Our goal is to minimize total direct resource costs plus tardiness costs over the horizon while completing all projects.

$$\textit{minimize} \sum_{t=1}^{T} (c_1 y_t + c_2 u_t) + \sum_{j=1}^{n} L_j v_j \qquad (52)$$

Subject to:

$$u_t + y_t = \sum_{j=1}^{M} w_{jt}, \qquad t = 1, \ldots, T, \qquad (53)$$

$$u_t \ge \sum_{j=1}^{M} w_{jt} - RT, \qquad t = 1, \ldots, T, \qquad (54)$$

Lateness Tracking: $v_j \ge F_j - D_j$, for all $j \in J$, (55)

Capacity Limits: $\sum_{j=1}^{M} w_{jt} \le RT + OT$, $t = 1, \ldots, T$, (56)

Task Requirements: $\sum_{t=1}^{T} w_{jt} = P_j$, for all $j \in J$, (57)

Task Forcing: $w_{jt} \le M x_{jt}$, for all $j \in J$, $t = 1, \ldots, T$, (58)

Start/Finish Relations:	$F_j \geq S_j + \sum_{t=1}^{T} x_{jt} - 1,$	for all $j \in J$,	(59)
Task Start Time:	$S_j \geq SS_j,$	for all $j \in J$,	(60)
Sequencing 1:	$S_j \geq F_i - Mz_{ij},$	for all $(i, j) \in J$, $i \neq j$,	(61)
Sequencing 2:	$S_i \geq F_j - M(1 - z_{ij}),$	for all $(i, j) \in J$, $i \neq j$,	(62)
Precedence:	$S_i + l_{ij} \leq S_j,$	for all $(i, j) \in H$	(63)
Binary Variables:	$x_{jt}, z_{ij} \in \{0,1\}$	for all $i, j \in J$, $i \neq j$,	(64)
Integrality:	$F_j, S_j \geq 0$, integer	for all $j \in J$,	(65)
Nonnegativity:	$w_{jt}, u_t, y_t, v_j \geq 0$	for all $j \in J$, $t = 1,\ldots, T$.	(66)

Constraints (53) and (54) keep track of our regular and overtime usage. Constraint (55) tracks the lateness of each project, while (56) limits the capacity available in a period. Constraint (57) ensures each task is fully completed, and (58) keeps track of which projects are worked on in any period *t*. Constraint (59) forces the finish period for a project to equal or exceed the start period plus the number of days worked on the task and constraint (60) forces the start period of a project to equal or exceed its release period. Constraints (61) and (62) enforce our nonpreemption relationships between each pair of tasks by ensuring that either task *i* begins after task *j* finishes (or vice versa) for all (*i*, *j*) task pairs. Constraint (63) enforces precedence relationships where necessary, while (64), (65), and (66) encode our variable restrictions.

6.2 Heuristic Algorithm for the Non-preemption Case

Since the RCPSP is in general NP-Hard, we seek a good heuristic algorithm for minimizing total resource consumption plus lateness costs. This section describes an algorithm for the minimum cost non-preemption RCPSP described in the previous section. For simplicity the algorithm we present assumes that no precedence constraints exist for pairs of jobs, although we can easily extend this

algorithm to account for precedence constraints. The heuristic first sorts tasks in non-decreasing order of the task release period, SS_j. Then in increasing index order, we attempt to schedule tasks at the earliest time possible following the release period. Our first goal is to try to schedule the job using regular resource consumption time. If the job is fully scheduled in regular time and is not late, we consider the task scheduled and not subject to further change. If scheduling the job only in regular time results in late delivery of the job, we then attempt to reduce the delivery time by scheduling part of the job using overtime, if the overtime costs are less than the lateness costs; otherwise we deliver the project late (since this is cheaper than delivering on time). Using the notation in the previous section, the algorithm is described as follows:

Non-preemption RCPSP Heuristic Algorithm

STEP 0: Sort projects in nondecreasing order of start date, SS_j, Set $j = 0$, $C_0 = 0$, and $P_0 = 0$.

STEP 1: Set $j = j + 1$; if $j = n + 1$ stop with complete schedule.

Otherwise, if $C_{j-1} \geq SS_j$, start project j in period C_{j-1} and let $S_j = C_{j-1}$; set $p_j = P_j$. Go to STEP 2.

Otherwise, start project j in period SS_j and let $S_j = SS_j$; set $p_j = P_j$. Go to STEP 2.

STEP 2: Beginning with $t = S_j$, allocate $\min\{SL - r_t, p_j\}$ units of time to project j in successive periods. Each time a portion (or all) of the project has been allocated to some resource in period t, set $p_j = p_j - \min\{SL - r_t, p_j\}$. Stop when $p_j = 0$. Let R_j be the proposed finish time of project j, where R_j is determined by the smallest index k such that

$$\sum_{t=S_j}^{k} (SL - r_t) \geq P_j$$

STEP 3: If $[R_j - D_j]^+ \leq 0$, go to STEP 1.

Otherwise, set $\tau = [R_j - D_j]^+$. Project j completion is currently scheduled to finish τ periods after its due date. Continue to STEP 4.

STEP 4: Calculate d_τ, the total resource time needed to reduce the completion date of project j by τ periods.

If $L_j\tau - c_2 d_\tau \geq 0$ and $d_\tau < \sum_{t=S_j}^{D_j} HL - r_t$, let $\alpha = d_\tau$ and let $t = S_j$.

a. Add $\Delta_t = \min\{HL - r_t, \alpha\}$ to the resource consumption in period t.

b. Let $\alpha = \alpha - \Delta_t$ and let $r_t = r_t + \alpha$. If $\alpha = 0$ go to STEP 1.

c. Otherwise let $t = t + 1$, recomputed d_τ and repeat STEP 4a.

Otherwise, let $\tau = \tau - 1$.

If $\tau > 0$, repeat STEP 4.

Otherwise, go to Step 1.

Step 1 of the algorithm checks whether a task can begin in its earliest start period or if it must wait until the previously scheduled job finishes. In either case, the job starts as early as possible following its earliest start period. Step 2 sequentially allocates available "regular time" resource availability to the current task, until the entire task is tentatively scheduled. Step 3 then checks whether the job will be delivered late, based on the current partial schedule. If the task will not be late, we move on to the next job in the sequence. If the job will be delivered late, we first determine if expediting the job to an earlier finish time (using overtime) costs less than the lateness cost under the currently planned schedule. If expediting the job results in savings over the currently planned schedule, we use overtime beginning in the first scheduled period for the job and complete the job as early as possible, as long as the overtime cost does not exceed the lateness cost. When we have completed overtime scheduling for the current job, we then move to the next job in the sequence until all jobs have been scheduled.

We can view this algorithm in the bin-packing context, where we can split a single item among multiple bins. We first use bins with a size equal to the regular time available. We then consider expanding the bin size (by allowing for overtime) and using fewer bins to accommodate the task. The extra bin capacity comes at a cost (the overtime cost), and we compare whether this cost is less than the cost of using additional bins (or delivering the task after its due period). If the extra bin capacity results

in savings, we enlarge some of the bins to fit the task into fewer bins.

7. Summary and Future Directions

The purpose of this paper was to document recent literature on resource-constrained project scheduling problems (RCPSPs). We separated the RCPSP into several subclasses and summarized recent research directions, algorithms and heuristic approaches. Although the literature on RCPSPs has been abundant over the past 40 years, we found that there has been relatively little activity in this area that considers minimizing combined resource and lateness costs, which is a relevant concern in many practical scheduling contexts. We provide a new model and mixed integer programming formulation for addressing this class of problems. We presented a heuristic algorithm for this problem class in cases where task preemption is not allowed. In a follow up paper, we will further refine and generalize our heuristic algorithm to handle more general cost structures and preemption/precedence assumptions. Part of this future work will involve comprehensive computational testing to assess the performance of our heuristic approach. Furthermore, some additional assumptions (single resource, non-precedence constraint, independent resources) will be relaxed so that the algorithm can be applied to more general cases.

REFERENCES

[1] Bottcher, J., Drexl, A., and Kolisch, R. (1996) A branch and bound procedure for project scheduling with partially renewable resource constraints, Proceedings of the fifth international workshop on project management and scheduling, Poznan, April 11-13, 48-51.

[2] Brucker, P., Drexl, A., Mohring, R., Neumann, K. and Pesch, E. (1999) Resource-constrained project scheduling: Notation, classification, models and methods, *European Journal of Operational research* 112, 3-41.

[3] Burgess, A.R. and Killebrew, J.B. (1957) Variation in activity level on a cyclical arrow diagram, *The Journal of Industrial Engineering* 8, 76-83.

[4] Coffman, E.G., Garey, M.R. and Johnson, D.S. (1984) Approximation Algorithms for Bin-packing – An Updated Survey, in *Algorithm Design for Computer Design*, G. Ausiello, M. Lucertini, and P. Serafini (eds.), Springer-Verlag, 49-106.

[5] Davis, E.W. and Heidorn, G.E. (1971) An algorithm for optimal project scheduling under multiple

resource constraints, *Management Science* 27 B803-B816.

[6] Deckro, R.F. and Hebert, J.E. (1989) Resource constrained project crashing, *OMEGA International Journal of Management Science* 17, 69-79.

[7] Demeulemeester, E. (1992) Optimal algorithms for various classes of multiple resource-constrained project scheduling problems, Ph.D. Thesis, Department of Applied Economic Science, Katholieke University, Leuven.

[8] Demeulemeester, E., Reyck, B. D. and Herroelen, W. (1997), The discrete time/resource trade-off problem in project networks-A branch-and-bound approach, Research report 9717, Department of Applied Economics, K.U. Leuven.

[9] Demeulemeester, E., Herroelen, W. and Dommelen, P.V. (1996) An optimal recursive search procedure for the deterministic unconstrained MAX-NPV scheduling problem, Research Report 9603, Department of Applied Economics, K.U. Leuven.

[10] Fernandez, W. and Lueker, G.S. (1981) Bin packing can be solved within $1+\varepsilon$ in linear time, *Combinatorica* 1, 349-355.

[11] Fulkerson, D.R. (1961) A network flow computation for project cost curves, *Management Science* 7, 167-178.

[12] Garey, M.R., Graham, R.L., Johnson, D.S. and Yao, Chichin (1976) Resource constrained scheduling as generalized bin packing, *Journal of Combinatorial Theory* (A)21, 257-298.

[13] Hartmann, S. (1997) Project scheduling with multiple modes: A genetic algorithm, Mauskripte aus den Instituten fur Betriebswirtschaftslehre der University Kiel, No. 435.

[14] Harvey, R.T. and Patterson, J.H. (1979) An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis, *Found. Control Eng.* 4 , 107-117.

[15] Hindelang, T.J. and Muth, J.F. (1997) A dynamic programming algorithm for decision CPM networks, *Operations Research* 27, 225-241.

[16] Herroelen, W., Reyck, B.D. and Demeulemeester, E. (1998) Resource-constrained project scheduling: a survey of recent developments, *Computers & Ops. Res.* 25(4), 279-302.

[17] Kaplan, L. (1988) Resource-constrained project scheduling with preemption of jobs. Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

[18] Kelly, J.E. (1963) The critical path method: Resource planning and scheduling, in J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice Hall, N.J., 347-365.

[19] Kolisch, R. (1995) Project scheduling under resource constraints, Physica-Verlag Heidelberg.

[20] Jansen, J. and Ohring, S. (1997), Approximation algorithms for time constrained scheduling, *Information and computation* 132, 85-108.

[21] Pinedo, M. (1995) Scheduling theory, algorithms and systems, ISBN 0-13-706757-7, Prentice Hall.

[22] Skutella, M. (1997) Approximation algorithms for the discrete time-cost tradeoff problem, in Proceedings of the eighth Annual ACM-SIAM symposium on Discrete Algorithms, New Orleans, LA, 501-508.

[23] Sprecher, A. (1994) Resource-constrained project scheduling: exact methods for the multi-mode case, lecture Notes in Economics and Mathematical Systems, No.409, Springer, Berlin.

[24] Sprecher, A. and Drexl, A. (1998) Solving multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm, *European Journal of Operational Research* 107, 431-450.

[25] Srivastav, A. and Stangier, P. (1997) Tight approximations for resource constrained scheduling

and bin packing, *Discrete applied mathematics* 79, 223-245.

[26] Zimmermann, J. and Engelhardt, H. (1998) Lower bounds and exact algorithms for resource levelling problems, Report WIOR-517, University Karlsruhe.