1.3.4 A Formulation for the Preemptive RCPSP

The literature for the preemptive RCPSP (PRCPSP) is very limited. It is commonly known, as first shown by Blazewicz et al. (1983), that the RCPSP is $\mathcal{NP}$-hard. Since the PRCPSP can be considered as an RCPSP in which each activity with duration of p_i is replaced by independent activities with unit duration, the $\mathcal{NP}$-hardness follows. Afshar-Nadjafi (2014) study different preemption penalty costs and a weighted earliness-tardiness in the objective function of the PRCPSP. Afshar-Nadjafi and Arani (2014) published similar results for the multi-mode case. Quilliot et al. (2013) use some interesting properties of precedence graphs to develop a novel branch and bound procedure for the problem. Coffman et al. (2015) explore the smallest time between events (shift) in an optimal schedule for parallel machine scheduling problem with preemptions allowed at non-integer (rational) times. Additionally, in this article, authors introduce an alternative definition for events to account for preemptions, these are identified as an activity start, interruption, resumption or completion; let us adopt such definition.

We wish to extend the OOE formulation from Koné et al. (2011) (discussed in Section 1.3.3) to allow preemptions. For this, we need to introduce some facets. Let p_i^- be a value that represents the minimum time allowed to be spent on activity i, with $p_i^- < p_i$ (see Figure 1.3). This can also be thought of as the amount of time after starting/resuming an activity in which we cannot complete/interrupt said activity. From this, the fraction $n_i = \lfloor p_i/p_i^- \rfloor$ determines the maximum number of

subactivities activity i can be broken into. Hence,

$$\sum_{i \in A} \left\lfloor \frac{p_i}{p_i^-} \right\rfloor = \sum_{i \in A} n_i = N$$

determines the new number of events. That is, $\mathcal{E} = \{0, 1, \ldots, N\}$. Furthermore, we impose the assumption, that two subactivities can only be processed in parallel if they do not divide the same activity.

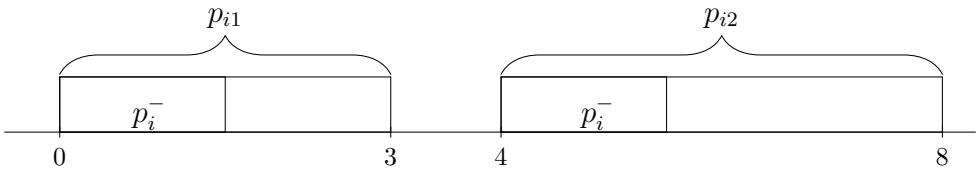

Figure 1.3: An activity broken down into two subactivities.

With this definition, we can change the constraint 1.2f to

$$p_i^- \left((z_{ie} - z_{i,e-1}) - (z_{if} - z_{i,f-1}) - 1 \right) \leq t_f - t_e \leq p_i \qquad \forall\, (e,f) \in \mathcal{E}^2\ (f > e),\ \forall\, i \in A\,. \tag{1.5}$$

This constraint bounds elapsed times between events. The upper bound on the is the total processing time, p_i. While the lower bound is set to be the minimum time allowed, p_i^-.

To monitor the progress of individual subactivities throughout the project, let us introduce a new continuous time variable, a_{ie} which represents the amount of processing time that has been dedicated to activity i by event e. In terms of the the variables introduced for Model 1.3.2, for a single process of each activity, we may write the processing variable as follows,

$$a_{ie} = \begin{cases} a_{i,e-1} + t_e - t_{e-1}, & \text{if } z_{ie} = 1 \vee z_{i,e-1} = 1 \\ a_{i,e-1}, & \text{otherwise} \end{cases} \qquad \forall\ i,\ e.$$

With $0 \leq a_{ie} \leq p_i \ \forall \ e \in \mathcal{E}$; $a_{i0} = a_{i,-1} = 0 \ \forall \ i \in A$; and, $a_{iN} = p_i \ \forall \ i \in A$. By using some big-$M$ constraints, this definition is equivalent to the following activity processing constraints,

$$a_{i0} = a_{i,-1} = 0 \qquad \forall \ i; \tag{1.6a}$$

$$a_{i,e-1} \leq a_{i,e} \qquad \forall \ i, \ e; \tag{1.6b}$$

$$a_{ie} \leq a_{i,e-1} + M \left(z_{ie} + z_{i,e-1}\right) \qquad \forall \ i, \ e; \tag{1.6c}$$

$$a_{ie} \geq a_{i,e-1} + (t_e - t_{e-1}) - M(1 - z_{ie}) \qquad \forall \ i, \ e; \tag{1.6d}$$

$$a_{ie} \geq a_{i,e-1} + (t_e - t_{e-1}) - M(1 - z_{i,e-1}) \qquad \forall \ i, \ e; \tag{1.6e}$$

$$a_{iN} = p_i \qquad \forall \ i; \tag{1.6f}$$

$$0 \leq a_{ie} \leq p_i \ \forall \ e \in \mathcal{E} \qquad \forall \ i, \ e. \tag{1.6g}$$

Constraints 1.6a initialise the dummy activity processing variable, $a_{i,-1}$, and the one corresponding first event, a_{i0}. Constraints 1.6b ensure that the processing variable in nondecreasing. Constraints 1.6c to 1.6e ensure that if either $z_{ie} = 1$ or $z_{i,e-1} = 1$ the processing variable is recursively updated, otherwise, because of M they become redundant. That is, if $z_{ie} = 1 \vee z_{i,e-1} = 1$, a_{ie} is updated with, $a_{i,e-1}$, the already processed time for activity i, plus the time elapsed, $t_e - t_{e-1}$, between events $e - 1$ and e. In fact, since constraints 1.5 bounds $t_e - t_{e-1}$ with p_i, for the same effect we can replace M with p_i. Constraints 1.6f ensure that every activity is processed to completion and least by the last event N. Constraints 1.6g bound the activity processing variable.

As for the objective function for the event-based PRCPSP we have similar options to those for the simpler version of the problem. In our notation, we have

(i) Makespan – $\min C_{\max} = t_N$.

(ii) Maximum lateness – $\min \max_{i \in A} t_{e(i)} - d_i$, where d_i is the deadline for activity i and $e(i) = \min\{e : \ a_{ie} = m_i p_i\}$ i.e. the event where the processing for activity is completed.

(iii) Preemption costs – $\min \sum_i \sum_e c_i^P (z_{i,e-1} p_i - (a_{ie} - a_{i,e-1}))$ where c_i^P is the cost of preemption for activity i.

Choosing one of these objective functions and including the constraints from Model 1.3.2, we can form a formulation for the event-based PRCPSP as follows,

Model 1.3.4. *Event-based formulation for the PRCPSP.*

$\min \quad C_{\max}$
Subject to
From Model 1.3.2
1.2*b to* 1.2*e*
1.5
1.2*g to* 1.2*l*
Earliest/Latest start times constraint
1.3
Activity Processing Constraints
1.6*a to* 1.6*g*

Example 1.3.2. Consider an extension to Example 1.3.1.

i	1	2	3	4
p_i	4	3	5	8
p_i^-	1	2	2	2
n_i	4	1	2	4
b_{i1}	3	4	2	3
b_{i2}	3	1	2	2

Table 1.1: Values for n_i for activities in Example 1.3.1.

In this case, from Table 1.1, we see that activity 2, for example, has to be processed in a single go. For the others, we have at most 4 subactivities. Thus, $N = 13$. A feasible solution for the scheduling of this project can be seen in Figure 1.4, with the corresponding values for the variables in Table 1.2.

$e = 0$:	$t_0 = 0,$	$z_{20} = 1$	$a_{20} = 0$
$e = 1$:	$t_1 = 0,$	$z_{21} = z_{11} = 1$	$a_{21} = a_{11} = 0$
$e = 2$:	$t_2 = 3,$	$z_{32} = 1$	$a_{22} = a_{12} = 3,\ a_{32} = 0$
$e = 3$:	$t_3 = 3,$	$z_{33} = z_{43} = 1$	$a_{33} = a_{43} = 0$
$e = 4$:	$t_4 = 8,$	$z_{44} = 1$	$a_{34} = a_{44} = 5$
$e = 5$:	$t_5 = 8,$	$z_{45} = z_{15} = 1$	$a_{45} = 5,\ a_{15} = 3$
$e = 6$:	$t_6 = 9,$	$z_{46} = 1$	$a_{45} = 6,\ a_{16} = 4$
$e = 7$:	$t_7 = 11,$		$a_{47} = 8.$

Table 1.2: Values for the variables in Example 1.3.2.

With $C_{\max} = 11$ and rest of the variables are 0. The Gantt chart of the two resources considered R_1 and R_2 is presented in Figure 1.4. It is worth noting that, in comparison with Example 1.3.1, the non-preemptive case, the makespan, $C_{\max}$, is reduced.

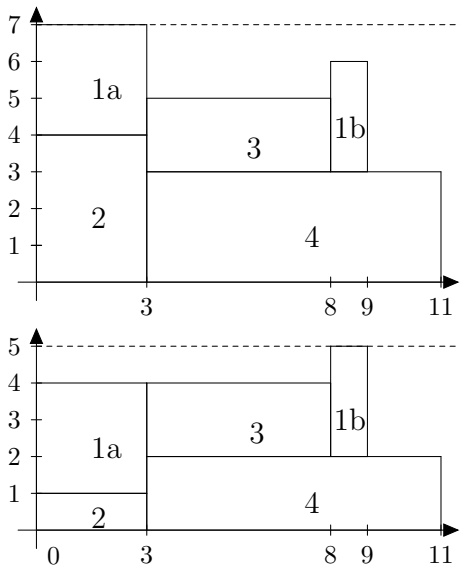

Figure 1.4: Example of a feasible preemptive schedule showing the processing of activities for two resources R_1 (top) and R_2 (bottom).

Implementation and Computational Tests

To solve Models 1.3.2 and 1.3.4 efficiently, one has to implement some preprocessing steps. We apply the well-known local constraint programming (CP) algorithm from Demassey et al. (2005) for the estimation of earliest/latest start times ES_i, LS_i. As can be seen in Figure 1.5, it involves four steps.

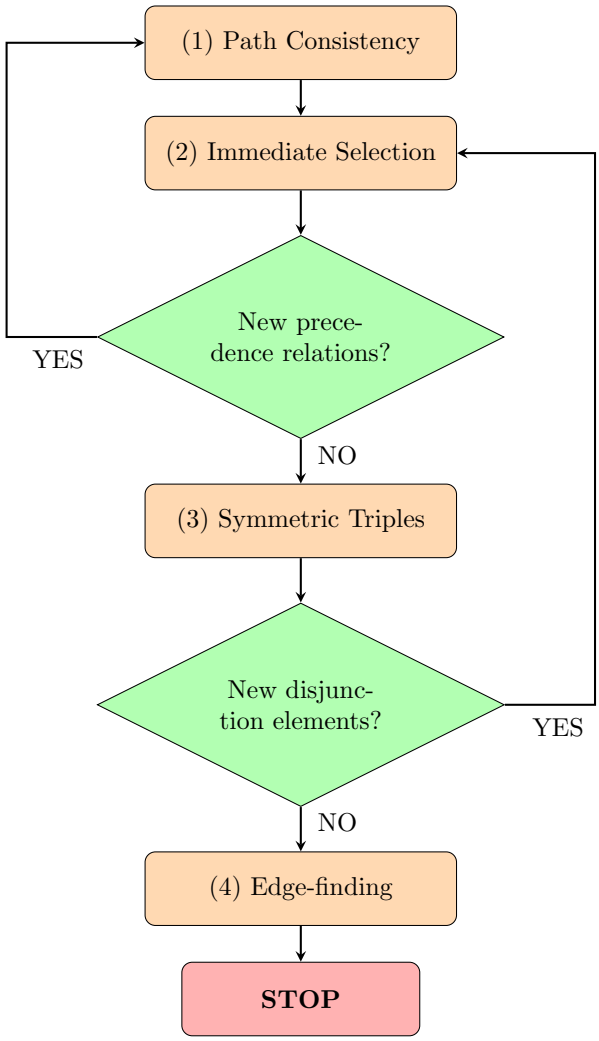

Figure 1.5: Local constraint propagation algorithm from Demassey et al. (2005).

Before we can describe the individual steps of the algorithm, we require the introduction of some notation. Let $D = (d_{ij})_{A\times A}$ be start-start distance matrix that

satisfies $S_j - S_i \geq d_{ij}$ for any feasible schedule S. To initialise this matrix, let

$$D_{ij} = \begin{cases} 0, & \text{if } i = j; \\ p_i, & \text{if } (i,j) \in E; \\ -T, & \text{otherwise;} \end{cases}$$

where T is the planning horizon. We also need to update a disjunction relation of activities J, such that $(i,j) \in J$ if activities i and j cannot be executed in parallel. Furthermore, to take resource constraints into consideration, we utilise all minimal forbidden sets of two, F_2, and three, F_3 activities. These are defined as subsets that include the minimal number of two or three activities not linked by any precedence relations and that satisfy $\sum_{j \in F} b_{ik} > B_k$ for some resource k.

Now, the four steps in Figure 1.5 can be described,

1. Path Consistency. Implements the Floyd-Warshall algorithm, which computes the transitive closure of the matrix D by setting

   $$d_{il} = \max_{j \in A}(d_{ij} + d_{jl}).$$

   Whenever just one entry d_{hl} is updated in this fashion, we set

   $$d_{ij} = \max(d_{ij}, d_{ih} + d_{hl} + d_{lj}) \; \forall (i,j) \in A \times A.$$

2. Immediate Selection. Replaces each element in the disjunction set $(i,j) \in J$ whenever $d_{ij} \geq 1 - p_j$ by the precedence constraint $i \to j$.

3. Symmetric Triples. Deduces new disjunctions considering all symmetric triples. That is, all sets F_3, minimal forbidden sets of three activities. We implement the algorithm from Brucker et al. (1998).

4. Edge-finding. Also deduces new precedence relations but employs cliques of disjunctions, sets in which each pair of activities is in disjunction. Edge-finding ultimately determines the earliest start and end times, ES_j and LS_i as follows. For each j in a clique C, if

$$\min_{i \in C} d_{0i} + \sum_{i \in C} p_i > \max_{i \in C \ , i \neq j} -d_{i0} + p_i \ ,$$

then the earliest start time of activity j, ES_j, is updated by setting,

$$ES_j = d_{0j} = \max \left\{ d_{0j} \ , \ \max_{C' \subseteq C \setminus \{j\}} \left(\min d_{0i} + \sum_{i \in C'} p_i \right) \right\} \ ,$$

and the latest start time of activity $i \neq j$, is updated by setting,

$$LS_i = d_{i0} = \max\{ d_{i0} \ , \ d_{j0} + p_i \} \ .$$

Furthermore, this step allows us to compute a lower bound for the makespan,

$$C_{\max}^{LB} = d_{0,n+1} = \max \left\{ d_{0,n+1}, \min_{i \in C} d_{0i} + \sum_{i \in C} p_i + \sum_{i \in C} b_{i,n+1} \right\} .$$

After implementing the preprocessing algorithm, we tested Model 1.3.4, allowing a single preemption, using the well-known `psplib` library instances (Kolisch and Sprecher, 1997). Particularly, we used the `j30` data set with 30 activities (parameter `1` instances `1-10`, i.e. `j301_1` to `j301_10`). Since we are only allowing one preemption, for each activity i, we set

$$p_i^- = \left\lfloor \frac{p_i}{2} \right\rfloor \ .$$

Hence,

$$N = \sum_{i \in A} \left\lfloor \frac{p_i}{p_i^-} \right\rfloor = \sum_{i \in A} \left\lfloor \frac{p_i}{\lfloor p_i/2 \rfloor} \right\rfloor \leq 2|A| \ ,$$

that is, as one would expect, the number of events is at most twice the number of activities.

The results for three different formulations are presented in Tables 1.3 (Model 1.3.4), 1.4 (the `psplib` benchmark results) and 1.5 (and Model 1.3.2). We compare the makespan (MS) and the computational times for the ten instances studied. Additionally, the local constraint programming algorithm is efficient as shown in the fourth column of Table 1.3. The makespan for Model 1.3.4 is an average of 11.6% lower than the benchmark results with a slight increment in the solution times. When compared to solutions obtained with Model 1.3.2, an average decrease of 0.8% is shown for the makespan with a lower average time.

Table 1.3

Inst.	MS	Time (s)	PP Time (s)
1	38	6.21	0.17
2	42	4.35	0.14
3	43	2.49	0.12
4	55	5.48	0.09
5	31	6.97	0.14
6	38	3.10	0.15
7	60	5.99	0.12
8	53	3.97	0.10
9	42	4.37	0.20
10	37	5.99	0.20

Table 1.4

MS	Time (s)
43	0.3
47	0.11
47	0.12
62	0.64
39	0.48
48	0.04
60	0.01
53	0.03
49	0.1
45	0.04

Table 1.5

MS	Time (s)
38	5.32
42	0.87
44	300.02
55	0.91
33	300.01
38	1.96
60	0.67
53	0.91
42	0.76
37	1.07

1.4 Maintenance Scheduling

Traditionally, machine maintenance was performed on a run-to-failure basis. In the late ‘50s and early ‘60s, with the recent developments in operational research (OR), *preventive maintenance* was introduced and maintenance planning was formulated