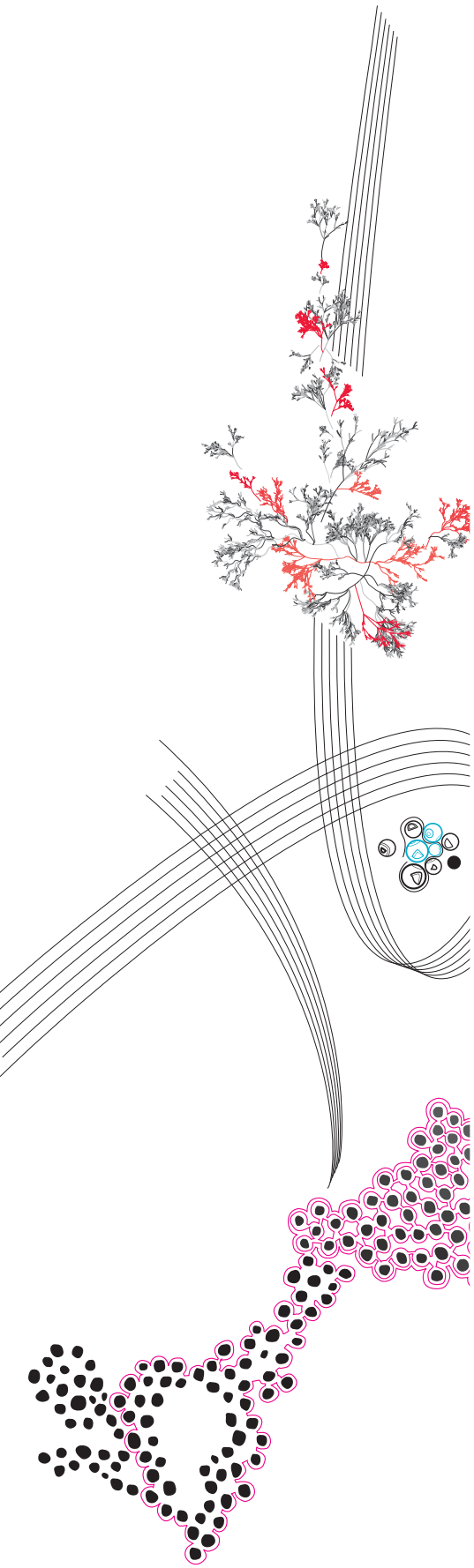


Master's Thesis Applied Mathematics

Planning in Supply Networks Using Aggregated Resource Feasibility

R.R. Mauritz
(Mathematics of Data Science, University of Twente)

Graduation Committee:
Prof. dr. M.J. Uetz (UT)
Dr. A. Antoniadis (UT)
Dr. J.C.W. van Ommeren (UT)
Drs. M. Taal (Togetr B.V.)

August, 2022

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

Planning in Supply Networks Using Aggregated Resource Feasibility

R.R. Mauritz *

August, 2022

Abstract

In this thesis, we focus on resolving time and resource conflicts in manufacturing plans while minimising the makespan. This problem is a special case of the Manufacturing Planning Problem (MPP), which can be reduced to the Resource Constrained Project Scheduling Problem (RCPSP). In order to better capture practical settings, we distinguish between detailed and aggregated resource constraints and propose a relaxation of the RCPSP to which we refer as the Window Aggregated Resource Constrained Project Scheduling Problem (WARCPSP). The WARCPSP defines resource feasibility via aggregated resource constraints by requiring that for any time window in a pre-defined window configuration, the total resource requirement does not exceed the total availability. By adjusting the window configuration to practical needs, the planner is able to fully determine the time-dependent level of aggregation. We develop a heuristic Conflict Resolution Algorithm (CRA) that resolves time and resource conflicts in a manufacturing plan using the WARCPSP model. It is designed such that it can be applied to a manufacturing plan at any time and to specific parts only, increasing its practical usage. The algorithm resolves time conflicts by using a special earliest start procedure. Resource conflicts are then resolved in an iterative, time-advancing fashion, each iteration consisting of a forward and a backward pass. Inspired by disjunctive arc based methods and minimal forbidden sets, the CRA moves activities forward by adding a precedence relation, using priority rules to make local decisions. This forward pass is then followed by a backward pass, that tries to improve the schedule by backward shifting activities. By means of experiments on instances of the PSPLIB library, we show that the CRA is able to produce close-to-optimal solutions with an average run-time that is within the order of seconds, allowing the CRA to be used in a real-time setting to support human planners.

*Email: r.r.mauritz@student.utwente.nl

Preface

This master's thesis is the final part of 6 years of studying Applied Mathematics at the University of Twente, providing me the degree of Master of Science. I look back at a beautiful period of my life that taught me so many things.

The research was conducted from February 2022 up to August 2022 at the company Togetr B.V. in Veenendaal, The Netherlands. Togetr is a software company that develops ERP software for the manufacturing industries. In order to enrich its Supply Chain Management module, they asked me to conduct a thesis at their company and provided important insights in the manufacturing planning domain.

I owe a debt of gratitude to the many people that supported me in coming to the result that is embodied by this document. First of all, I would like to thank my UT supervisors Marc Uetz and Antonios Antoniadis for their enormous help and willingness to make a success of this project. I enjoyed all of our meetings, each one giving me motivation to continue with what I was doing. The same goes for Togetr's representative Martin Taal. I thank him especially for his enthusiasm and creativity in guiding this process and coming up with interesting, practical ideas. I thank Jan-Kees van Ommeren for his willingness to be part of the graduation committee. I would like to thank Jop Zwienenberg who was conducting his master's dissertation at the same time. Our informal discussions were of great support and made me realise that I was not the only one struggling at times. I would like to thank my friends and family for their unconditional and warm support, especially my brother who was graduating at the same time. Our countless coffee breaks were a welcoming distraction. Last but not least, I would like to thank my girlfriend for just being who she is. Being together was the distraction that I needed the most.

Thank you all.

Rutger

Table of Contents

List of Abbreviations

AON Activity-On-Node

APS Advanced Planning Systems

CRA Conflict Resolution Algorithm

ERP Enterprise Resource Planning

FBI Forward-Backward Improvement

MIP Mixed-Integer-Programming

MPP Manufacturing Planning Problem

MRP Material Requirements Planning

NC Network Complexity

PARCPSP Periodically Aggregated Resource Constrained Project Scheduling Problem

PSGS Parallel Schedule Generation Scheme

RCPSP Resource Constrained Project Scheduling Problem

RF Resource Factor

RS Resource Strength

SGS Schedule Generation Scheme

SSGS Serial Schedule Generation Scheme

WARCPSP Window Aggregated Resource Constrained Project Scheduling Problem

List of Symbols

Δ_w	Window width parameter
Δ_{sz}	Window step size parameter
ℓ	Number of windows
$\hat{\mathcal{S}}_i$	(In)direct successors of activity i
$\hat{\mathcal{S}}_i$	(In)direct successors of activity ı
$\hat{T}_i$	Upper bound on move backward time range
$\mathcal{A}(S,t)$	Activities in progress in S at time t
$\mathcal{S}$	Set of time- and resource-feasible schedules
$\mathcal{S}_i$	Direct successors of activity i
$\mathcal{T}$	Time horizon
μ_{CPU}	Run-time performance measure
μ_{OPT}	Makespan performance measure
π	Priority rule
$\prec$	End-start precedence relation
A	Set of activities
c	Set of resource capacities
C_{max}^{OPT}	Best-known makespan
C_{max}	Schedule makespan
D	Decision set
d	Set of activity durations
$d_{i,w}(S)$	Duration of activity i in window w given schedule S
$ES()$	Earliest start schedule
ES	Vector of earliest start values
F	Forbidden set
G	Precedence graph
i,j	Activity indices
I	Instance of the (WA)RCPSP
k	Resource index
LB/lb	Vector of earliest start lower bound values
LF	Vector of latest finish values
LS	Vector of latest start values
m	Number of resources
n	Number of activities
O	Strict order
P	Set of precedence relations
R	Set of resources
S	Schedule
S''	Schedule after backward move
S'	Schedule after forward move
S^*	Optimal schedule
T	Right bound of the time horizon
t	Time period

t_w^l	Left bound of window w	u	Set of activity-resource requirements
t_w^r	Right bound of window w	W	Set of windows/window configuration
T_i	Lower bound on move backward time range	w	Time window (index)

Part 1

Introduction & Motivation

1.1 Manufacturing Planning and Conflict Resolution

In this section, we introduce the problem of resolving conflicts in manufacturing plans. This is a common problem in manufacturing planning and can be seen as a special case of a combinatorial optimization problem to which we refer as the Manufacturing Planning Problem. This problem of resolving conflicts in manufacturing plans serves as a motivation for developing an algorithm that automatically resolves such conflicts.

At first, we use Sect. 1.1.1 to introduce the main aspects of the Manufacturing Planning Problem. We proceed with Sect. 1.1.2, where we introduce the problem of resolving conflicts in manufacturing plans and mention two families of conflicts that frequently arise in such plans. We then use Sect. 1.1.3 and Sect. 1.1.4 to zoom into some of the main characteristics of the problem of resolving those conflicts. The first one, *aggregation*, is of importance when defining a planning model and consequently, an algorithm that is compatible with such a model. In this section we define what we mean with 'aggregated resource feasibility'. The other two characteristics, *online* and *fast*, pose a restriction on the desired properties of the algorithm.

1.1.1 Manufacturing Planning

In this thesis, we focus on the mid-term single-facility *Manufacturing Planning Problem* (MPP) that frequently arises in the manufacturing industries. The MPP can be seen as a sub-class of the Supply Chain Planning problem. The MPP can be defined as follows.

Definition 1. *The MPP is the problem of planning time- and resource-consuming tasks that are interrelated via time and demand dependencies. This planning is done with the aim of maximising company profits and customer satisfaction, while respecting constraints that are imposed by the interdependencies between the tasks and the limited capacities of the resources.*

In a manufacturing context, a task generally denotes an operation that is needed to turn raw material in an end product as required by an external demand such as sales orders, stock refills, etc. These tasks are interrelated via time and demand dependencies that follow from the Bill of Material or Routing sequence. In order for a task to be executed, resources such as employees,

machines, tools and material supply are needed. These resources posses a particular skill that is needed by the operation and have limited capacity. The network that consists of these tasks and their interdependencies is referred to as a *supply network*. An interdependence between two tasks denotes a time-wise precedence. The output of the MPP is a schedule, referred to as *manufacturing plan*. In Sect. 3.1, we provide a more precise definition of the supply network and manufacturing plan. A manufacturing plan is called *feasible* when it satisfies both the constraints imposed by the interdependencies of the supply network and the constraints imposed by the resource capacities. We refer to the first as *time constraints* and to the latter as *resource constraints*. The MPP goal of maximising company profits and customer satisfaction can be achieved in numerous ways. When a known and fixed supply network corresponds to a project linked to one and the same demand, a goal is often to minimise the overall production cycle length of the project. From a planning perspective, this means that the completion time of the last-finishing task in the manufacturing plan is minimised, in scheduling literature known as minimising the *makespan* [34]. This leads to a combinatorial optimization problem that has attracted a lot of research in the operation research community.

As Stadtler [39] points out, transactional *Enterprise Resource Planning* (ERP) systems that are used in Supply Chain Planning have a lack of strength when it comes to the core business of planning and detailed scheduling. So-called *Advanced Planning Systems* (APS) have been introduced to fill this gap. APS are based on the notion of hierarchical planning and consist of several software modules, see Fig. 1.1. The associated planning tasks can be considered at different levels of aggregation and planning intervals. The MPP that we consider can best be seen as part of the short/mid-term functionality Production Planning and short-term functionality Scheduling. Production Planning focuses mainly on defining a mid-term schedule that enables the planners to get a rough overview of the feasibility of the work to be executed and to take measures if needed. Scheduling focuses on defining a short-term schedule for the shop-floor and can be seen as the short-term execution of the prior mid-term production plans.

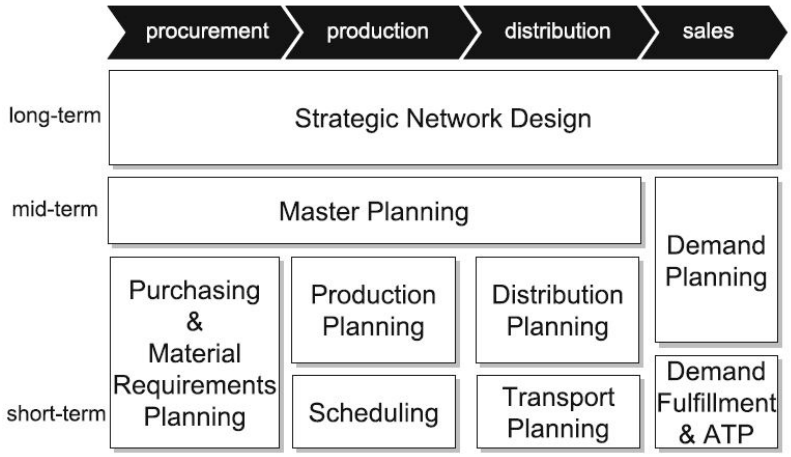


Figure 1.1: *Architecture and software modules of Advanced Planning Systems, see Stadtler [39].*

1.1.2 Conflicts in Manufacturing Plans

A common problem in manufacturing planning is the existence of conflicts in generated manufacturing plans which make the plan infeasible. As a result, these conflicts need to be resolved in order to obtain a workable plan. Resolving these conflicts is equivalent to re-generating a feasible manufacturing plan from the original infeasible manufacturing plan, making it a special case of the MPP. A manufacturing plan might contain two types of conflicts, which we refer to as *time conflicts* and *resource conflicts*.

A time conflict in a manufacturing plan is the violation of an interdependency $i \prec j \in P$ between two tasks i and j that are related to each other in time. Assuming that $\prec$ denotes an end-start relation, such a dependency imposes a time-wise precedence relation between i and j, meaning that j can only start once i has completed. A time conflict then means that the start time of either i or j violates the precedence $i \prec j$. Such a conflict might arise due to various reasons. Due to the dynamic and unpredictable nature of the manufacturing environment, a common reason is the delay of a task relative to its prior duration that was used to generate the plan. When task i is delayed, the planned start time of successor task j may be earlier than the completion time of i, leading to a violation of the precedence $i \prec j$.

A resource conflict is a situation where at a certain time, more resources than available are needed to execute all tasks that are in progress, leading to a capacity problem so that some of the tasks have to be postponed or extra capacity has to be attracted. Conflicts like these are common in manufacturing planning. Manufacturing planning makes often use of traditional production control systems based on Orlicky's *Material Requirements Planning* (MRP) [35]. MRP does not consider finite-capacity planning [30] but instead, assumes an infinite resource capacity. This in turn, leads to resource-infeasible manufacturing plans for which the resource conflicts need to be manually resolved by the planner. Several attempts have been made to integrate finite capacity planning into MRP upfront. Nevertheless, traditional MRP production control systems are still widely used. Furthermore, since manufacturing planning environments are dynamic with continuous changes (in e.g. resource usage, shop floor work in progress, etc.) focusing on a procedure that deals with resolving conflicts in manufacturing plans afterwards is still of significant importance.

A practical case in which conflicts may arise is the case where the planner wants to examine what-if scenarios. The planner might be interested in the effect of adding a new customer order to the resulting manufacturing plan. The planner often wants to know this roughly and quickly. In practice, this means that he or she adds the order to the network without considering all constraints, which may lead to time and resource conflicts as a consequence. The planner can only get a reliable outcome of such a what-if scenario if these conflicts are resolved. This does not only serve as a motivation for resolving conflicts, it also serves as motivation for a fast solution procedure to do so. We further elaborate on this in Sect. 1.1.4.

1.1.3 Aggregated Planning

The concept of *aggregation* plays an important role in defining a model for the MPP. In this thesis, we make a distinction between *detailed planning* and *aggregated planning*. The latter uses a specific type of aggregation that defines so-called *aggregated resource feasibility*. We first provide a motivation for this kind of aggregation, after which we provide related definitions.

As Wortmann et al. [44] pointed out, the manufacturing environment is a dynamic environment that is prone to uncertainty. From a time-feasibility perspective, the supply network may change due to a change of the demand, e.g. a customer order is added or changed. This uncertainty is especially true for resource availability. Machine resources can break down, human resources can be unavailable due to unpredictable illness, etc. This also affects the uncertainty regarding the realisation of the planned start and completion times of the tasks. When a resource is suddenly unavailable, this might result in the delay of a task that requires that resource. Tasks can also be delayed due to other reasons. A common reason is that tasks are planned assuming a specific duration, which turns out to be too short in practice, resulting in a delay. Consequently, successor tasks may be delayed as well. The further we move into the future, the higher the uncertainty regarding resource availability and the realisation of the remaining planning.

In order to distinguish planning methods that use a different level of aggregation, we first need define *detailed resource constraints* that are used in *detailed planning*.

Definition 2. *Detailed resource constraints state that at any time, the total resource requirement is not more than the total resource availability.*

Definition 3. *Detailed planning refers to constructing a manufacturing plan that satisfies the time constraints imposed by the interdependencies of the supply network and the detailed resource constraints.*

When a human planner uses detailed planning, he or she assumes to know when exactly a task will start and end. Furthermore, the planner assumes to know the resource availability at any time with full certainty. Detailed planning may therefore lead to a false feeling of certainty and exactness. When designing a manufacturing plan, human planners often deal with this uncertainty by using detailed planning constraints in the near future and more relaxed planning constraints as time progresses. For the far future, a human planner is often only interested whether a manufacturing plan roughly satisfies time and resource constraints. It is assumed that potential small violations of these constraints can be resolved later on the shop-floor when the plan rolls into the near future.

Relaxing these constraints is done based on the following modelling idea; resource and plan realisation uncertainty can be modelled by not assigning resource requirement and availability to the exact time periods at which a task is scheduled, but rather to a *time interval* that covers multiple time periods. From the perspective of a task, this models that it is not known exactly when a task is executed, only that it will be executed in a certain period. The same then holds for the corresponding resource requirement. From the perspective of resource availability, this models that it is not exactly known for how much a resource is available at each time, only how much a resource is in total available in a time period.

This idea can be captured by relaxing the resource constraints.

Definition 4. *Aggregated resource constraints refer to the resource constraints that require that the total resource requirement in a time interval is not more than the total resource availability in that interval.*

This is a kind of aggregation as the resource constraints at any time point are combined to a single

resource constraint in a time interval. In Sect. 3.2 we provide mathematical details on how we relax the resource constraints.

Definition 5. *Aggregated resource feasibility refers to the satisfaction of aggregated resource constraints.*

The time constraints are not relaxed, as these are needed to calculate time-related KPI's such as the makespan of a project. Based on this definition, we can define *aggregated planning*.

Definition 6. *Aggregated planning refers to producing a manufacturing plan that satisfies both the time constraints imposed by the interdependencies of the supply network and the aggregated resource constraints.*

1.1.4 The Necessity for a Fast and Online Solution Procedure

A solution procedure for resolving conflicts has two main characteristics, it should be *fast* and it should be possible to apply it in an *online* setting.

The problem of resolving conflicts in manufacturing plans often requires fast solutions. As was already mentioned in Sect. 1.1.3, when a planner performs a what-if scenario, he or she wants to quickly see the effects for the current manufacturing plan. Furthermore, due to the fact that a manufacturing environment is a dynamic environment, conflicts might frequently arise that need to be resolved in real-time to prevent blockage or other inefficiency problems.

Another important characteristic of the problem is what we refer to as the online characteristic. With this we mean that in practical cases, it should be possible to apply conflict resolution to a manufacturing plan at any given time and to specific parts only. Conflicts may arise during the execution of the plan, so that a real-time revision may be needed. As a consequence, this means that it should be possible to apply conflict resolution to a specific part of the plan so that the rest of the plan is guaranteed to remain unchanged. To see why, consider the case of a plan for which some tasks are already in progress. In case there is a conflict in such a plan, the start and end times of the tasks in progress cannot be re-planned, so that this part of the plan should remain unchanged. In this case we therefore only want to apply conflict resolution to the part of the plan that lies in the future. Apart from the above, there might be other reasons for changing part of the plan. As an example, a planner might want to apply conflict resolution to a specific, troublesome, part of the plan only.

1.2 Proposed Solution and Contribution

In this section, we propose a solution to the problem of resolving conflicts in manufacturing plans, a special case of the MPP. The contribution of this thesis will be two-fold.

Aggregated planning model for the MPP

In Sect. 3.1.2, we show that the MPP can be reduced to the classical Resource Constrained Project Scheduling Problem (RCPSP). In order to support aggregated manufacturing planning, we propose a relaxation of the RCPSP that allows a time-dependent level of aggregation. The level of aggregation can be fully configured depending on practical needs.

Conflict resolution algorithm

Given the size - defined by the number of tasks and pairwise interdependencies - of practical supply networks and the fact that the MPP can be reduced to the $\mathcal{NP}$-hard RCPSP, the problem of resolving conflicts in manufacturing plans leads to a complex optimization problem, making it a non-trivial and time-consuming process. Given the online environment in which conflicts have to be resolved in a short amount of time, manually resolving conflicts often leads to sub-optimal solutions.

It is for this reason that we develop an algorithm that is capable of resolving time and resource conflicts automatically, as a support to the human planner in his or her daily work. We specifically focus on resolving conflicts such that the duration of the manufacturing plan is minimized, which is equivalent to the well-known makespan minimisation objective in project scheduling [34]. For large MPP instances, it can be expected that such an algorithm is generally capable of finding better solutions in terms of optimality than a human planner. The algorithm that we develop has the following main properties:

1. It is able to work with an aggregated planning model: detailed planning in the near future and aggregated planning as time progresses. The level of aggregation can be fully configured depending on practical needs.
2. It produces near-optimal solutions in terms of makespan.
3. It is able to resolve conflicts in a short amount of time, that is, in the order of seconds, allowing it to be used in a real-time setting.
4. It can be applied to an existing manufacturing plan at any time. Moreover, it can be applied to parts of a manufacturing plan so that other parts are guaranteed to remain unchanged, allowing it to be applied in an online setting.

1.3 Thesis Outline

In this chapter we introduced the MPP and the problem of resolving conflicts in manufacturing plans. Moreover, we introduced the need for an algorithm that is able to resolve such conflicts. Before describing the algorithm, we introduce important related work.

In Sect. 2.1, we introduce the Resource Constrained Project Scheduling Problem (RCPSP), that we will use as a model for the MPP. In Sect. 2.2 we proceed with mentioning solution approaches for solving this problem. Two approaches, disjunctive arc based methods and priority rule based heuristics, contain elements that were used in the development of the conflict resolution algorithm, which we describe in more detail.

We then use Sect. 3.1 to give a formal description of a model for the MPP, after which we briefly show that this model can be reduced to the well-known RCPSP. In Sect. 3.2, we introduce a relaxation of the RCPSP that allows a time-dependent level of aggregation, ranging from detailed planning to aggregated planning. We refer to this relaxation as the Window Aggregated Resource Constrained Project Scheduling Problem (WARCPSP). This relaxation supports aggregated planning and will be used as an abstract model for the MPP throughout the rest of this thesis, in particular for the

conflict resolution algorithm. Eventually, we use Sect. 3.3 to describe the conflict resolution algorithm (CRA). We describe each of its components in full detail, including pseudo-code. Throughout this entire section, we provide a running example. In Sect. 3.4, we prove that the CRA is well-defined, meaning that it returns a conflictless schedule in a finite number of steps. Moreover, we show that it generates active schedules when being applied to RCPSP instances. Finally, we use Sect. 3.5 to introduce a post-processing technique called Forward-Backward Improvement (FBI). Experimental analysis showed that by adapting this technique to the MPP and the WARCPSP problems, the resulting algorithm has a better performance, both in terms of makespan and run-time, compared to the CRA.

We use Sect. 4.1 to show the performance of the CRA, both in terms of makespan and run-time. For all of these experiments, we use data from the PSPLIB library. We compare it to the performance of the FBI algorithm. Furthermore, we conduct an experiment that motivates the use of a specific component of the CRA and an experiment that shows the influence of instance measure parameters that were used to generate the PSPLIB instances. We conclude this thesis in Sect. 4.2 by summarising the results and proposing recommendations for future work.

Part 2

Related Work

2.1 The Resource Constrained Project Scheduling Problem

When using detailed planning, the MPP problem introduced in Sect. 1.1.1 can be reduced to a subclass of scheduling problems referred to as the *Resource Constrained Project Scheduling Problem* (RCPSP). This is further clarified in Sect. 3.1.2. The standard RCPSP consists of scheduling activities that are subject to both precedence and resource constraints so that the makespan is minimized. The RCPSP is a well-researched problem. In order to get a better understanding of the problem, we refer to Artigues et al. [1]. Furthermore, Hartmann et al. [15] provide an excellent and up-to-date survey on variants to and extensions of the RCPSP. Since the RCPSP is a well-researched problem and can be used as a reduction of the MPP, we will first introduce its basic concepts and definitions. Its notation will be further used in the rest of this paper.

2.1.1 Concepts and Definitions

An instance I of the RCPSP is given by the tuple $I = (A, R, P, d, u, c)$. Here, A denotes the set of activities $\{1, 2, \ldots, n\}$. Preemption is not allowed, which means that once started, an activity may not be interrupted. Parameter R denotes the set of renewable resources $\{1, 2, \ldots, m\}$. Those resources are called *renewable* since their full capacity is available in every period and is independent of its usage in the past. Set P contains the end-start precedence relations of the form $i \prec j$, which means that activity i should end before activity j starts. The resulting project network can be visualised by means of an *Activity-On-Node* (AON) graph, which is the directed graph $G = (A, P)$, see Fig. 2.1 for an example.

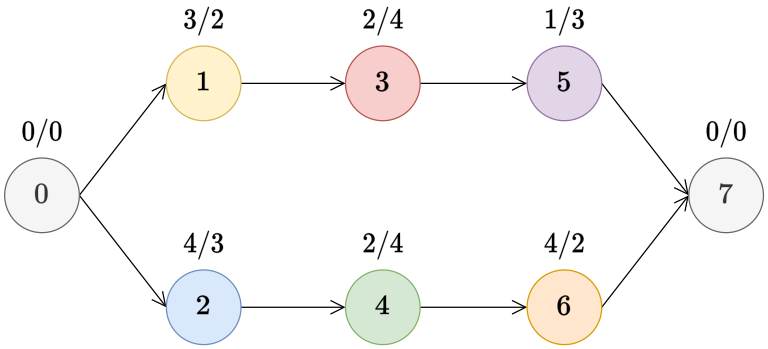


Figure 2.1: *RCPSP instance that was taken from Kolisch et al.[24], represented by an AON graph. In this example only one resource with capacity 4 is available. For each node i, the first attribute denotes the duration d_i and the second attribute the resource usage $u_{i,1}$ of activity i.*

A necessary condition for feasibility is that this graph is acyclic. The function $d : A \to \mathbb{N}$ maps a non-zero duration d_i to each activity i. The function $u : (A \times R) \to \mathbb{N}$ maps for each activity i its resource usage for each resource type k. For activity i and resource k, $u_{i,k}$ means that activity i requires $u_{i,k}$ units of resource k for the entire time it is in progress. Each resource comes with a positive capacity that is given by the function $c : R \to \mathbb{N}^+$. For resource k, a capacity value of c_k means that c_k units of resource k are available at any time. In order for an instance to be feasible, it is assumed that $u_{i,k} \leq c_k \; \forall i \in A, \; \forall k \in R$.

For convenience, many authors introduce two dummy activities, a start activity 0 that precedes all other activities and a sink activity $n+1$ that succeeds all other activities, i.e. $A = \{0, 1, \ldots, n, n+1\}$. These activities have zero duration and zero resource requirement. In this way, the start and end time (makespan) of the project are given by the start time of activity 0 and the start time of activity $n+1$, respectively. All of the mentioned parameters are assumed to be deterministic and known in advance.

A solution to an instance of the RCPSP is a vector $S = \{S_0, S_1, \ldots, S_{n+1}\} \in \mathbb{N}^A$ that contains a start time S_i for each activity $i \in A$. An example of a solution to the instance from Fig. 2.1 is given in Fig. 2.2.

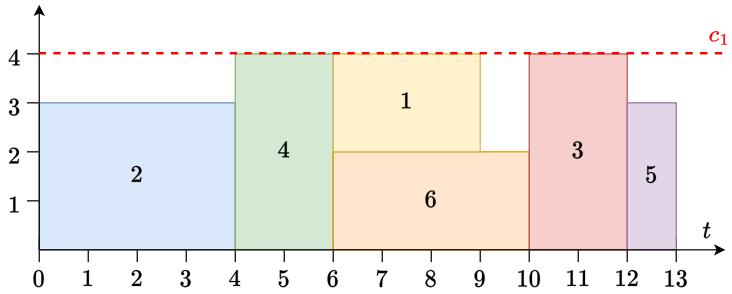


Figure 2.2: *A Gantt chart representation of an optimal solution to the RCPSP instance from Fig. 2.1. With time on the horizontal and resource usage on the vertical axis, each block i shows when activity i is scheduled and how much of a resource it uses. Note that all precedence constraints are satisfied and the total resource usage never exceeds the capacity 4, making this a feasible solution with makespan 13.*

In this thesis, we work with a time discretisation of the time horizon. This means that we fix a time horizon $\mathcal{T}$ that is right bounded by T, meaning that all activities should be completed by time T. T serves as an upper bound on the project duration and can be set to the sum of completion times of all activities, i.e. $T = \sum_{i \in A} d_i$. The time from 0 tot T is then divided into elementary periods with unit length, i.e. $\mathcal{T} = \{0, 1, \ldots, T\}$. We assume that the event times, i.e. start times and completion times of activities, happen at these discrete time points. When an activity is in progress at time t, we mean that it is in progress during the interval $[t, t+1)$.

In order for the solution to be feasible, the schedule should be *time-feasible* as well as *resource-feasible*. Time-feasibility means that the all precedence constraints are satisfied, i.e.

$$S_i + d_i \leq S_j \quad \forall\, (i \prec j) \in P. \tag{2.1}$$

Resource-feasibility means that at any time $t \in \mathcal{T}$, the total resource usage of any resource does not exceed its capacity. When in schedule S, the activities in progress at time t are given by

$$\mathcal{A}(S, t) := \{i \mid S_i \leq t < S_i + d_i\},$$

resource-feasibility can then be written via the constraints

$$\sum_{i \in \mathcal{A}(S,t)} u_{i,k} \leq c_k \quad \forall t \in \mathcal{T},\ \forall k \in R. \tag{2.2}$$

The standard objective of the RCPSP is to minimize the *makespan* $C_{max} = S_{n+1}$, i.e. the completion time of the last scheduled activity. The standard RCPSP can then be defined as follows.

Definition 7. *Given an instance $I = (A, R, P, d, u, c)$, the RCPSP can be defined as finding a solution vector $S \in \mathbb{N}^A$ so that $C_{max} = S_{n+1}$ is minimised while satisfying the time constraints from Eq. (2.1) and resource constraints from Eq. (2.2).*

2.2 Solution Approaches for the RCPSP

Over the years, different approaches to solving the RCPSP have be developed. Those approaches can be broadly categorised into *exact* or *optimal* and *heuristic* methods. In this section, we will mention some of the main results in this field. Moreover, we provide details on those literature results that we used for the development of our algorithm.

2.2.1 Exact Solution Approaches

An exact or optimal approach is a solution approach that guarantees to find the optimal solution. It achieves this by implicitly searching the entire solution space. Exact approaches often involve the use of *mathematical programming* such as *Mixed-Integer-Programming* (MIP) [36, 32, 2] and *implicit enumeration* techniques such as branch-and-bound [16, 4, 10, 5].

An example of a discrete-time MIP formulation that is suitable to model the RCPSP problem from Sect. 2.1.1 is the model from Pritsker et al. [36]. Let $x_{i,t} \in \{0,1\}$ be a binary decision variable that equals 1 if and only if activity $i \in A$ starts at time $t \in \mathcal{T}$. To abuse notation, it is assumed that $x_{i,t} = 0 \; \forall i \in A, t \leq 0$. The model is then as follows:

$$\text{minimize} \sum_{t \in \mathcal{T}} t \cdot x_{n+1,t} \tag{2.3}$$

$$\text{subject to} \sum_{t \in \mathcal{T}} x_{i,t} = 1, \qquad \forall i \in A \tag{2.4}$$

$$\sum_{i \in A} \sum_{t'=t-d_i+1}^{t} u_{i,k} \cdot x_{i,t'} \leq c_k, \qquad \forall t \in \mathcal{T}, \forall k \in R \tag{2.5}$$

$$\sum_{t \in \mathcal{T}} t \cdot x_{j,t} - \sum_{t \in \mathcal{T}} t \cdot x_{i,t} \geq d_i, \qquad \forall i \prec j \in P \tag{2.6}$$

$$x_{i,t} \in \{0,1\}, \qquad \forall i \in A, t \in \mathcal{T}$$

The objective Eq. (2.3) is to minimize the makespan that is attained at the starting time of dummy activity $n+1$. Eq. (2.4) models that each activity starts only once. Inequalities Eq. (2.5) enforce resource-feasibility, whereas the inequalities Eq. (2.6) enforce time-feasibility.

It has been shown by Blazewicz et al.[7] that the RCPSP as an optimization problem belongs to the class of (strongly) $\mathcal{NP}$-hard problems. Even more, it is among the most intractable combinatorial optimization problems. An important question is the existence of a polynomial-time approximation scheme. As pointed out by Uetz [41], the colouring problem in undirected graphs is polynomially equivalent to a special case of the standard RCPSP with makespan objective. Based on results from Feige and Kilian [13] on the graph colouring problem, Uetz [41] shows that for the RCPSP it is not possible to approximate the minimal makespan within a factor of $n^{1-\epsilon}$ in polynomial time, for any constant $\epsilon > 0$, unless $\mathcal{NP} \subseteq \mathcal{ZPP}$[1]. For practical instances of RCPSP, the number of calculations and as a result, the amount of time needed to get to the optimal solution is

[1]Complexity class $\mathcal{ZPP}$ contains all decision problems having a polynomial time randomised algorithm with zero probability of error, see [41] for more details.

therefore impractical, see e.g. Brucker et al. [9]. It is for this reason that many researchers focus on heuristic solution approaches.

2.2.2 Heuristic Solution Approaches

As is mentioned in Sect. 2.2.1, heuristic solution procedures are indispensable when solving larger instances of the RCPSP. According to Kolisch and Hartmann [24], these heuristic approaches can be classified into 1) priority rule based heuristics, 2) meta-heuristic approaches such as Simmulated Annealing, Tabu Search and Genetic Algorithms (see Table 6 of Kolisch and Hartmann [24] for an overview) and 3) other methods such as truncated branch and bound methods [38, 14] and disjunctive arc based methods [37, 33, 6].

Disjunctive arc based methods inspired our algorithm in how we resolve resource conflicts by adding precedence relations between activities, see Sect. 3.3.6. It is for this reason that we provide more details on the underlying ideas of these methods. Its underlying theory is strongly related to order based representation results for the RCPSP. We therefore first introduce these main results.

Order based representations of schedules for the RCPSP

The aim of this section is to mention main results for order based representations of the RCPSP that were introduced by Bartusch et al. [5]. It uses the notion of a *strict order*, which is defined as follows.

Definition 8. *A strict order $\prec$ on a set A is a homogeneous relation on A that is irreflexive, asymmetric and transitive. This means that for $i, j, k \in A$, it satisfies:*

1. *Irreflexivity: $(i \prec i) \notin A$.*
2. *Asymmetry: if $(i \prec j) \in A$ then $(j \prec i) \notin A$.*
3. *Transitivity: if $(i \prec j) \in A$ and $(j \prec k) \in A$ then $(i \prec k) \in A$.*

In the case of the RCPSP, a strict order $\prec$ on a set of activities A represents an end-start precedence relation between activities from A.

Let $\mathcal{S}$ denote the set of all time- and resource-feasible schedules to an instance of RCPSP. Furthermore, let O denote a strict order on A and $\mathcal{S}(P \cup O) = \{S \mid S_j - S_i \geq d_i \; \forall (i \prec j) \in P \cup O\}$ denote the polyhedron containing all schedules that satisfy the precedences from both P and O. Bartusch et al. [5] showed that $\mathcal{S}$ is the union of the polyhedra $\mathcal{S}(P \cup O)$ for all inclusion-minimal feasible strict orders O on A.

Definition 9. *Given an instance $I = (A, R, P, d, u, c)$ of the RCPSP, a strict order O is said to be feasible whenever $\mathcal{S}(P \cup O) \subseteq \mathcal{S}$. This means that all schedules that satisfy the precedences from P and O should be feasible.*

Checking whether a strict order is feasible is related to the concept of forbidden sets, which is explained in the section below.

Definition 10. *Given an instance $I = (A, R, P, d, u, c)$ of the RCPSP, a strict order O is said to be inclusion-minimal when removing any precedence from O makes O infeasible.*

Specifically, Bartusch et al. [5] showed that in case of the minimisation of a regular objective function (such as makespan), the minimal element of $\mathcal{S}(P \cup O)$ dominates the other elements in the set and this minimal elements is exactly the earliest start schedule $ES(P \cup O)$ for which the following holds:

$$ES(P \cup O) \leq S' \; \forall S' \in \mathcal{S}(P \cup O),$$

where the $\leq$ denotes an element-wise relation. Consequently, this allows us to come up with a different definition of the RCPSP, a result that was established in the Main Representation Theorem by Bartusch et al. [5].

Definition 11. *The RCPSP can be seen as the problem of finding a feasible strict order O as an extension to P such that the corresponding earliest start schedule $ES(P \cup O)$ is of minimal makespan.*

We refer to Bartusch et al. [5] and Artigues [1] for more details.

Intuitively, this new definition makes sense, as for an optimal solution S^* to RCPSP with a regular objective function, the start time of any activity j is either equal to the start of the time horizon or coincides with a completion time of another activity i. This means that if we define $O = \{(i \prec j) \mid S^*_j \geq S^*_i + d_i, (i \prec j) \notin P\}$, the earliest start schedule $ES(P \cup O)$ will be equal to S^*. Note that by defining O in this way, it is clearly feasible but it is not necessarily inclusion-minimal so that precedences may be removed while maintaining the same conclusion. As an example, the optimal schedule from Fig. 2.2 can be obtained by computing an early start schedule using the precedences from P together with the extension $O = \{(4 \prec 1), (6 \prec 3)\}$.

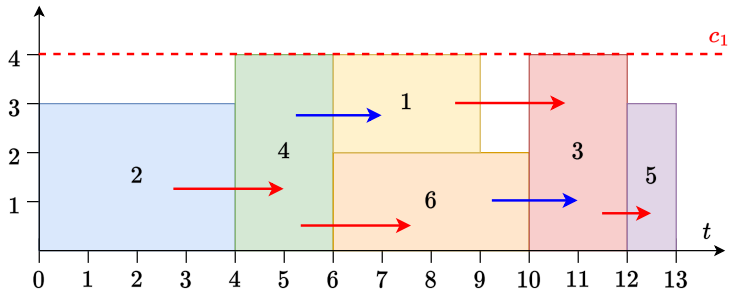


Figure 2.3: *The optimal solution S^* from Fig. 2.2 corresponding to the RCPSP instance from Fig. 2.1 together with the precedences from P (in red) and the feasible inclusion-minimal strict order O (in blue). Feasibility of O can be verified by noticing that there is a path from i to j or j to i in $G(A, P \cup O)$ whenever activity i and j do not run in parallel. It can be easily verified that $S^* = ES(P \cup O)$.*

Disjunctive Arc Based Methods and Minimal Forbidden Sets

As a generalization of the disjunctive graph model for machine- and shop problems [3], disjunctive arc based methods rely on the concept of *minimal forbidden sets*.

Definition 12. *A forbidden set $F \subseteq A$ is a set of activities that cannot be feasibly scheduled in parallel as the total resource requirement would violate the total resource availability of some resource $k \in R$, i.e. $\sum_{i \in F} u_{i,k} > c_k$. A forbidden set is minimal if no proper subset is forbidden, that is for any of its subsets $F' \subset F$ we have $\sum_{i \in F'} u_{i,k} \leq c_k \ \forall k \in R$.*

Let $G = (A, P)$ denote the AON graph corresponding to an instance I of the RCPSP. Disjunctive arc based methods are based on extending the set of arcs P by adding additional arcs (the disjunctive arcs) such that all minimal forbidden sets are 'destroyed' and the corresponding earliest start schedule is both time and resource feasible. This yields a different formulation of the RCPSP, as pointed out by Artigues [1].

Definition 13. *The RCPSP can be seen as the problem of finding a strict order O such that $G(A, P \cup O)$ is acyclic, all minimal forbidden sets are destroyed and the earliest start schedule $ES(P \cup O)$ has minimal makespan.*

Destroying a minimal forbidden set F that constitutes a resource conflict for resource $k \in R$ means that for $i, j \in F$, we add an end-start precedence relation $i \prec j$ so that j has to wait until i has finished. Since F is minimal, the activities from F do not constitute a resource conflict for $k \in R$ anymore. A disadvantage of this approach is that it requires the nontrivial task of enumerating all minimal forbidden sets. As Stork and Uetz [40] pointed out, there does not exist a polynomial time algorithm that generates all minimal forbidden sets, unless $\mathcal{P} = \mathcal{NP}$.

Priority Rule Based Heuristics

The algorithm that we develop is related to priority rule based heuristics as it will make use of priority rules in local decisions to eliminate resource conflicts. It is for this reason that we use this section to mention the underlying ideas of these heuristics in more detail.

A priority rule based heuristic consists of two components, a *Schedule Generation Scheme* (SGS) and a *priority rule*. In 1963, Kelley [18] introduced a SGS which was later revisited by Kolisch [23]. An SGS is a constructive heuristic that builds a feasible schedule by iteratively extending a partial schedule, using priority rules to select unscheduled activities to be scheduled at each iteration. Two types of SGSs exist, the *Serial Schedule Generation Scheme* (SSGS) and *Parallel Schedule Generation Scheme* (PSGS). For the first, incrementation is done w.r.t. the activities, whereas for the latter, incrementation is done w.r.t. time, see e.g. Kolisch [23] for more details on SGS.

Besides the SGS component, the other main component consists of the priority rules. In the SGS method, a priority rule π is a mapping that maps a value $\pi(i)$ to each activity i in the decision set used to determine which activity is scheduled in each iteration, after which the activities with highest/lowest value are selected. The most important question is what priority rules lead to the best schedules in terms of makespan minimisation. Much research has been done for designing good priority rules and testing their performance. Kolisch [20] analysed four different studies regarding classical priority rules: Davis & Patterson [11], Alvarez-Valdes & Tamarit [33], Boctor [8] and Valls et al.[42]. In Table 2.1 we show some of the priority rules form literature, including the studies mentioned above.

Table 2.1: *Well-known priority rules from literature.*

Rule	**Name**	$\pi(i)$	**Ref.**
SDT	Shortest Duration Time	d_i	[33]
MTS	Most Total Successors	$\lvert\hat{\mathcal{S}}_i\rvert$	[33]
MLST	Minimum Latest Start Time	LS_i	
MSLK	Minimum Slack	$LS_i - ES_i$	[11, 8, 42]
GRD	Greatest Resource Demand	$d_i \cdot \sum_{r \in R} u_{i,k}$	[11]
MLFT	Minimum Latest Finish Time	LF_i	[11, 8, 33]
GRPW	Greatest Resource Positional Weight	$p_i + \sum_{j \in \mathcal{S}_i} p_j$	[33, 42]

The earliest start ES_i, latest start LS_i and latest finish LF_i values of activity i that are exploited by rules from Table 2.1 are computed using a critical path method [19]. The MTS rule employs $|\hat{\mathcal{S}}_i|$, the number of direct and indirect successors of activity i. The MSLK rule looks at the *total float* of an activity, being the time that an activity may be delayed from its time-feasible earliest start time without increasing the makespan or violating a precedence constraint. The GRPW rule employs the processing time of activity i together with the sum of processing times of all direct successors of i, denoted by $\mathcal{S}_i$.

The studies by Davis & Patterson [11], Boctor [8] and Valls et al. [42] show that the MSLK rule generally leads to the best performance among the priority rules that they tested. Alvarez-valdes & Tamarit [33] conclude in their studies that the GRPW rules performs better than the MLFT and MTS rule. The latter is also concluded by Valls et al. [42]. However, when using a PSGS, Alvarez-valdes & Tamarit [33] conclude that the MSLK and MLFT rule perform equally well.

Part 3

Model & Algorithm

3.1 The MPP and its Relation to the RCPSP

In Sect. 1.1.1 we introduced the problem of resolving conflicts in manufacturing plans as a special case of the Manufacturing Planning Problem (MPP). The main elements of the MPP are the supply network and manufacturing plan. These are general terms that need to be defined precisely. In this section, we give a more precise description of these elements and mention corresponding assumptions that are important for the remainder of this thesis. This is covered in Sect. 3.1.1. We proceed with Sect. 3.1.2 in which we show that MPP can be reduced to the RCPSP.

3.1.1 Supply Networks and Manufacturing Plans

In manufacturing industries, a supply network typically consists of raw, semi-finished and finished physical products that are linked together by tasks or operations. Each such a task adds value to the product by further transforming it towards the final product that is required by an external demand such as a sales order or stock refill. These operations are interrelated via time and demand dependencies that are imposed by the Bill of Material or Routing sequence.

In this thesis, we focus on supply networks that can be represented by a directed precedence graph $G(A, P)$ where A represents the set of tasks and P represents the pair-wise precedences between the tasks. Approaches to modelling supply networks as such can be found in Chapter 6 of Kolisch [22], Li and Amini [28] and Li and Womer [29]. We assume that a supply network corresponds to a single external demand. In that way, the supply network and manufacturing plan can be seen as corresponding to a single project. In this way, minimising the makespan, i.e. the completion time of the last finishing task in the plan, makes sense from a company and customer point of view. In case the network would represent multiple orders, we would need to minimise completion times of different tasks representing the deliveries of different final products.

Each node $i \in A$ in the supply network represents a task. A task $i \in A$ is a generic concept that can represent an operation or (set of) material component(s) or both. Each task i has two attributes d_i and $u_{i,k}$. Attribute d_i represents the duration of the task, which can be positive in case of a time-consuming operation or 0 in case i represents a material component. Furthermore, a task

may require resources such as personnel, machines and tools in order for the task to be executed. Set R denotes the resource types, each resource type k having an availability of c_k units. For resource type k, this resource usage is denoted by $u_{i,k}$ and we assume that this resource requirement is for the entire duration of the task. Each task may require 0 (e.g. in case the task represents a component) or multiple resource types k and multiple units of a specific resource (e.g. 2 welders). It is assumed that a unit of a specific resource can only be used by one task at a time. In case a task represents a material component, the task may have zero resource requirement.

Just as in Li and Womer [29], we assume that the Bill of Material and Routing sequences for a supply network are known and fixed so that P is known and fixed as well. The set P represents time and demand dependencies between the tasks, introduced by the Bill of Material and Routing sequences. The only difference between a time and demand dependency is that for the latter, a demand of a physical component is involved, whereas this is not necessarily true for the latter. In both cases, dependency $(i \prec j) \in P$ is an end-start precedence between task i and j, see e.g. Kolisch [21]. As an example, in case i and j represent two consecutive operations in the same Routing sequence, $i \prec j$ simply means that operation j cannot be executed before operation i is finished as j requires the added value to the component that is provided by i. If i represents a raw material that is needed to start operation j, the dependency represents a consumption and entity i should be first executed (the raw material should be present) before j can be executed (operation j can be started). In case i represents a production of x components and j requires only $y < x$ components, we assume that j has to wait before the entire production of i is finished. Altogether, this means that the supply network can be regarded as a precedence network. We assume that P can introduce both converging and diverging structures in G. A converging structure means that there are at least two activities $i_1, i_2 \in A$ so that $(i_1 \prec j), (i_2 \prec j) \in P$ for some $j \neq i_1, i_2$. A diverging structure means that there at least two activities $j_1, j_2 \in A$ so that $(i \prec j_1), (i \prec j_2) \in P$ for some $i \neq j_1, j_2$.

An example of a supply network that meets our definition is depicted in Fig. 3.1.

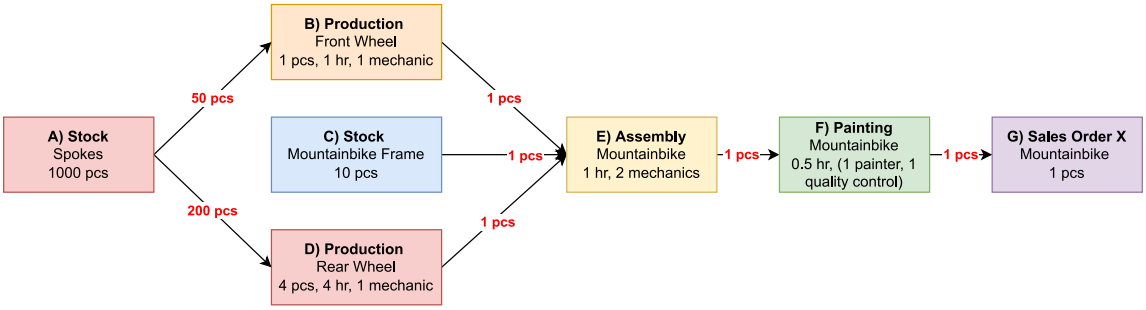


Figure 3.1: *An example of a Supply Chain network.*

The supply network of Fig. 3.1 consists of tasks that require time and produce components (e.g. B and F) as well as tasks that only represent the delivery of components (e.g. A and C). Node G represents a sales order that requires the production of a single mountainbike. The Bill of Material specifies that the mountainbike is an assembly of a frame, front and rear wheel. The latter two components require spokes. The arcs between the corresponding nodes A-E in the graph represent

the demand dependencies. The spokes and frame are available in stock, so that they do not have to be produced. As a result, node A and C have zero duration. Although task E requires only 1 rear wheel and task D produces 4, E can only start when D is completely finished. The precedence between task E and F is an example of a precedence arising from the Routing sequence, stating that the mountainbike should first be assembled before it can be painted. As for resource requirement, task B and D require 1 mechanic, E requires 2 mechanics and F requires both a painter and someone from quality control. Note that this network contains both converging and diverging structures.

Given a supply network, a corresponding manufacturing plan is a schedule S that contains a start time for each task $i \in A$ in the supply network.

3.1.2 Reduction from the MPP to the RCPSP

When detailed planning, the MPP problem and model defined in Sect. 3.1 closely resembles the RCPSP problem and model from Sect. 2.1.1. In mathematical sense, the MPP can be reduced to the RCPSP. Although most may be trivial, we show this reduction between the MPP and RCPSP in Table 3.1.

Table 3.1: *Relation between the MPP and the RCPSP.*

RCPSP	**MPP**
Activities $i \in A$	Tasks in the supply network
Resources $k \in R$	Renewable resources such as personnel, machines, tools, etc.
Precedences P	Time and demand dependencies introduced by the Bill of Material and Routing sequences
Project network $G = (A, P)$	Supply network corresponding to a single demand
Duration d_i	Duration of task i
Resource usage $u_{i,k}$	The number of units of resource k required by task i
Capacity c_k	The number of units for which resource k is available
Time horizon $\mathcal{T} = \{1, 2, \ldots, T\}$	Horizon over which the activities from the supply network are planned
Solution S	A manufacturing plan containing start times for all tasks
Activity S_{n+1}	The final component as required by the demand

By definition, it is trivial that the time and resource constraints that should be satisfied when using a detailed planning model for the MPP can be reduced to the RCPSP time and resource constraints from Eq. (2.1) and Eq. (2.2), respectively. The makespan minimisation goal of the MPP corresponds with the same goal of the RCPSP.

3.2 Window Aggregated RCPSP

In Sect. 1.1.3, we motivated the need for aggregated planning. We do so by relaxing the RCPSP resource constraints from Eq. (2.2). This will be explained in detail below.

The underlying idea in our relaxed approach is that instead of enforcing that at any time the total resource requirement should not exceed the resource availability for any resource, we enforce that for pre-defined time windows, the total resource requirement in every *time window* should not be more than the corresponding total availability in that window. Given is a win-

dow $w = [t_w^l, t_w^r),\ w \in W = \{1, 2, \ldots, \ell\}$[1]. We refer to W as the *window configuration*. Let $|w|$ denote the width of window w, i.e. $|w| = t_w^r - t_w^l$. For each window $w \in W$, the total resource requirement in w should not exceed the total availability of the resources in w. Given schedule S, let us define $d_{i,w}(S)$ to be the duration of activity i in window w, that is

$$d_{i,w}(S) = \Big|[S_i, S_i + d_i) \cap [t_w^l, t_w^r)\Big| = \max\Big\{0, \min\big\{S_i + d_i, t_w^r\big\} - \max\big\{S_i, t_w^l\big\}\Big\}.$$

We then replace the resource constraints from Eq. (2.2) with the relaxation

$$\sum_{i \in A} d_{i,w}(S) \cdot u_{i,k} \leq |w| \cdot c_k \quad \forall\ w \in W,\ \forall k \in R. \tag{3.1}$$

In the remainder of this paper, we refer to these constraints as the *aggregated resource constraints*. We refer to the resulting problem as the *Window Aggregated Resource Constrained Project Scheduling Problem* (WARCPSP).

Definition 14. *Given the instance $I = (A, R, P, W, d, u, c)$, the WARCPSP is a relaxation of the standard RCPSP. It is a relaxation in the sense that the resource constraints from Eq. (2.2) are replaced by the aggregated resource constraints from Eq. (3.1), for which the window configuration W is used.*

To see that the WARCPSP is indeed a relaxation of the RCPSP, consider Theorem 3.2.1.

Theorem 3.2.1. *The WARCPSP is a relaxation of the RCPSP.*

Proof. The proof is a slight modification of the proof by More et al. [31]. Given any schedule S that satisfies Eq. (2.2), we only need to show that S satisfies Eq. (3.1), as these are the constraints that distinguish the WARCPSP from the RCPSP. First, for $i \in A$ and for all $t \in \mathcal{T}$, let $\delta_{i,t}(S) = 1$ if $S_i \leq t < S_i + d_i$ and 0 otherwise. Then $\mathcal{A}(S, t) \equiv \{i \in A \mid \delta_{i,t}(S) = 1\}$. This means that Eq. (2.2) can be rewritten as

$$\sum_{i \in A} u_{i,k} \delta_{i,t}(S) \leq c_k \quad \forall t \in \mathcal{T},\ \forall k \in R.$$

Now, $d_{i,w}(S)$ can be rewritten as

$$d_{i,w}(S) = \sum_{t=t_w^l}^{t_w^r - 1} \delta_{i,t}(S) \quad \forall i \in A,\ \forall w \in W.$$

To see this, Fig. 3.2 shows the relation between $d_{i,w}(S)$ and $\delta_{i,t}(S)$ by means of an example.

[1]The careful reader will notice that this is a slight abuse of notation as w denotes both a window index and the window itself, defined by the time interval it spans. From now on, W will be used both as index set as well as set of windows (given by the time intervals). From the context it is clear what is meant.

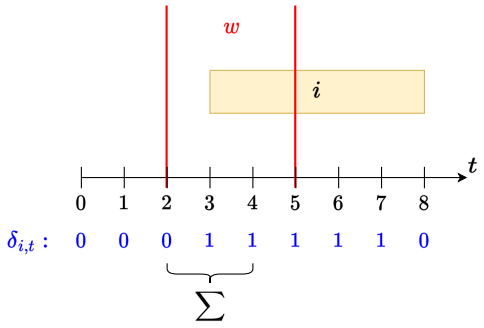


Figure 3.2: *Relationship between $d_{i,w}(S)$ and $\delta_{i,t}(S)$. Here, $d_{i,w} = 2$ because $\delta_{i,t} = 1$ for $t = 3, 4$.*

As a result, we have that

$$\sum_{i \in A} d_{i,w}(S) \cdot u_{i,k} = \sum_{t=t_w^l}^{t_w^r - 1} \sum_{i \in A} u_{i,k} \delta_{i,t}(S) \leq \sum_{t=t_w^l}^{t_w^r - 1} c_k = |w| \cdot c_k \quad \forall w \in W, \forall k \in R.$$

■

It can be easily seen that when using consecutive, non-overlapping windows of width 1, the WARCPSP reduces to the RCPSP.

A similar relaxation was introduced by Morin et al. [31], called the *Periodically Aggregated Resource Constrained Project Scheduling Problem* (PARCPSP). The PARCPSP uses a uniform subdivision of the time horizon so that windows do not overlap. Our definition of the WARCPSP allows for more general window configurations, see Sect. 3.3.4 for more details. Another difference between the WARCPSP and PARCPSP is that the first uses a discrete time horizon for the start time decision variables, whereas the latter uses a continuous time horizon.

To see the difference between the WARCPSP and RCPSP, consider the example from Fig. 3.3. This instance contains only one activity that requires 2 units of a resource with capacity 1. When using the detailed RCPSP resource constraints from Eq. (2.2), this would always lead to a resource-infeasible schedule. However, when using the aggregated WARCPSP resource constraints from (3.1) with window configuration $W = \{[0, 2), [1, 3)\}$, we would have the constraint $2 \leq 2$ twice, which is satisfied.

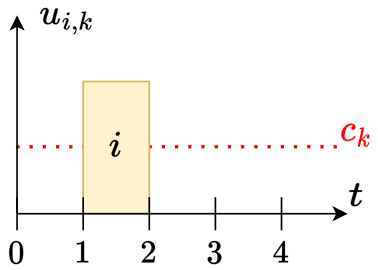


Figure 3.3: *Example of a schedule that is infeasible when using the RCPSP resource constrains from Eq. (2.2). The schedule is feasible when using the WARCPSP resource constraints from Eq. (3.1) with window configuration* $W = \{[0, 2), [1, 3)\}$.

The WARCPSP allows detailed planning in the near future and aggregated planning as time progresses, an idea that an idea that was introduced in Sect. 1.1.3. This can be achieved by increasing the window size over time. Sect. 3.3.4 provides more details on constructing different window configurations W.

3.3 Design of the CRA Algorithm

In this section, we describe the algorithm that resolves both time and resource conflicts in a given schedule corresponding to an instance of the WARCPSP. The algorithm does so by taking into account the aggregated resource constraints from (3.1). From now on, we will refer to this algorithm as the *Conflict Resolution Algorithm* (CRA). The CRA starts by resolving time conflicts which it does by transforming the schedule to a time-feasible earliest start schedule, see Sect. 3.3.3. Since the window configuration of a WARCPSP instance is used by the CRA, we explain different types of configurations in Sect. 3.3.4. The main part of the algorithm consists of resolving resource conflicts. Resolving resource conflicts is done in a time-iterative fashion and consists of three core components. At first, resource conflicts are detected in the given schedule, see Sect. 3.3.5. Consequently, such a conflict is resolved by adding a precedence between two activities so that one activity is shifted forward, see Sect. 3.3.6. Finally, the algorithm tries to shift some activities backward to heuristically improve the schedule, without violating time or aggregated resource constraints, see Sect. 3.3.7.

For each of these components, we provide pseudo-code, for which we assume zero-based numbering.

3.3.1 Outline

Given is an instance $I = (A, R, P, W, d, u, c)$ with corresponding infeasible schedule S_{in} and vector LB with lower bounds on the earliest start values of the activities. We first resolve time conflicts by transforming the schedule to a time-feasible earliest start schedule S, using the values from LB. Once the schedule is time-feasible, the CRA starts scanning for a resource conflict from window

w_c onward. In case a resource conflict has been found in window w_c', the conflict is (partially) resolved by adding a precedence $i \prec j$ which moves activity j forward in time. A new schedule S' is computed including this new precedence. For S', it is then checked whether some activities can be moved backward in time without violating time or aggregated resource constraints. The necessity of this step depends on how S' was computed in the first place. This will be explained subsequently by also giving an example. This leads to a new schedule S'', for which the above steps are repeated, starting with resource conflict detection at the new window w_c'. These iterations continue until all conflicts have been resolved, so that a conflictless schedule S_{out} is returned. This procedure is depicted in Fig. 3.4.

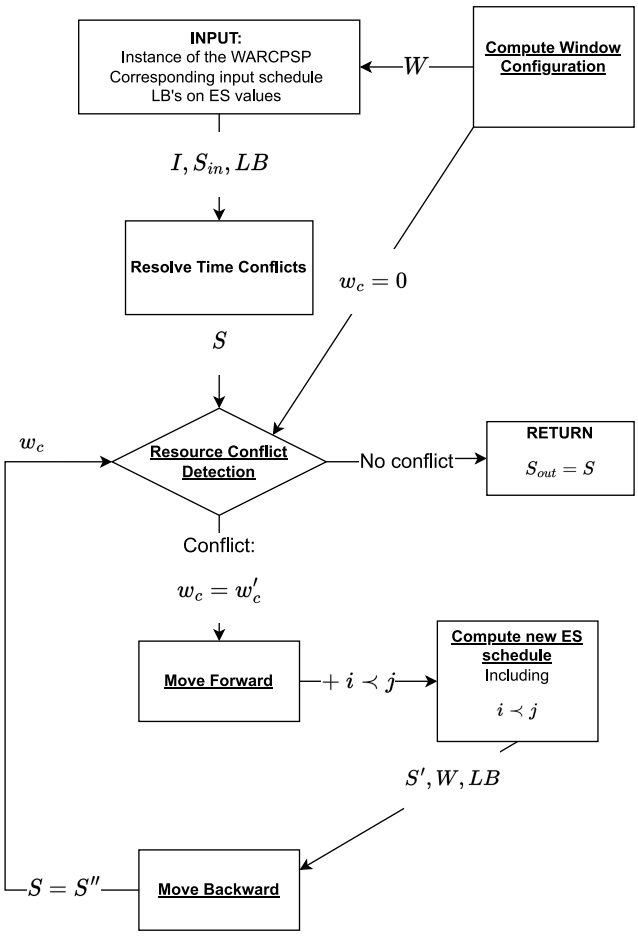


Figure 3.4: *Outline of the CRA.*

3.3.2 Running Example: Input

In order to better explain each component of the CRA, we present a running example throughout this section. The instance of the WARCPSP that we consider as input is the one that is presented in Fig. 2.1 together with the window configuration $W = \{[0, 4), [1, 5), [2, 6), \ldots\}$. The corresponding schedule that we will use as input to the CRA is visualised via a Gantt chart in Fig. 3.5.

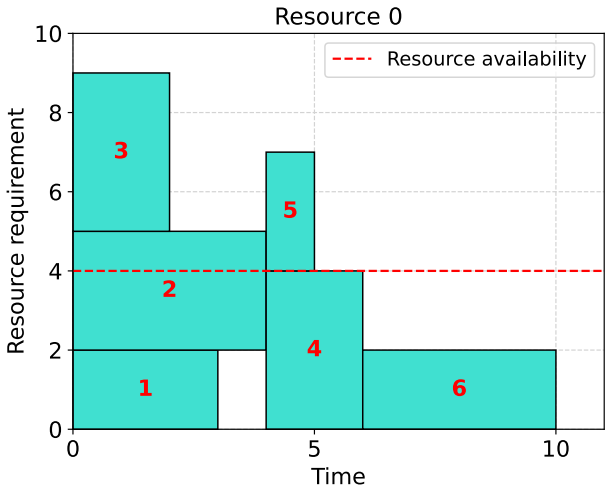


Figure 3.5: *Time and resource-infeasible schedule S_{in} corresponding to the instance from Fig. 2.1 that will serve as input to the CRA algorithm.*

3.3.3 Resolving Time Conflicts

A time-feasible schedule is a schedule that satisfies Eq. (2.1). Consider a time conflict

$$S_i + d_i > S_j$$

for some $(i \prec j) \in P$. The only way to resolve this time conflict is to either start activity i earlier or activity j later in time. For a makespan minimisation goal, starting activity i earlier in time would be a reasonable choice. We therefore transform the input schedule S_{in} to a corresponding earliest start schedule, in which each activity starts as early as possible, considering the precedences from P. In practice, however, this means that S might be totally different than S_{in}. There can be many reasons why in a given schedule, some activities were scheduled later than their time-feasible earliest start time. When the planner already considered resource constraints for part of the schedule, activities are likely scheduled later than their time-feasible earliest start time. Transforming the schedule to an earliest start time would ignore this design. Furthermore, this might even result in activities starting in the past. In order to capture this issue for later use in an online setting, see Sect. 1.1.4, we propose to work with lower bounds in computing the earliest start values. These lower bounds are stored in the vector LB. For each activity $i \in A$, we compute its earliest start time ES_i by considering both the precedences from P and that $ES_i \geq LB_i$. The latter restriction distinguishes this procedure from the standard earliest start procedure used in Critical Path Methods [19]. More formally, we define an earliest start schedule as follows:

Definition 15. *Given is a set of precedences P and a vector of lower bounds LB. The corresponding earliest start schedule $ES(P, LB)$ is a schedule S for which none of the start times S_i can be decreased without violating precedences from P or lower bounds from LB.*

Vector LB is an input to the CRA and its values can be set depending on the context of the planning problem. A planner might decide to set these lower bound values to the current start times in the input schedule, i.e. $LB = S_{in}$, so these activities can only start at the same or later in time. The earliest start values of the activities are calculated via a forward pass in a topological ordering [2] of their corresponding nodes in the precedence graph. Pseudo-code for this can be found in Alg. 1.

Algorithm 1: ResolveTimeConflicts(A, P, d, LB)

```
S_i ← LB_i ∀i ∈ A        //Set S to the lower bounds for the earliest start values
stack ← TopologicalOrdering(P)
while stack ≠ ∅ do
    i ← stack.pop(0)              //Select and delete the first item from the stack
    S_i ← {j ∈ A | (i ≺ j) ∈ P}                          //All direct successors of i
    for j ∈ S_i do
        S_j ← max{S_j, S_i + d_i, LB_j}
    end
end
return S
```

The topological ordering is computed using Kahn's algorithm [17]. The time complexity of this algorithm is $O\big(n + |P|\big)$, where $|P|$ denotes the number of precedences in P. Since the precedence graph contains $n + 2$ nodes, Alg. 1 has $n + 2$ iterations. In each iteration, a linear computation is evaluated for all successors of a selected activity i, which are $|P|$ at most. Consequently, the time complexity of Alg. 1 is $O\big(n \cdot |P|\big)$.

Running example continued

When we consider the instance from Fig. 2.1 and the corresponding input schedule from Fig. 3.5, we do see one time conflict: $1 \not\prec 3$. One should note that activities 1, 2, 4 and 6 start already as early as is time-feasibly possible. As a result, when we use the lower bounds $LB_i = 0\ \forall i \in A$, we get that activity 3 now starts when 1 completes, so that activity 5 has to start later as well. The start times of the other activities remain unchanged. This can be seen in the time-feasible schedule from Fig. 3.6.

[2] A topological ordering of a directed graph is a linear ordering of its vertices such that for every arc (i, j), i comes before j in the ordering.

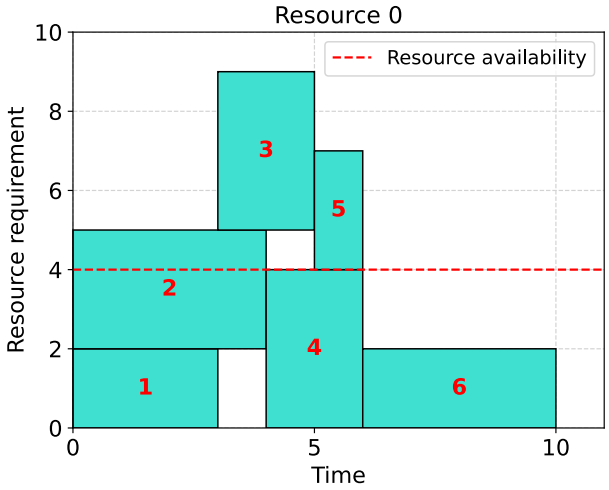


Figure 3.6: *The earliest start time-feasible schedule $ES(P, \boldsymbol{0})$ corresponding to the time-infeasible schedule from Fig. 3.5.*

In the remaining sections, we proceed with explaining how resource conflicts are detected and resolved, meanwhile maintaining a time-feasible schedule.

3.3.4 Window Configuration

Before the aggregated resource constraints from Eq. (3.1) can be evaluated, one needs to define a window configuration W. This configuration influences the effect of the aggregated resource constraints from Eq. (3.1) and therefore has an important impact on the feasibility of resulting schedules.

Different window configurations W can be used, of which we describe a few.

Sliding window configuration

Given the parameter Δ_w, we consider a window $w(t)$ starting at $t \in \{0, 1, \ldots, C_{max} - \Delta_w\}$ for which $|w(t)| = \Delta_w$ holds. Window $w(t)$ spans the period $[t, t + \Delta_w)$. Besides the window width Δ_w, another parameter that determines this configuration is the step size Δ_{sz} with which the windows slides over the time horizon. The configuration is then given by $W = \{w(0), w(0+\Delta_{sz}), w(0+2\cdot\Delta_{sz}), \ldots\}$. This configuration gives the impression that the window slides forward over the time horizon, therefore it is referred to as the *sliding window* configuration.

As an illustration, consider the example from Fig. 3.7. In this example, $\Delta_w = 4$ and $\Delta_{sz} = 1$, starting from time 0.

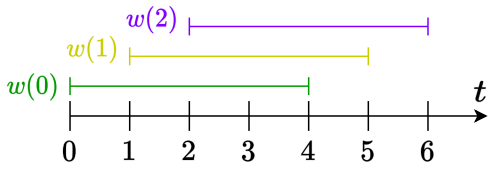


Figure 3.7: *Example of a sliding window configuration with* $\Delta_w = 4$ *and* $\Delta_{sz} = 1$.

Some special cases of this configuration are given below.

PARCPSP configuration

If we set $\Delta_w = \Delta_{sz}$ we get consecutive, non-overlapping windows of the same length, just as what Morin et al. [31] use in their PARCPSP.

RCPSP configuration

In case we use the sliding window configuration with $\Delta_w = \Delta_{sz} = 1$, we have that the aggregated resource constraints from Eq. (3.1) become the usual resource constraints from Eq. (2.2).

Configuration with a varying window size

Another special case of the sliding window configuration is the case where the window width parameter is time-dependent, i.e. $\Delta_w \equiv \Delta_{w(t)}$. This allows to increase the width of the time windows over time, leading to more relaxation of the resource constraints as time progresses, see Sect. 1.1.3.

Well-defined window configuration

We assume from now on that for for any window index w, $t^l_{w+1} \geq t^l_w$ holds for the left boundaries and $t^r_{w+1} \geq t^r_w$ holds for the right boundaries. This ensures that the CRA terminates in a finite number of steps with a conflictless schedule, see Theorem 3.4.1.

Running example continued

The window configuration of the running example from Sect. 3.3.2 can be characterised as a sliding window configuration W with a fixed window size $\Delta_w = 4$ and a step size $\Delta_{sz} = 1$, just as in Fig. 3.7.

3.3.5 Detecting Resource Conflicts

Resource conflicts are detected by comparing the total resource requirement with the total availability for each time window from the pre-defined window configuration, see Sect. 3.2. We resolve conflicts in a chronological fashion by starting with the first window w_c that contains a resource conflict. For this window w_c, we compute for each resource k the total resource requirement in w_c and verify if it does not exceed the total availability in w_c. That is, we verify whether

$$\sum_{i \in A} d_{i,w_c}(S) \cdot u_{i,k} \leq |w_c| \cdot c_k.$$

We iterate over the resources $k \in R$ in an order that is determined by the resource requirement in the window, ordered ascending. In that way, for a given window w_c, we first resolve resource conflicts for a resource $k \in R$ that has the smallest violation, after we proceed to another possible resource conflict in w_c. Experimental analysis showed that this leads to slightly better results than using the reversed or random ordering. We also compute a set of activities that are active in w_c, which are the activities i for which $d_{i,w_c}(S) > 0$. We refer to this set as the *decision set* D. Activities from D are returned in case of a resource conflict, and serve as input to the `MoveForward` method. In case of a violation in w_c for resource k_c, the algorithm returns `True` so that the `MoveForward` method is triggered. Furthermore, the conflicting resource k_c and window w_c are returned. Otherwise, if there is no resource conflict for any resource k in w_c we proceed with the next window with index $w_c + 1$ and repeat resource conflict detection for this window. Pseudocode for this algorithm can be found in Alg. 2.

Algorithm 2: DetectResourceConflicts(w_c, S, A, R, W, d, u, c)

```
conflict = FALSE
while w_c < |W| do
    left ← W[w_c][0]                          //Left and right boundaries of window w_c
    right ← W[w_c][1]
    D ← [i ∈ A|d_{i,w_c}(S) > 0]
    reqs ← [∑_{i∈D} d_{i,w_c}(S) · u_{i,k} for k ∈ R ]          //Window req. per resource
    K ← sort(R, reqs, ascending)          //Sort resources asc. based on window req.
    for k ∈ K do
        ava_total ← |w_c| · c_k
        req_total ← ∑_{i∈D} d_{i,w_c}(S) · u_{i,k}
        if req_total > ava_total then
            conflict = TRUE
            return conflict, k, w_c, D
        end
    end
    w_c ← w_c + 1
end
return conflict, None, None, None
```

Running example continued

In order to better visualise potential resource conflicts, we show the resource profile, the total resource requirement at any time, corresponding to the time-feasible input schedule S from Fig. 3.6, see Fig. 3.8.

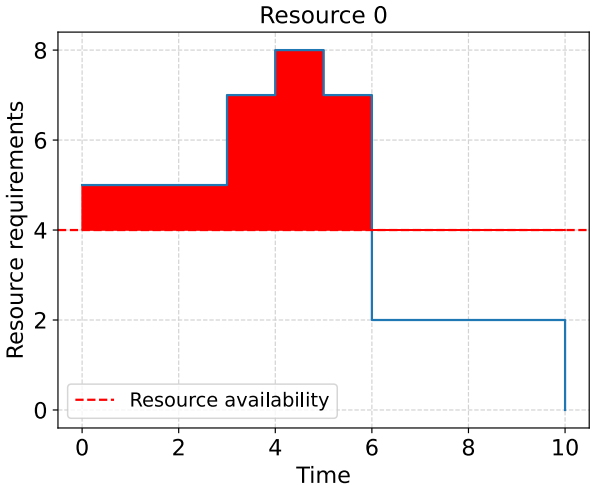


Figure 3.8: *Resource profile corresponding to the schedule S from Fig. 3.6.*

From Fig. 3.8, one can clearly see that already in the first window $w(0) = [0, 4)$, a conflict occurs as the total resource requirement equals $22 > 4 \cdot 4 = c_0$. The activities that are active in this window are $i = 1$ and $i = 2$ and $i = 3$.

3.3.6 Resolving Resource Conflicts by Forward Shifting

This section explains how a resource conflict can be resolved by forward shifting an activity in time that contributes to this conflict. When a resource conflicts occurs for resource k_c in window w_c, at least one activity requiring resource k_c has to be moved in order to decrease the resource requirement in w_c. Because the CRA schedules activities as early as possible (see Sect. 3.3.3 and the remainder of this section), resource conflicts can only be resolved by shifting such activities forward.

Motivating forward shifting by adding precedences

We shift an activity forward by extending the precedence set P with a precedence $i \prec j$, in such a way that this results in j shifting forward in time when calculating a new earliest start schedule. Adding precedences in this way eventually resolves the resource conflict, see Lemma 3.4.2.

This idea was inspired by the order based representation results from Sect. 2.2.2 and the notion of breaking minimal forbidden sets from Sect. 2.2.2. The first showed that for the RCPSP, the procedure of adding the right precedences and computing the resulting earliest start schedule should be able to produce the optimal solution. Furthermore, the activities in window w_c that contribute to the resource conflict for resource k_c can be seen as a (not necessarily minimal) forbidden set that we need to destroy by adding precedences.

Selecting candidates via priority rules

Given a window w_c in which a resource conflict occurs for resource k_c, we consider the activities from the decision set D, which was defined as the set of all activities that are active in window w_c, i.e. for which $d_{j,w_c} > 0$ holds. In case a resource conflict was detected for resource k_c, only activities j that have $u_{j,k_c} > 0$ can resolve the conflict. Therefore, we filter out activities j for which $u_{j,k_c} = 0$. Moreover, we should have that that j does not start later than the completion times of all other activities in w_c, as in this case D does not contain activities i that are eligible to be used as predecessor of j in the added precedence $i \prec j$. This will become clear in the rest of this section.

For the remaining activities, we apply a priority rule π (see Sect. 2.2.2) to select which of the activities from D is shifted forward. The underlying idea is to select the activity $j \in D$ whose forward shift is least disadvantageous in terms of makespan minimisation. For this, we use the `SelectByPriority` method that returns all activities j that have the highest priority value $\pi(j)$. In case of a tie, we take the activity with highest index value.

We experiment with different classical priority rules that are listed in Table 2.1. However, instead of using these rules exactly, we used their counterpart by reversing the priorities, meaning that when such a rule π selects an activity with the highest priority, we select the activity with the lowest priority. The reason for this is that the rules listed in Table 2.1 were designed to schedule the next activity in a SGS context, whereas we use priority rules to determine which activity to shift forward or schedule later. This means that the rules listed in Table 2.1 are adapted and renamed, as can be seen in Table 3.2.

Table 3.2: *Adapting and renaming priority rules from Table 2.1. The new rules reverse the priorities of their counterpart in Table 2.1.*

Old Rule	**New Rule**	**New Name**
SDT	**LDT**	Longest Duration Time
MTS	**LTS**	Least Total Successors
MSLK	**MSLK**	Maximum Slack
MLST	**MLST**	Maximum Latest Start Time
GRD	**SRD**	Smallest Resource Demand
MLFT	**MLFT**	Maximum Latest Finish Time
GRPW	**SRPW**	Smallest Resource Positional Weight

The earliest start, latest start and latest finish values are calculated based on the most up-to-date schedule before activity j is shifted forward. For the LTS and SRPW rule, the number of (immediate) successors has been calculated based on the precedence graph induced by P.

During the algorithm design, we tested the performance of the CRA for different priority rules on 60 instances of the PSPLIB-120 data set, each of these instances having a different parameter values for Network Complexity, Resource Factor and Resource Strength, see Sect. 4.1.1 for more details. For each of these instances, we applied the CRA and compared the resulting makespan with the best known makespan. In Table 3.3, one can find the average relative optimality gap for each of the priority rules. More about PSPLIB, parameters used to generated the instances and best known solutions can be found in Sect. 4.1.1.

Table 3.3: *Performance comparison of the CRA for different priority rules from Table 3.2 on PSPLIB-120 instances. For this experiment, a sliding window configuration with $\Delta_w = 4, \Delta_{sz} = 1$ has been used, see Sect. 3.3.4 for more details.*

Rule	Relative Gap
SDT	26.3%
GRPW	20.3%
GRD	19.7%
MTS	10.1%
MSLK	8.9%
MLFT	7.6%
MLST	6.8%

From the results of Table 3.3, we conclude that the choice of priority rules is of significant importance and that the MLST has the best average performance. It is for this reason that we use the MLST priority rule in the rest of the experiments, unless stated otherwise.

Determining the predecessor

Now activity $j \in D$ has been selected to move forward, the amount that j will be moved forward has to be determined. Instead of moving forward unit by unit, we choose a suitable activity $i \in D, i \neq j$, after we add the precedence $i \prec j$. Whether i is a suitable activity, depends on two aspects. First of all, adding the precedence $i \prec j$ should actually enforce j to shift forward in time in order for the new schedule S' to remain time-feasible. If j would not be enforced to shift forward in time, the resource conflict would remain. For this reason, activity i is only suitable if

$$S_j < S_i + d_i.$$

Secondly, the precedence $i \prec j$ can only be added when this does not lead to a cycle in the precedence graph induced by $P \cup (i \prec j)$, which is the case when there is already a path from j to i. Such a cycle can be detected by computing the transitive closure of the precedence graph, which only needs to be done once as the precedence graph induced by P does not change. For this reason, activity i is only suitable if

$$j \not\rightarrow i.$$

Considering these two constraints, we select the activity $i \in D$ for which adding the precedence $i \prec j$ leads to the *smallest* forward shift in time. The underlying idea for this is that shifting activities forward by the least amount leads to less unnecessary idle gaps in the resource profile, which later would have to be filled by the `MoveBackward` procedure, see Sect. 3.3.7. The amount that activity j will be shifted forward as a result of adding $i \prec j$ is given by $S_i + d_i - S_j$, see Fig. 3.9.

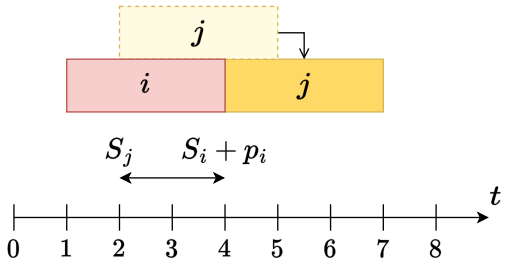


Figure 3.9: *When adding precedence $i \prec j$, activity j gets shifted forward by an amount $S_i + d_i - S_j$.*

All that said, given a set of suitable activities I, we select activity i via

$$i = \arg \min_{(i,j) \in I} \{S_i + d_i - S_j\}.$$

In case of a tie, we make an arbitrary choice and choose the activity with the highest activity index. The precedence $i \prec j$ is then returned by the method so that it will be used to compute a new schedule S', see Fig. 3.4. Pseudocode for the resulting algorithm can be found in Alg. 3.

Algorithm 3: MoveForward(D, S, d, u, k_c, π)

```
eligible ← {j ∈ D | u_{j,k_c} > 0 and S_j ≱ max_{u∈D,u≠j} S_u + d_u}
j ← SelectByPriority(eligible, S, π)
eligible ← ∅
for i∈ D do
    if S_j < S_i + d_i and j ↛ i then
        eligible ← eligible ∪ (i, j)
    end
end
i ← arg min_{i∈eligible}{S_i + d_i − S_j}
return i ≺ j
```

Updating the schedule

Once an activity pair (i, j) has been selected, the schedule needs to be updated so that $i \prec j$ is incorporated. This means that from a given schedule S, we create a new schedule S' for which the start time of activity j and possibly some of its (both immediate and non-immediate) successors in $\hat{\mathcal{S}}_j$ have to be updated in order to keep a time-feasible schedule. We do so by using an earliest start procedure, i.e. $S' = ES\big(P \cup (i \prec j), lb\big)$. Similarly as in Sect. 3.3.3, the vector lb [3] contains lower bounds on the start values of each of the activities. If we would not use these lower bounds, computing an earliest start schedule would lead to a schedule for which resource conflicts that were resolved in an earlier stage might be re-introduced as resource constraints are not taken into account. We prevent this by using appropriate lower bounds on each earliest start value ES_i,

[3] We make an explicit distinction in notation between LB from Sect. 3.3.3 and lb from Sect. 3.3.6. The first is input to the CRA and remains fixed, the latter is computed during run-time and is variable.

which means that lb_i is set to the current start time S_i of activity i. In this way, activities can only keep the same starting time or move forward in time. Activities are updated via a forward pass in a topological ordering of their corresponding nodes in the precedence graph. This leads to an algorithm that is similar to Alg. 1. Pseudocode for the algorithm that updates the schedule can be found in Alg. 4.

Algorithm 4: UpdateScheduleForward(S, $(i \prec j)$, A, P, d, lb $= S$)

$S'_u \leftarrow lb_u \ \forall u \in A$ `//Set` S' `to the lower bounds for the earliest start values`
stack $\leftarrow$ TopologicalOrdering($P \cup (i \prec j)$)
while stack $\neq \emptyset$ **do**
  u $\leftarrow$ stack.pop(0)
  $\mathcal{S}_u \leftarrow \{v \in A \mid u \prec v \in P \cup (i \prec j)\}$ `//All direct successors of` u
  for $v \in \mathcal{S}_u$ **do**
    $S'_v \leftarrow \max\{S'_v, S'_u + d_u, lb_v\}$
  end
end
return S'

In the new schedule S', it might still be the case that the resource requirement for resource k_c in window w_c is too high, or that the resource requirement for another resource k'_c in w_c is too high. In this case, the CRA will continue with moving activities forward in w_c in the next iteration.

Running example continued

Given the resource conflict in window $w(0)$ (see Fig. 3.8), in which activities $1, 2$ and 3 are active (see Fig. 3.6), we can add the precedences $1 \prec 2, 2 \prec 1, 2 \prec 3, 3 \prec 1$ and $3 \prec 2$. Given schedule S, the latest start values for activity $1, 2$ and 3 are $4, 0$ and 7, respectively. Thus, when using the MLST rule, activity 3 is decided to move forward. This leads to the only possibility of adding $2 \prec 3$. The schedule is then updated via Alg. 4 to S' by including $2 \prec 3$. This leads to the schedule from Fig. 3.10.

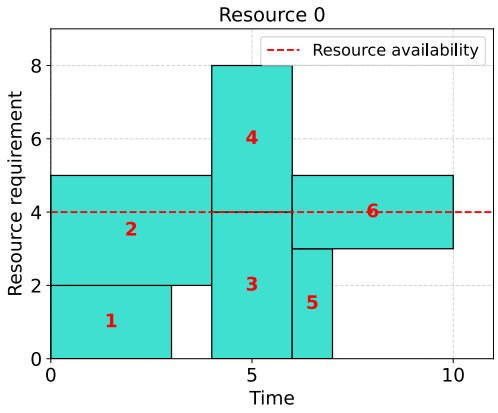


Figure 3.10: *The schedule S from Fig. 3.6 updated to $S' = ES\big(P \cup (2 \prec 3), S\big)$ by including new precedence $2 \prec 3$ and using the values from S as lower bounds on the start time values of S'.*

3.3.7 Schedule Improvement via Backward Shifting

In this section, we introduce a procedure that aims to heuristically improve the schedule by removing added precedences and backward shifting activities. This procedure is inspired by a deficiency of resolving resource conflicts in a WARCPSP related schedule via forward shifting activities by adding precedences.

Motivation

As was mentioned in Sect. 3.3.6, the order-theoretical results for the RCPSP from Sect. 2.2.2 and notion of minimal forbidden sets from Sect. 2.2.2 served as an inspiration for forward shifting activities by adding precedences. The main order-theoretical result captured by Def. 11 states that by extending the precedence set P by the right precedences and computing an earliest start schedule, the optimal solution can be obtained. This, however, turns out to not be true for the WARCPSP in general because of how we define aggregate resource feasibility. For the WARCPSP, it can be easily shown that it is not possible to represent any optimal schedule by an earliest start schedule corresponding to a set of precedences. Consider the example from Fig. 3.11.

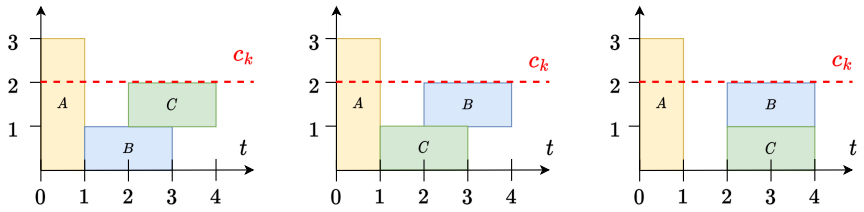


Figure 3.11: *Optimal schedules for the WARCPSP instance with* $P = \{A \prec B, A \prec C\}$ and $W = \{[0, 2), [1, 3), \ldots\}$.

The schedules from Fig. 3.11 are all optimal schedules with an optimal makespan value of 4. It can be easily seen that none of these schedules can be constructed by calculating an earliest start schedule based on a set of end-start precedences, as there is at least one activity that starts at a time that is not the completion time of any other activity.

Moreover, even when applying the CRA to an instance of RCPSP, its heuristic nature may lead to sub-optimal decisions in selecting precedences. When we do nothing about this and keep the added precedences fixed[4] throughout the conflict resolution phase, this turns out to be a too strict approach in practice. To be more precise, it will lead to idle gaps where no activity is in process, leading to an unnecessary increase of the makespan. An example of this is given in Fig. 3.12.

[4] A *fixed* precedence $i \prec j$ means that $i \prec j$ remains satisfied throughout the entire execution phase of the CRA.

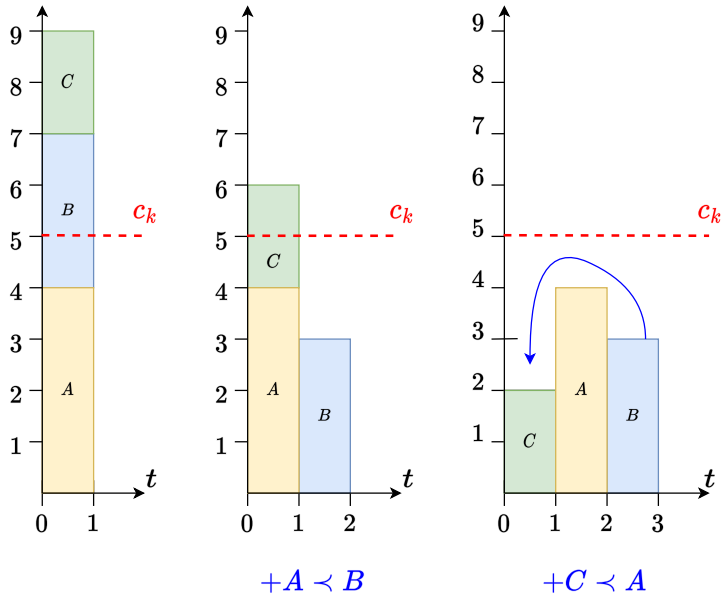


Figure 3.12: *Motivation for shifting activities backward: by removing $A \prec B$ after adding $C \prec A$, B can be shifted backward and start together with C so that the makespan can be reduced.*

For simplicity, this example uses the RCPSP window configuration, that is $\Delta_w = \Delta_{sz} = 1$, see Sect. 3.3.4. In the initial situation, total resource requirement of 9 exceeds the availability of $c_k = 5$. The precedence $A \prec B$ is added to partially resolve the resource conflict. Because this results in a total requirement of 6 for time period $[0, 1)$, there is still a resource conflict, so that the algorithm adds the arc $C \prec A$. Because of the previously added $A \prec B$, not only A is shifted forward but B is as well. Note, however, that now B can be scheduled together with C, as they together have a resource usage of 5, reducing the makespan by 1. In other words, by keeping $A \prec B$ fixed when adding other precedence constraints, the resulting schedule may be sub-optimal. The makespan can be improved by removing the previously added precedence constraints so that activities can be shifted backward.

Based on the reasons mentioned above, we propose to add a heuristic method that tries to improve the schedule by removing the previously added precedences and applying backward shifting of activities after each forward shifting step.

Backward shifting procedure

We propose to use a backward shifting procedure that is applied each time after a precedence has been added. When in window w_c a resource conflict is (partially) resolved by adding a precedence $i \prec j$, a new schedule S' is computed, taking into account $i \prec j$, see Sect. 3.3.6. We then check whether activities can be moved backward, which means that previously added precedences are removed. We do so by keeping a list of activities that are eligible for shifting backward, ordered

ascending based on their current starting time. In order to gain efficiency, it is best if we can filter out activities i from this list from which we know that they can definitely not move backward. When activity j is the activity that has been shifted forward in the current iteration, the underlying idea is to filter out activities that end a sufficient time amount before its original start time S_j. One idea is that the resource feasibility of such activities is not influenced by shifting j forward, as there is no window that contains both S_i and S_j. Hence, such activities will not be able to move backward without introducing a resource conflict, otherwise they would already have done so in a previous iteration. The question to be answered is how far before S_j these activities i should complete, or to phrase it differently: what is the latest time before which the resource feasibility cannot be affected by moving activity j forward? This time is given by the latest time t' so that there is a window w that contains both t' and S_j, which is given by

$$t' = \arg\min_{t_w^l} \{w \in W \mid t_w^l \leq S_j < t_w^r\},$$

i.e. the left boundary of the earliest window that contains S_j. This is visualised by means of an example in Fig. 3.13. The strict inequality $S_j < t_w^r$ is used, because when j starts exactly at the right boundary of a window, it does not effect that window.

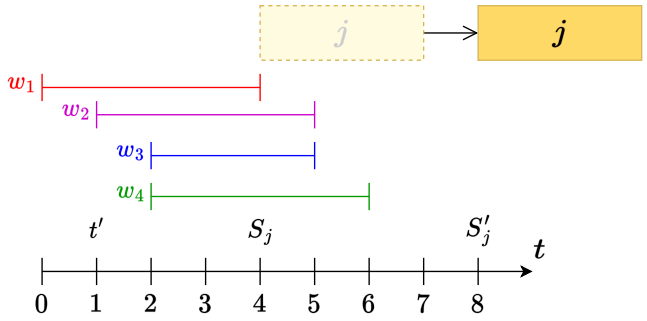


Figure 3.13: *Activity j is shifted forward in time. Given is the window configuration $W = \{w_1, w_2, w_3, w_4\}$. The left boundary of window w_2, denoted by t', is the latest time such that the resource feasibility of any activity completing on or before t' is unaffected by moving activity j forward.*

Given a list of eligible activities, ordered ascending based on start times in the input schedule S', we remove the first item from the list and check whether and how far an activity can be moved backward. If it can be moved backward, we do so and update the schedule to S'', after which we again remove the first item from the list and check if it can be moved backward, until the list is empty.

Let i be an activity for which we want to check if it can be moved backward. The time range over which it can be moved backward is given by a lower bound T_i and upper bound $\hat{T}_i$. The upper bound is given by the current start time of activity i, i.e. $\hat{T}_i = S_i''$. We then check for each $t \in \{T_i, \ldots, \hat{T}_i - 1\}$ if i can feasibly start at t and we look for the smallest of such time points. In order to make the algorithm more efficient, we look for a good T_i so that less time instances have to be checked. We need to make sure that the new start time is not earlier than the fixed

lower bound value LB_i that was provided as input to the CRA, see Sect. 3.3.3. Furthermore, an activity i cannot start earlier than the completion time of its predecessors. A suitable lower bound for activity i would therefore be the maximum over LB_i and the maximum completion time of its direct predecessors, i.e. $T_i = \max\{LB_i, \big(S_j^{''} + d_j | (j \prec i) \in P\big)\}$. Note that we use the updated schedule $S^{''}$ for calculating T_i and $\hat{T}_i$, so that updates (activities that moved backward) of predecessor activities are taken into account. Let t_i be the candidate time at which we want i to start. The procedure is then to start with $t_i = \hat{T}_i - 1$ and decrement with one unit until $t_i = T_i$. At each time instance t_i, we check if i can feasibly start at that time via the `CheckFeasibility` method, which is further explained below. All of the above is summarised in pseudo code given by Alg. 5.

Algorithm 5: MoveBackward(S', S_j, P, W, d, u, c, LB)

```
S'' ← S'
t' ← arg min_{t^l_w} {w ∈ W | t^l_w ≤ S'_j < t^r_w}
eligible ← [i | S'_i + d_i > t']
eligible ← sort(eligible)                    //Sort ascending based on start times in S'
while eligible ≠ ∅ do
    i ← eligible.pop(0)
    T̂_i ← S''_i
    T_i ← max{LB_i, (S''_j + d_j | (j ≺ i) ∈ P)}
    t_i ← T̂_i − 1
    while t_i ≥ T_i do
        if CheckFeasibility(i, t_i, S'', P, W, d, u, c) then
            S''_i ← t_i
        end
        t_i ← t_i − 1
    end
end
return S''
```

Finding a new feasible start time

When moving an activity backward to time t_i, we need to make sure that no new conflicts are introduced before t_i. In order to check if i can feasibly start at t_i, it means that starting activity i at t_i should not introduce new conflicts, both time and resource conflicts.

To remain time-feasible, we should verify that all predecessors of i complete before t_i, that is

$$S_j^{''} + d_j \leq t_i \quad \forall (j \prec i) \in P.$$

This constraint, however, is already enforced by how we defined T_i together with $t \in \{T_i, \ldots, \hat{T}_i - 1\}$.

To remain resource-feasible, we should have that the new execution period $[t_i, t_i + d_i)$ does not lead to new resource conflicts. This means that we need to check for any window w that overlaps with $[t_i, t_i + d_i)$ if the new total resource requirement in w does not exceed the total availability in w for any resource $k \in R$.

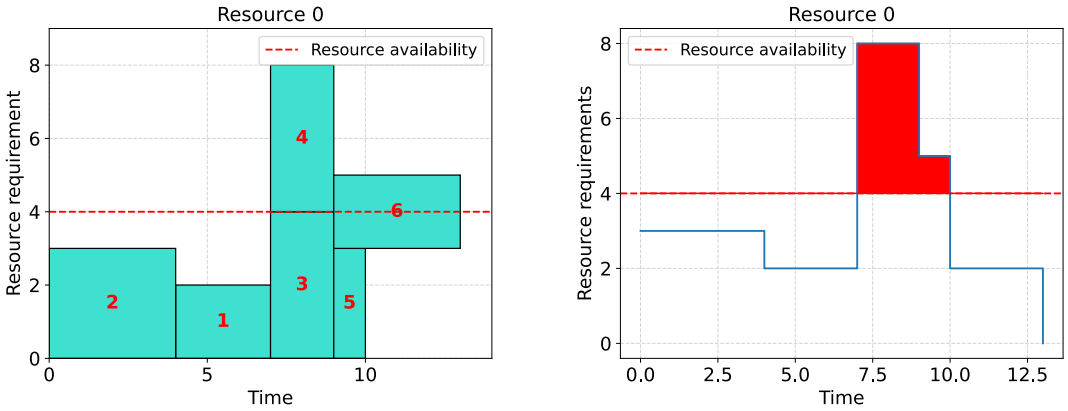
Pseudocode for the algorithm that checks the feasibility of activity i starting at t_i is given in Alg. 6. With $S^{''} \cup (S_i^{''} = t_i)$ we denote the schedule $S^{''}$ for which the start time of activity i has been changed to t_i.

Algorithm 6: CheckFeasibility(i, t_i, $S^{''}$, P, W, d, u, c)

```
overlapping_w ← [w ∈ W | d_{i,w}(S'' ∪ (S_i'' = t_i)) > 0]
for w ∈ overlapping_w do
    for k ∈ R do
        ava_total ← |w| · c_k
        req_total ← Σ_i d_{i,w}(S'' ∪ (S_i'' = t_i)) · u_{i,k}
        if req_total > ava_total then
            return FALSE
        end
    end
end
return TRUE
```

Running example continued

Say that after some steps of the Conflict Resolution, the precedences $2 \prec 3$, $2 \prec 1$ and $1 \prec 4$ have been added, leading to the schedule and resource profile from Fig. 3.14.



(a) *Schedule S' before backward move.*

(b) *Resource profile before backward move.*

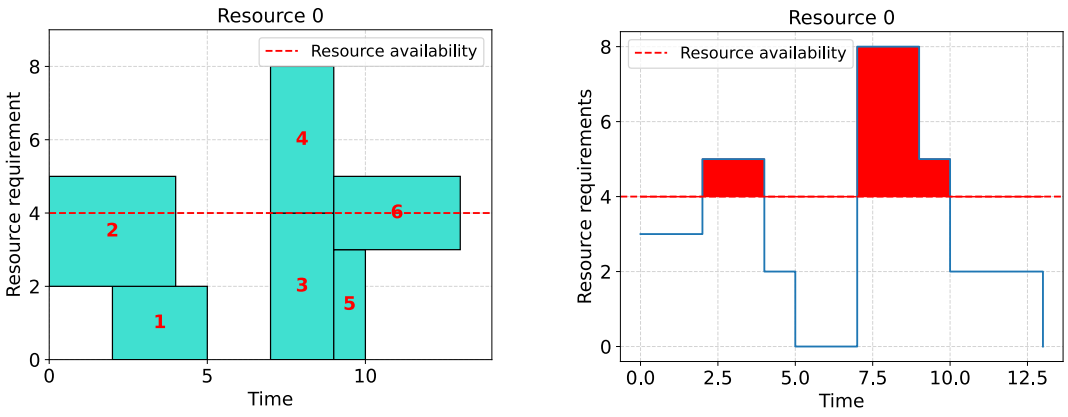
Figure 3.14: *The schedule schedule S' and corresponding resource profile after $2 \prec 3$, $2 \prec 1$ and $1 \prec 4$ have been added.*

In this situation, activity 1 can be resource-feasibly moved backward two time units so that $S_1 = 2$. In order to see that this is correct, we need to consider all windows that overlap with the execution time interval $[2, 5)$ of activity 1. The windows from W that overlap are given in Table 3.4, together with the old and new resource requirement in that window.

Table 3.4: *Resource requirement in windows that overlap with new execution time interval* $[2, 5)$ *of activity* 1*, before and after it has been moved backward from* $[4, 7)$*.*

	Total window requirement	
Window	**Old**	**New**
$[0, 4)$	**12**	**16**
$[1, 5)$	**11**	**15**
$[2, 6)$	**10**	**12**
$[3, 7)$	**9**	**7**
$[4, 8)$	**14**	**10**

As can be seen from Table 3.4, in each of the windows overlapping with $[2, 5)$, the total resource requirement is less than or equal to the total aggregate availability of 16. Furthermore, no precedence from P is violated by doing so. Therefore, activity 1 can indeed feasibly start at time 2. This leads to the schedule and resource profile from Fig. 3.15.



(a) *Schedule* S'' *after backward move.*

(b) *Resource profile after backward move.*

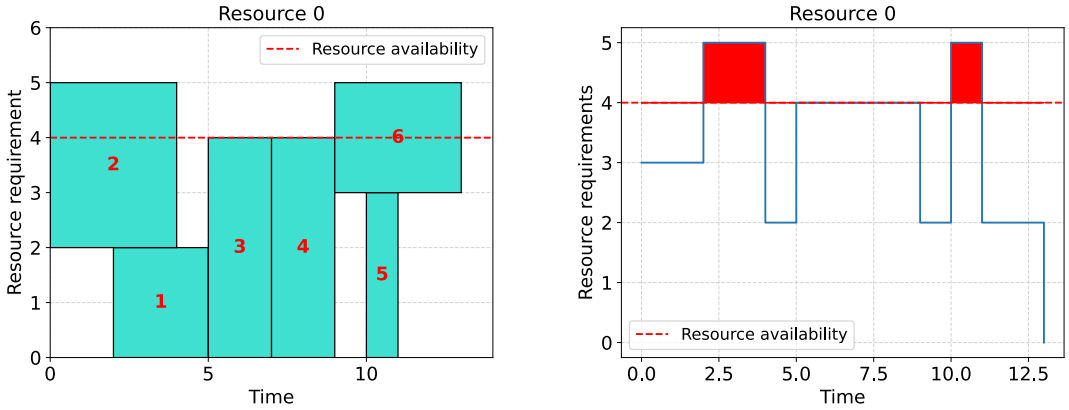
Figure 3.15: *The schedule schedule* S'' *and corresponding resource profile after activity* 1 *has been moved backward to start at time* 2*.*

After shifting activity 1 backward, other activities can move backward without violating time-feasibility, however this would lead to a violation of resource-feasibility so that none of the activities can move back further at this point. To see this, we consider the window requirements after moving backward either activity 3 and 4 (it has the same effect as they have the same resource requirement and duration) one time unit in the new schedule S''. It can be shown that this leads to a resource conflict in the window $[6, 10)$ as the resource requirement in this window is 21, whereas the aggregated availability is 16. As a result, activity 3 nor 4 can be moved backward one unit. It can be shown that they cannot move backward more units either. Since activity 5 and 6 are successors, they cannot move backward either.

The main reason for why activity 3 or 4 cannot move backward is because of the future resource conflict that is yet present around time $t = 9$. Since the CRA has not yet 'worked' on windows around this time period, resource conflicts are still present. This negatively effects the resource requirement in the windows whose resource requirement are used to determine whether a backward move is feasible. Once this future conflict is resolved, it will be possible to shift activity 3 or 4 backward, see Fig. 3.16a.

3.3.8 Running Example: Output

Given the time and resource-infeasible input schedule S_{in} from Fig. 3.5, when using sliding window configuration with $\Delta_w = 4$ and $\Delta_{sz} = 1$ and use the priority rule MLST, we obtain the schedule S_{out} and corresponding resource profile after the CRA has been applied that is depicted in Fig. 3.16.



(a) *Final schedule.* **(b)** *Final resource profile.*

Figure 3.16: *The resulting schedule S_{out} and resource profile after the CRA has been applied.*

As one can see from Fig. 3.16a, all of the precedence constraints from Eq. (2.1) are satisfied. When considering the resource profile from Fig. 3.16b, we do see that the classical resource constraints from Eq. (2.2) are violated. However, one can easily verify that the aggregated constraints from (3.1) are satisfied. Together, this leads to a time and resource-feasible schedule with a makespan of 13.

3.4 Algorithmic Properties

In this section, we mention some of the properties of the CRA. In Sect. 3.4.1, we prove that the CRA returns a conflictless schedule in a finite number of steps. In Sect. 3.4.2 we prove that the CRA generates so-called active schedules.

3.4.1 The CRA is Finite and Returns a Conflictless Schedule

Any well-defined and practical algorithm should return the expected result in a finite number of steps. In this section, we prove that given the tuple (I, S_{in}, LB) as input to the CRA, it returns a conflictless schedule S_{out} in a finite number of steps. We do so by first proving some lemmas, after which we give the final proof of the claims above in Theorem 3.4.1.

Lemma 3.4.1. *The input schedule S_{in} is transformed into a time-feasible schedule in a finite number of steps. Furthermore, after each iteration of the CRA algorithm, the schedule remains time-feasible.*

Proof. Given the input schedule S_{in}, it is transformed to a time-feasible schedule S in a finite number of steps, see Sect. 3.3.3. This schedule is then iteratively adjusted in order to solve resource conflicts. In each iteration, S might change due to `MoveForward` that adds a precedence $i \prec j$. In the new schedule S', time-feasibility is ensured as S' is constructed using an earliest start procedure, taking into account the precedences from P and $i \prec j$, see Sect. 3.3.6. Subsequently, S' might then be updated to S'' by moving activities backward via the `MoveBackward` method. In doing so, an activity is only moved backward if its new start time is time- and resource-feasible, see Sect. 3.3.7. Summarising, the input schedule S to each iteration of CRA is time-feasible and the same holds for the output schedule S''. ■

Lemma 3.4.2. *Given a schedule S for which a resource conflict is detected in window w_c. The CRA resolves all of the conflicts in w_c a finite number of steps.*

Proof. Assume that the CRA does not make use of the `MoveBackward` method. Given schedule S, the `MoveForward` method tries to resolve resource conflicts by adding precedences $i \prec j$. Activity i and j are both selected from the decision set $D = \{i \in A | d_{i,w_c}(S) > 0\}$, see Sect. 3.3.6. Let $|D|$ denote the corresponding cardinality of D. The CRA selects precedences from D until window w_c is free of resource conflicts. Since adding the precedence $i \prec j$ leads to activity j shifting forward in time, $|D|$ can not increase by adding precedences. Precedences can be added as long as there are activities that run in parallel. Considering the number of 2-permutations of D, it takes at most $|D|!/(|D|-2)!$ precedences for all activities to run sequentially, which means that w_c does not contain activities that run in parallel. In this case, the total resource requirement for all resources $k \in R$ in window w_c is less than the total availability. This follows from the assumption $u_{i,k} \leq c_k \; \forall i \in A, \; \forall k \in R$. In case no activities run in parallel, we have that

$$\sum_{i \in A} d_{i,w_c}(S') \cdot u_{i,k} \leq \sum_{i \in A} d_{i,w_c}(S') \cdot c_k = c_k \cdot \sum_{i \in A} d_{i,w_c}(S') \leq c_k \cdot |w_c| \; \forall k \in R.$$

By the aggregated resource constraints from Eq. (3.1), it then follows that window w_c is free of resource conflicts.

Assume now that in each iteration, the `MoveBackward` method is used by the CRA. This does not change the result from above. By construction, as long as window w_c contains resource conflicts, activities cannot move backward so that they overlap with w_c, see Alg. 6. As a consequence, `MoveBackward` does not influence the resource feasibility of window w_c nor does it change the decision set D as long as it contains resource conflicts. Furthermore, if w_c is free of resource conflicts, `MoveBackward` does not introduce new resource conflicts in w_c.

Summarising, after at most a finite number of iterations, window w_c is free of resource conflicts. ■

Lemma 3.4.3. *Let window w_c be the window in which the CRA is resolving a resource conflict. Then after resource conflicts have been resolved in w_c, no resource conflict is introduced in any window w with $w < w_c$.*

Proof. Let w_c be the window in which a conflict is detected for schedule S. Then, the `MoveForward` method tries to (partially) resolve the conflict by adding a precedence $i \prec j$. This leads to a new schedule S' for which $S'_i \geq S_i \; \forall i \in A$ holds. In Sect. 3.3.4, we mentioned the assumption that for any window $w \in W$, $t^l_{w+1} \geq t^l_w$ and $t^r_{w+1} \geq t^r_w$ holds for the left and right boundaries, respectively. This assumptions together with the fact that adding a precedence shifts activity j forward in time, means that $d_{i,w}(S') \leq d_{i,w}(S) \; \forall i \in A \, , \forall w \in W : w < w_c$. Subsequently,

$$\underbrace{\sum_{i \in A} d_{i,w}(S') \cdot u_{i,k}}_{\text{New resource req. in } w} \leq \underbrace{\sum_{i \in A} d_{i,w}(S) \cdot u_{i,k}}_{\text{Old resource req. in w}} \quad \forall w \in W : w < w_c, \; \forall k \in R.$$

In other words, moving activities forward in time by the `MoveForward` method will not increase the resource requirement of a window in the past and thus cannot lead to resource conflicts in the past.

Given schedule S', in each iteration activities might be moved backward by the `MoveBackward` method to get to a new schedule S''. In doing so, an activity can only start earlier if this does not lead to a resource conflict in any window overlapping with its new start time, see Sect. 3.3.7. Therefore, `MoveBackward` cannot introduce resource conflicts for windows in the past. ■

Theorem 3.4.1. *Given a time- and resource-infeasible input schedule S_{in}. The CRA transforms this to a time- and resource-feasible schedule into a finite number of iterations.*

Proof. S_{in} is initially transformed into a time-feasible schedule in a finite number of steps, see Lemma 3.4.1. When w_1 is the first window in which resource conflicts occur, this is resolved in a finite number of iterations, see Lemma 3.4.2. The CRA then proceeds to the next window w_c in which a resource conflict occurs and resolves the conflict, so that no conflict is introduced in any window w with $w < w_c$, see Lemma 3.4.3. This proceeds until all windows w in the finite set W are free of resource conflicts, taking a finite number of steps for each window. Furthermore, after each iteration, the schedule remains time-feasible, see Lemma 3.4.1, which completes the proof. ■

3.4.2 The CRA Generates Active Schedules for RCPSP instances

An important question for a heuristic is the kind of solution space that the heuristic is operating on, which is an indication for whether the heuristic will generally be capable of finding the optimal solution. In this section, we show that the CRA produces schedules that belong to the set of active schedules. This is a desirable property as at least one optimal solution to any instance of RCPSP belongs to this class of schedules, for which we give a short proof.

Before doing so, we explain the notion of a *local* and *global left shift*, see [1].

Definition 16. *A global left shift is an operator $LS(S, i, \Delta)$ that transforms a feasible schedule S into an feasible identical schedule S', except for $S'_i = S_i - \Delta$. A global left shift is local when in addition all schedules obtained by $LS(S, i, \rho)$ with $0 < \rho \leq \Delta$ are also feasible.*

Given the notion of local and global left shifts, we can define the set of *active schedules*.

Definition 17. *An active schedule is a time and resource-feasible schedule such that none of its activities can be globally left shifted without violating feasibility.*

A well-known result for the RCPSP with regular objective (such as makespan minimisation) is that the set of active schedules contains at least one optimal solution, see e.g. Artigues et al. [1].

Theorem 3.4.2. *For the RCPSP with makespan minimisation objective, the set of active schedules contains at least one optimal solution.*

Proof. The proof follows by contradiction. Suppose that for an instance I of the RCPSP, no active schedule is optimal. Then, among all optimal schedules, at least one of the activities i can be feasibly started earlier. We can move i backward by starting it as early as possible. As a result, for the new schedule the makespan can only be the same or lower. However, since the schedule is optimal, the makespan cannot be lower thus it must be the same. As long as the schedule is not active, we keep backward shifting activities as far as possible, until we get a schedule for which none of the activities can be feasibly started earlier. Since the makespan remains the same as the optimal makespan value, we end up with an active schedule that is optimal, which contradicts with our initial assumption. ■

Theorem 3.4.3. *Given any instance $I = (A, R, P, W, d, u, c)$ of the WARCPSP with corresponding (infeasible) schedule S_{in}. Assume that W is of the RCPSP configuration type, which means that I reduces to an instance of RCPSP. Moreover, assume that $LB_i = 0\ \forall i \in A$, i.e. there are no additional lower bound restrictions on the start times of the activities. Then, CRA transforms this schedule to an active schedule.*

Proof. First of all, S_{in} is transformed into an earliest start schedule S, see Sect. 3.3.3. An earliest start schedule clearly is an active schedule, as none of the activities can be further left shifted without introducing a time-conflict. Schedule S is then iteratively adjusted in order to resolve resource conflicts. Each iteration consists of an action by the `MoveForward` and `MoveBackward` method. When an activity is forward shifted by `MoveForward`, the resulting schedule S' may not be active anymore, see the example from Fig. 3.17 as an illustration.

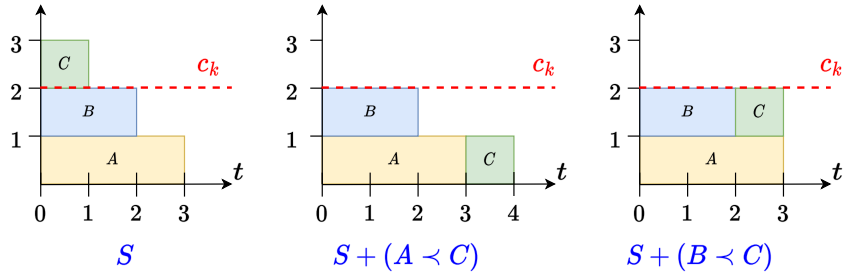


Figure 3.17: *When the CRA decides to add the precedence $A \prec C$, the resulting schedule is feasible but non-active as C can feasibly start earlier. When adding the precedence $B \prec C$ instead, this leads to an active schedule which is also optimal.*

The `MoveBackward` method, however, trivially turns the possibly non-active schedule $S^{'}$ into an active schedule $S^{''}$. This is true since by design, `MoveBackward` considers all activities that can

possibly be locally or globally left-shifted. Then, it sequentially left-shifts activities as far as is feasibly possible. For the RCPSP, this means that the start time of an activity will always coincide with the completion times of other activities or the start of the time horizon. As a result, in S'' no activities can be further locally or globally left-shifted without introducing a time or resource-conflict, hence S'' is active. Since after each iteration, the resulting schedule S'' is active, it follows that the final schedule S'' returned by the CRA is active. ■

3.5 Forward Backward Improvement

In this section, we mention a noticeable observation that we made near the end of period that was reserved for this thesis. During this research, we experimented with different techniques, aiming to find improvements to the CRA. One of the techniques we considered is a post-processing technique referred to in literature as *Forward-Backward Improvement* (FBI) or *Justification*. It turned out that when transforming a schedule S to a time-feasible earliest start schedule and applying FBI, this leads to slightly better results in terms of makespan and run-time performance, see Sect. 3.5.3 for more details.

For more details and notation, we refer to de Nijs [12].

3.5.1 FBI Example

FBI is inspired by the concept of free slack, which is the amount of time that an activity can start later without affecting the makespan, i.e. $LS_i - S_i$ for activity i, where LS_i denotes the latest start time of activity i. In active schedules, see Def. 17, activities start as soon as possible. This means that activities with a positive amount of slack consume resources at an early time, even though they do not really need to do so. When these activities are shifted forward in their slack (so that the makespan remains unaffected), resources may be freed up earlier in time, allowing other activities to start earlier, which may ultimately reduce the makespan.

Algorithmically, FBI exploits the idea explained above via a backward and forward pass. An instance of a schedule is scheduled backwards so that activities start as late as possible in their slack windows. Then, this solution is rescheduled "in the forward direction", so that resources that are freed up earlier in time can be used by other activities. An illustrative example of this can be found in Fig. 3.18.

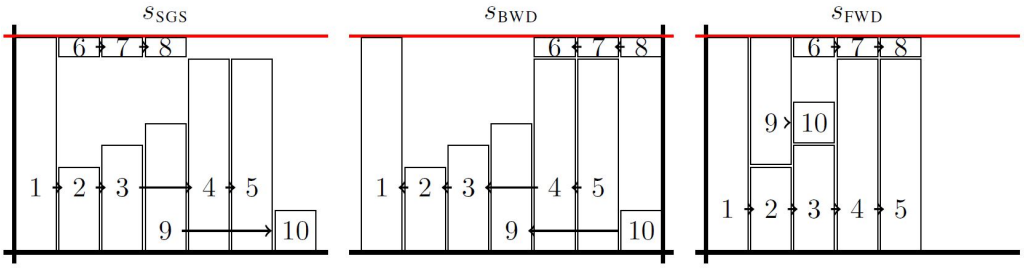


Figure 3.18: *Example of Forward-Backward Improvement [12].*

The schedule S_{SGS} on the left is generated via an SGS using the MLFT priority rule. In S_{SGS}, activity 6, 7 and 8 are the only activities that have free slack as they all can be shifted 3 units forward without affecting the makespan. Note that if activity 9 were to run parallel with activity 2, this would reduce the makespan as then activity 10 could start earlier as well. Since activity 6 runs in parallel with 2, this is not possible.

The backward pass is applied to the S_{SGS} schedule, which essentially transforms S_{SGS} into a backwards schedule S_{BWD} which is done by reversing all precedence constraints from P, after which the Serial SGS is applied with a priority rule that prioritises the activity with the highest finish time in S_{SGS}. In this backward solution, there is sufficient time for activity 2 and 9 to run parallel.

Consequently, schedule S_{BWD} is rescheduled forward with the original precedence constraints and a serial SGS method with a priority rule that prioritises the activities in S_{BWD} that start earliest. This results in schedule S_{FWD} which has a lower makespan than the original schedule S_{SGS}.

3.5.2 FBI Pseudo-code

In the below, we briefly explain the FBI algorithm and provide pseudo-code for each of its components.

The main body of the FBI algorithm can be found in Alg. 7. Given an instance I of the WAR-CPSP and corresponding schedule S, the precedences from P are reversed. Then, the backward pass shifts activities forward in their slack, in an order that prioritises activities with the highest finish time in the input schedule S, resulting in a backward schedule S_{bwd}. The activities are then shifted forward (or backward, depending on what orientation one reasons from), in an order that prioritises activities with the earliest start time in S_{bwd}, yielding the final schedule S_{fwd}.

Algorithm 7: FBI$(S, I = (A, R, P, W, d, u, c))$

$P' \leftarrow \emptyset$
for $(i \prec j) \in P$ **do**
  $P' = P' \cup (j \prec i)$
end
$I' = (A, R, P', W, d, u, c)$
S_{bwd} =Justify(S, I)
S_{fwd} =Justify(S_{bwd}, I')
return S_{fwd}

The forward and backward pass are both executed by the `Justify` method, see Alg. 8. This method first reverses the precedences from P, which it uses to return a schedule that is created by an SGS. The `SGS` method uses a so-called justification priority rule π_j. Given a decision set D containing activities to be scheduled, π_j prioritises the activity with $\max_{i \in D} S_i + d_i$.

Algorithm 8: Justify($S, I = (A, R, P, W, d, u, c)$)

$P' \leftarrow \emptyset$
for $(i \prec j) \in P$ **do**
  $P' = P' \cup (j \prec i)$
end
$I' = (A, R, P', W, d, u, c)$
return SGS(I', π_j)

The `SGS` method takes an instance I of the WARCPSP as input, together with a priority rule π. Starting with an empty schedule, it schedules activities in a serial way. It does so by maintaining a decision set D that contains all unscheduled activities that are eligible to be scheduled, that is, for which all predecessors have been scheduled. This is verified via the `DetermineEligible` method. Among the activities in D, π selects the activity i with highest priority. Then, i is scheduled at the first time point t at which it can feasible start. This latter is checked by the `IsFeasible` method. Pseudocode can be found in Alg. 9.

Algorithm 9: SGS($I = (A, R, P, W, d, u, c), \pi$)

$S_i \leftarrow$ NULL $\forall i \in A$
$D \leftarrow$ DetermineEligible(S, I)
while $D \neq \emptyset$ **do**
  $i \leftarrow \pi(D, I, S)$ `//Select activity based on priority rule` π
  $t \leftarrow 0$
  while not IsFeasible(i, t, S, I) **do**
    $t \leftarrow t + 1$
  end
  $S_i \leftarrow t$
  $D \leftarrow$ DetermineEligible(S, I)
end

As mentioned above, given a partial schedule S, the `DetermineEligible` method determines which activities can be scheduled, meaning that their predecessors should be scheduled, i.e. are part of the partial schedule. Pseudo-code can be found in Alg. 10.

Algorithm 10: DetermineEligible($S, I = (A,\ R,\ P,\ W,\ d,\ u,\ c\ )$)

```
D ← ∅
for i ∈ A do
    if S_i =NULL then
        ELIGIBLE = TRUE
        for (j ≺ i) ∈ P do
            if S_j =NULL then
                ELIGIBLE = FALSE
            end
        end
        if ELIGIBLE = TRUE then
            D ← D ∪ i
        end
    end
end
return D
```

When the `SGS` method selects an activity i to be scheduled, the `IsFeasible` method is used to find a feasible start time t. This method checks for both time and resource feasibility. The time-feasibility constraints from Eq. (2.1) imply that all predecessors of i should complete on or before t. The aggregated resource constraints from Eq. (3.1) imply that for any window w overlapping with $[t, t + d_i)$, the total resource requirement should not be more than the total availability. The `IsFeasible` method is in that sense almost similar to the `CheckFeasibility` method of the CRA, as expressed in Alg. 6. Pseudo-code can be found in Alg. 11. Here, $S \cup (S_i = t_i)$ denotes the partial schedule S to which i has been added with start time t_i.

Algorithm 11: IsFeasible($i, t_i, S, I = (A,\ R,\ P,\ W,\ d,\ u,\ c\ )$)

```
for (j ≺ i) do
    if S_j =NULL or S_j + d_j > t_i then return FALSE
end
overlapping_w ← [w ∈ W | d_{i,w}(S ∪ (S_i = t_i)) > 0]
for k ∈ R do
    for w ∈ overlapping_w do
        ava_total ← |w| · c_k
        req_total ← Σ_j d_{j,w}(S) · u_{j,k} + d_{i,w}(S ∪ (S_i = t_i)) · u_{i,k}
        if req_total > ava_total then
            return FALSE
        end
    end
end
return TRUE
```

3.5.3 Experimental Observation

During experimental evaluation, we noticed a good performance of FBI when being applied to earliest start schedules. To be more specific, given an instance I of WARCPSP, we compute an earliest

start S_{ES} schedule based on the earliest start procedure given by Alg. 1 without using further lower bound restrictions, i.e. $LB_i = 0 \; \forall i \in A$. We then apply FBI to S_{ES}. In the rest of this paper, we refer to this as the *FBI approach*. Surprisingly, experiments showed that the performance of the FBI approach is generally better than the performance of the CRA, both in terms of makespan and run-time. These experiments can be found in Sect. 4.1.6.

There is one noticeable difference between the CRA and FBI. As mentioned in Sect. 1.2, the CRA is designed such that it can be applied in an online setting, allowing certain parts of the plan to remain unchanged. This becomes clear from how resource conflicts are detected and resolved. This is not the case for the FBI as described in this section. The FBI does not resolve conflicts in a time-advancing fashion, but it rather uses a forward and backward pass on the entire plan. It neither uses lower bounds on the earliest start values of the activities, which the CRA uses to ensure that after replanning, the activities do not start earlier than desired, see Sect. 3.3.3. Nevertheless, we mention the FBI as we believe it can be adapted so that it does have the online characteristic.

Part 4

Results & Conclusion

4.1 Experimental Analysis

In this section, we test the performance of the CRA by means of different experiments. The CRA was coded in Python 3.8 and all experiments were performed on a PC with 2.8GHz CPU and 17GB RAM.

In Sect. 4.1.1, we mention the instances that we used for our experiments and the parameters that were used to generate them. In Sect. 4.1.2, we introduce the two performance measures that we use to analyse the performance of the CRA. The general performance of the CRA is described in Sect. 4.1.3. In Sect. 4.1.4, we show by means of an experiment that backward shifting activities does generally lead to better schedules. In Sect. 4.1.5, we analyse the influence of the parameters that were used to generate the instances. Finally, in Sect. 4.1.6, we compare applying the CRA and applying the FBI method.

4.1.1 Instance Measure Parameters & PSPLIB Data

Before proceeding with the experiments, we first mention the data that will be used in the experiments and the underlying parameters that were used to generate this data.

Generating RCPSP instances

$\mathcal{NP}$-hard problems cannot (unless $\mathcal{P} = \mathcal{NP}$) be efficiently solved for every instance. Nevertheless, it can be the case that specific classes of instances can be solved in polynomial time. It can therefore be useful to look at the so-called *instance hardness*, i.e. how hard it is to solve a specific instance, and to understand how different parameters of the instance affect its hardness. Identifying such parameters can be useful in generating data sets for experimental evaluation, as it helps to generate different classes of instances and hopefully also captures the average performance of an algorithm that solves these instances. This is exactly what Kolisch and Sprecher [26] did when generating RCPSP instances for their PSPLIB data set. This data set contains benchmark instances for the RCPSP and were generated based on different values for the parameters *Network Complexity* (NC), *Resource Factor* (RF) and *Resource Strength* (RS) by ProGen [27]. These parameters are as follows:

- **Network Complexity**: NC is an indicator of the constrainedness of an instance as a result of the amount of precedences. In terms of the AON graph, it is defined as the number of non-redundant arcs per node (including the dummy source and sink node). An arc (i, j) is said to be redundant if it is an element of the transitive closure of $G = \big(A, P \setminus (i \prec j)\big)$, i.e. there is already a path from i to j. As a result, NC is defined as

$$NC = \frac{|\text{Non-Redundant Precedences}|}{n}.$$

  Kolisch et al. [27] showed that as NC increases, the instance hardness decreases. Intuitively, this makes sense as more precedence constraints lead to a smaller feasible region, shrinking the enumeration tree/search space, which was also concluded by Alvarez-Valdes [33].

- **Resource Factor**: RF is an indicator for the average portion of resources that is requested by each activity, or differently stated, how many different resources are used by the average activity. It is defined as

$$RF = \frac{1}{nm} \sum_{i \in A} \sum_{k \in R} \min\{1, u_{i,k}\}.$$

  If $RF = 1$, then each job requires all resources, if $RF = 0$, all jobs require zero resources. Kolisch et al. [27] showed that there is a positive correlation between RF and instance hardness.

- **Resource Strength**: RS relates the resource demand of the activities to the resource availability and is an indicator for the size of a resource conflict. For each resource k, RF_k is defined as the ratio between the maximal usage of resource k by an activity and the highest total usage for resource k at any time in the earliest start schedule S_{ES}:

$$\begin{aligned} u_k^{max} &= \max_{i \in A} u_{i,k} \\ u_k^{peak} &= \max_{t \in \mathcal{T}} \sum_{i \in \mathcal{A}(S_{ES}, t)} u_{i,k} \\ RS_k &= \frac{c_k - u_k^{max}}{u_k^{peak} - u_k^{max}}. \end{aligned}$$

  Kolisch et al. [27] showed that RS has the strongest impact on instance hardness: a larger RS value corresponds with a lower instance hardness. The RS of an instance is then defined to be the average over the resource strengths for all resources.

In our experiments, we use the PSPLIB-120 instances, which all have 120 activities. The parameter values used for generating these instances can be found in Table 4.1.

Table 4.1: *Parameter values used to generate PSPLIB-120 instances, the table is from [25].*

NC	RF	RS
1.5	0.25	0.1
1.8	0.50	0.2
2.1	0.75	0.3
	1.00	0.4
		0.5

A full factorial design leads to $3 \cdot 4 \cdot 5 = 60$ different parameter value classes, for which 10 instances were generated, leading to 600 different PSPLIB-120 instances in total.

Using RCPSP instances in a WARCPSP context

Since these RCPSP instances, we add a window configuration W to obtain instances I of the WARCPSP. From the experiments, the type of window configuration will be clear. Apart from the input I and $LB_i = 0 \ \forall i \in A$, the input schedule S_{in} is defined to be an earliest start schedule corresponding to the precedences from P. This leads to a time-feasible but resource-infeasible schedule, so that the CRA needs to resolve these resource conflicts. We refer to this approach as the *CRA approach*.

Definition 18. *The CRA approach refers to applying the CRA to the input tuple (I, S_{in}, LB) in order to resolve conflicts in S_{in}. The WARCPSP instance $I = (A, R, P, W, d, u, c)$ includes the predefined window configuration W. Schedule S_{in} is defined as the earliest start schedule $ES(P, LB)$. Finally, the vector of earliest start lower bounds LB is the vector of all zeros, i.e. $LB_i = 0 \ \forall i \in A$.*

Note that although S_{in} is already time-feasible by construction, this does not pose a problem for testing the model performance, as the CRA would otherwise start with constructing the exact same earliest start schedule by itself, which takes a negligible amount of time.

4.1.2 Performance Measures

In order to analyse the performance of the CRA, we apply the CRA to each instance of PSPLIB-120. We measure its performance by using two performance measures.

Makespan measure

For each instance of PSPLIB-120, we compare the makespan resulting from the CRA approach with the corresponding best known makespan found so far. Because of the complexity of solving these RCPSP instances, many of these instances are unsolved as to this date. The best known (heuristic) solutions are available via http://solutionsupdate.ugent.be/dataset/j120. This project was inspired by Vanhoucke et al. [43] who developed a procedure for researchers to upload solutions to RCPSP instances from PSPLIB with a makespan objective.

To be more specific, given any tuple (I, S_{in}, LB) as input, we apply the CRA, yielding makespan C_{max}. Provided the best-known makespan C_{max}^{OPT}, we use μ_{OPT} to measure the relative change in makespan, that is

$$\mu_{OPT} = \frac{C_{max} - C_{max}^{OPT}}{C_{max}^{OPT}}.$$

Run time measure:
Apart from μ_{OPT}, we also use the run time of the CRA as an indication of performance. Given input tuple (I, S_{in}, LB), measure μ_{CPU} denotes the time (in seconds) it takes for the CRA to return a feasible schedule S_{out}.

4.1.3 General Performance of the CRA Approach

In this section we show the general performance of the CRA approach when using instances of PSPLIB-120. We perform experiments for three different window configurations W, see Sect. 3.3.4: all configurations use a step-size of $\Delta_{sz} = 1$, one uses a window size of $\Delta_w = 1$, which means that the problem reduces to the RCPSP, the other use a window size of $\Delta_w = 4$ and 8, respectively. The results can be found in Fig. 4.1.

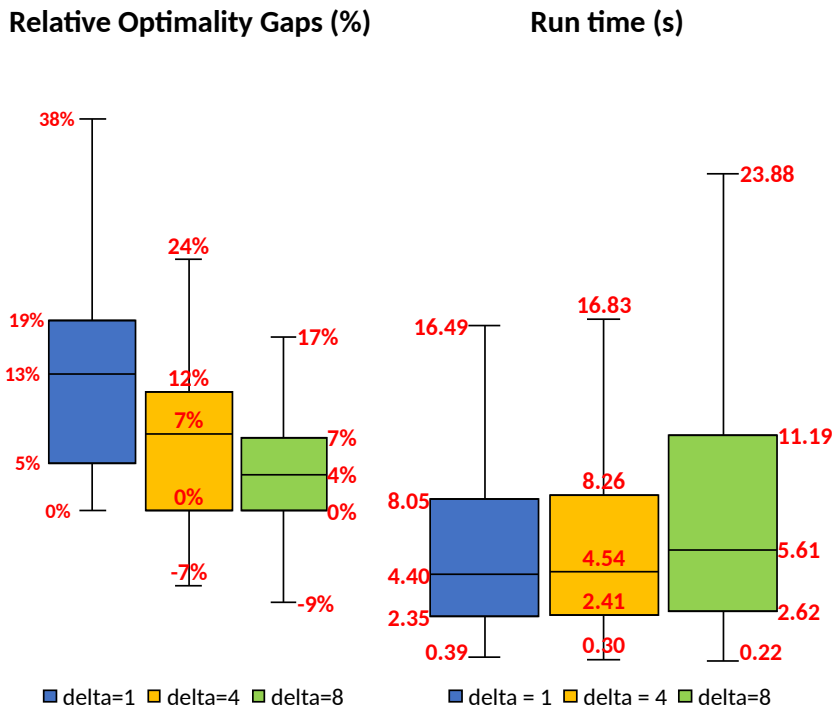


Figure 4.1: *General performance of the CRA approach on PSPLIB-120 instances. Performance is measured by relative optimality gap μ_{OPT} and run time μ_{CPU}.*

Observation 1. *When using detailed planning for PSPLIB-120 instances, the median makespan performance of the CRA approach is within 13% of the optimal solution and is never off by more than 38% from the optimal solution in the most extreme case. The median run time is below 5 seconds and the run time is well below 17 seconds for all instances.*

Observation 2. *When using aggregated planning for PSPLIB-120 instances, the general makespan performance of the CRA approach decreases on average as the window size of the sliding window configuration increases.*

From Fig. 4.1 one can observe that as the window size increases, on average the values of μ_{OPT} decrease. Since μ_{OPT} measures the relative increase from the best known solution value C_{max}^{OPT} that remains fixed, this means that this is caused by a decrease in makespan when applying the

CRA with increasing window sizes. This is as expected, as a larger window size leads to a stronger relaxation of the RCPSP resource constraints from Eq. (2.2), allowing for more violations.

Observation 3. *When using aggregated planning for PSPLIB-120 instances, the run time of the CRA approach increases as the window size of the sliding window configuration increases.*

From Fig. 4.1 once can observe that for a larger window size, the run time increases moderately. After run time analysis, we conclude that this is mainly due to the way the `MoveBackward` method selects activities that could be moved backward. When the window size increases and activity j is shifted forward, the change of the resource profile affects a larger number of activities that lie in an overlapping window with S_j, see Fig. 3.13. As a result, the `CheckFeasibility` method is evaluated more often, see Alg. 5, which has a significant impact on the run time.

4.1.4 Influence of Intermediate Backward Shifting

In this section, we will show the influence of the `MoveBackward` method on the makespan. As is mentioned in Sect. 3.3.7, when keeping the added precedences $i \prec j$ fixed while resolving conflicts, this might lead to unnecessary idle gaps due to a sub-optimal selection of the precedences, resulting in non-active schedules. Moreover, we showed in this section that it is not generally true that an optimal solution to an instance of WARCPSP can be represented by an earliest start schedule corresponding to a set of precedences. `MoveBackward` tries to improve schedules by removing added precedences and shifting back activities after each precedence has been added. To see that this indeed leads to an improvement in terms of the makespan, we compare the results for the CRA approach with and without `MoveBackward`. In the first, added precedences do not remain fixed but may be removed, in the latter they do remain fixed.

For this experiment, we used a sliding window configuration with $\Delta_w = 4$ and $\Delta_{sz} = 1$. For each of the data sets, we computed the relative difference in makespan for the method with `MoveBackward` compared to without using `MoveBackward`.

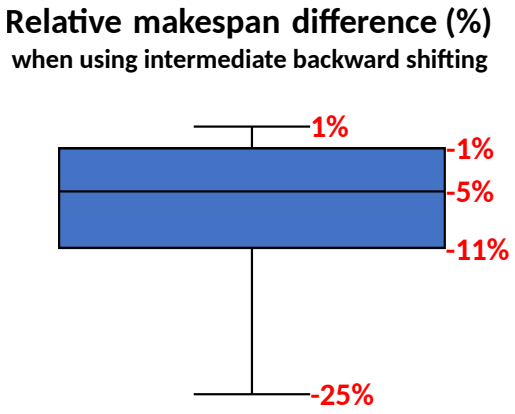


Figure 4.2: *Relative makespan increase of the CRA approach on PSPLIB-120 instances when it does use the* `MoveBackward` *method compared to when it does not.*

Observation 4. *Using the* `MoveBackward` *method as part of the CRA has significant effect on the makespan performance.*

In Fig. 4.2, one can see the results of the comparison between not using and using the `MoveBackward` method. These results clearly show that intermediate backward shifting has a significantly positive impact on the makespan as for almost all instances, the increase is negative, meaning that we see a reduction of makespan when using the `MoveBackward` method. Interestingly, for two instances, the makespan increased, but only by $<= 1\%$. Such a situation can occur because a backward shift may lead to worse heuristic decisions for the consequent forward shifts, resulting in a higher makespan.

4.1.5 Influence of Instance Measure Parameters

In this section, we verify if and how the instance measure parameters from Sect. 4.1.1 influence the performance of the CRA algorithm. We do so by applying the CRA to PSPLIB-120 instances, after which we compare the resulting makespan with the best-known makespan and look at the computational effort needed to resolve all conflicts. For the latter, we consider both the run time, the number of precedences added (i.e. number of iterations) and the total number of unit backward moves by the `MoveBackward` method. In order to measure the influence of a single parameter x, we use a full-factorial design of the other parameters and vary x. This means that for determining the influence of NC, RF and RS we used 200, 150 and 120 instances to determine the value of the statistics, respectively. For those experiments, a sliding window configuration with $\Delta_w = 4$ and $\Delta_{sz} = 1$ has been used. The results can be found in Table 4.2, Table 4.3 and Table 4.4. The statistic $\bar{\mu}_{OPT}$ denotes the average over the relative optimality gaps. The parameters $\bar{\mu}_{CPU}, \bar{\mu}_{fwd}$ and $\bar{\mu}_{bwd}$ denote the average run time, average number of forward moves and average number of unit backward moves, respectively.

Table 4.2: *Influence of Network Complexity NC.*

NC	1.5	1.8	2.1
$\bar{\mu}_{OPT}$ (%)	6.61	6.65	7.25
$\bar{\mu}_{CPU}$ (s)	7.45	6.29	4.87
$\bar{\mu}_{fwd}$	681	581	478
$\bar{\mu}_{bwd}$	38	42	47

Table 4.3: *Influence of Resource Factor RF.*

RF	0.25	0.50	0.75	1.00
$\bar{\mu}_{OPT}$ (%)	1.45	7.20	9.14	9.04
$\bar{\mu}_{CPU}$ (s)	1.96	4.60	7.59	10.67
$\bar{\mu}_{fwd}$	263	501	694	861
$\bar{\mu}_{bwd}$	32	50	48	39

Table 4.4: *Influence of Resource Strength RS.*

RS	0.1	0.2	0.3	0.4	0.5
$\bar{\mu}_{OPT}$ (%)	11.69	9.46	7.27	4.22	1.23
$\bar{\mu}_{CPU}$ (s)	12.06	7.73	5.31	3.65	2.26
$\bar{\mu}_{fwd}$	867	722	568	436	307
$\bar{\mu}_{bwd}$	78	50	38	27	18

Observation 5. *The Network Complexity negatively correlates with the run time of the CRA approach.*

Table 4.2 does not show a strong correlation between NC and the quality of the CRA solutions as measured by $\bar{\mu}_{OPT}$. It does show, however, that as the NC increases, the run time decreases, which correlates with less precedences added, so that the CRA finishes within less iterations. This is in line with the conclusion from Kolisch et al.[27], see Sect. 4.1.1. We also notice a positive correlation between NC and the number of unit backward shifts.

Observation 6. *The Resource Factor has a negative effect on the makespan performance of the CRA approach. Moreover, it positively correlates with the run time.*

From Table 4.3, one can clearly see that the quality of solutions decreases as the RF increases. A reason for this might that the CRA detects and resolves resource conflicts in a given window naively by resolving the conflicts for different resources separately in a sequential order, see Alg. 2 and the fact that resource requirement is not considered in the MLST priority rule. When activity i requires multiple resources, it might be that forwarding shifting i resolves the conflict for one resource k_1 in the conflicting window w_c, but the conflict for another resource k_2 still remains, so that another activity has to be forward shifted to resolve the resource conflict for k_2. In this situation, forward shifting an activity that resolved both conflicts simultaneously (if possible) could have been more efficient. Also, we do see a strong positive correlation between the number of added precedences

and the RF, effecting the run time, which is in line with Kolisch et al. [27]. We do not see a clear correlation between the number of unit backward shifts and RF.

Observation 7. *The Resource Strength positively correlates with the makespan performance and has a negative correlation with the run time of the CRA approach.*

Table 4.4 shows that as the RS increases, the quality of solutions increase and the run time, the number of added precedences and the number of unit backward shifts decrease. The decrease in run time is in line with Kolisch et al. [27].

4.1.6 FBI Performance Comparison

In Sect. 3.5, we mentioned our observation that the FBI approach generally leads to a better performance than the CRA approach.

Definition 19. *The FBI approach refers to applying the FBI to the input tuple (I, S_{in}) in order to resolve conflicts in S_{in}. The WARCPSP instance $I = (A, R, P, W, d, u, c)$ includes the pre-defined window configuration W. Schedule S_{in} is defined as the earliest start schedule $ES(P, LB)$, where LB denotes the vector of all zeros, i.e. $LB_i = 0\ \forall i \in A$.*

These observations follow from experimenting on PSPLIB-120 instances. In Fig. 4.3, one can find the resulting optimality gaps and run times μ_{OPT} and μ_{CPU}, respectively, for these experiments.

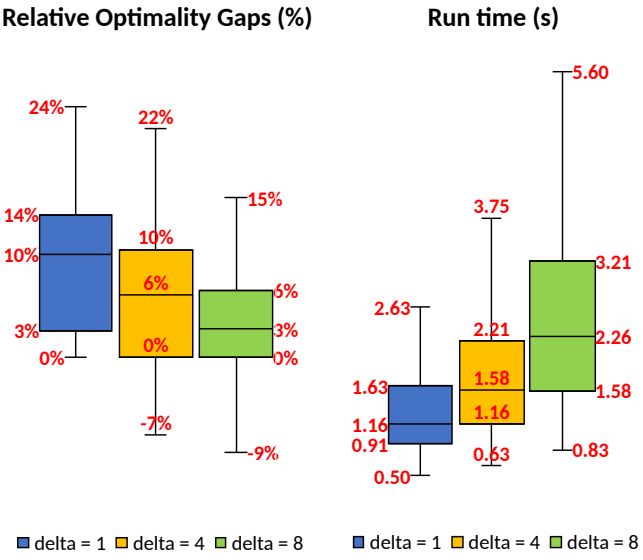


Figure 4.3: *General performance of the FBI approach on PSPLIB-120 instances. Performance is measured by relative optimality gap μ_{OPT} and run time μ_{CPU}.*

Comparing these result to CRA performance results of Fig. 4.1, we can make two observations.

Observation 8. *When using both detailed and aggregated planning for PSPLIB-120 instances, the FBI approach has on average a slightly better makespan performance than the CRA approach.*

Comparing Fig. 4.3 with the results from Fig. 4.1, we see that for all window widths $1, 4$ and 8, the quartiles of the makespan increase relative to the best known solution are lower for the FBI approach than the CRA approach.

Observation 9. *When using both detailed and aggregated planning for PSPLIB-120 instances, the run time of the FBI approach is on average more than a factor 2 lower than the run time of the CRA approach.*

Comparing Fig. 4.3 with the results from Fig. 4.1, we see that for all window widths $1, 4$ and 8, the quartiles of the run times of the FBI approach are significantly lower than the ones from the CRA approach. This is mainly a result of the fact that the CRA uses forward and backward passes in each iteration, whereas the FBI only uses one forward and one backward pass.

4.2 Conclusion & Future Work

We conclude this report by summarising our contributions and the results obtained from experimental analysis, see Sect. 4.2.1. In Sect. 4.2.2, we provide recommendations for future research related to our contributions.

4.2.1 Concluding Summary

In this thesis, we focused on resolving time and resource conflicts in manufacturing plans while minimising the makespan, which is a special case of the Manufacturing Planning Problem (MPP). We showed that the MPP can be reduced to the Resource Constrained Project Scheduling Problem (RCPSP). Motivated by practice, we distinguished detailed and aggregated planning, differing in the level of aggregation of the resource constraints. We introduced the Window Aggregated Resource Constrained Project Scheduling Problem (WARCPSP) as a relaxation of the standard RCPSP. This relaxation allows to model the MPP using both detailed and aggregated planning constraints. The WARCPSP uses a window configuration, which is a pre-defined set of time windows (intervals), in order to define resource feasibility. Instead of enforcing that at any time period the resource requirement is not more than the availability, the WARCPSP enforces that the total resource requirement in each window is not more than the total availability in that window. By changing the widths of the time windows, the WARCPSP supports a time-dependent level of aggregation, allowing detailed planning in the near future and more aggregated planning as time progresses.

Motivated by the complexity of resolving conflicts in manufacturing plans, we designed a heuristic Conflict Resolution Algorithm (CRA) that resolves conflicts in a given manufacturing plan while aiming to minimize the makespan. This algorithm is compatible with the WARCPSP, meaning conflicts are detected and resolved by taking into account different levels of aggregation, depending on whether detailed or aggregated planning is required. The CRA resolves time conflicts by using a special earliest-start scheduling procedure. Resource conflicts are resolved in an iterative fashion. Each iteration consists of a forward move that moves an activity forward based on an added precedence relation. This forward pass is followed by a backward pass, that tries to heuristically

improve the schedule by moving some activities backward.

By design, the CRA can be applied in an online setting, meaning that it can be applied to a manufacturing plan at any time and to specific parts only. Moreover, its heuristic nature means that the CRA is able to generate close-to-optimal conflictless schedules within a time amount that allows it to be used in a real-time setting by the human planner to quickly resolve potential resource conflicts and to evaluate the outcome of what-if scenario's.

In order to support claims about and analyse the performance of the CRA , we applied the CRA to WARCPSP instances that were derived from RCPSP instances of PSPLIB-120. More specifically, for each such an RCPSP instance, we transformed it to an instance of WARCPSP by adding a window configuration. We computed a corresponding (resource-infeasible) earliest start schedule that was used as input to the CRA, altogether referred to as the CRA approach. As mentioned in Obs. 1, when using detailed planning constraints, the median makespan performance over the 600 instances is within 13% of the best-known solution and is never off by more than 38% from the best-known solution in the most extreme case. The median run-time required to obtain those solutions is less than 5 seconds and is well below 17 seconds for all instances. When using aggregated planning constraints, we observed in Obs. 2 that the median makespan of schedules resulting from CRA approach decreased as the level of aggregation increased. In Obs. 3, we observed a positive correlation between the level of aggregation and the average run-time of the CRA. Additional experiments showed that the influence of Network Complexity, Resource Factor and Resource Strength, used to generate instances of PSPLIB, is in line with the results obtained from Kolisch et al. [27], see Obs. 5 - Obs. 7. Finally, we used an experiment to show that backward shifting activities does lead to improved scheduled, as was observed in Obs. 4.

Besides the design of a new algorithm, we looked at an existing post-processing technique called Forward-Backward Improvement (FBI). We slightly changed FBI in order to make it compatible with WARCPSP instances, i.e. aggregated planning. We applied the FBI algorithm to earliest-start schedules corresponding to instances of PSPLIB-120, referred to as the FBI approach. To our surprise, we observed in Obs. 8 that when comparing the CRA and FBI approach, the makespan performance of the FBI approach is generally slightly better than the makespan performance of the CRA approach. Similarly, Obs. 9 observes that the average run-time of the FBI approach is more than a factor 2 lower than the run-time of the CRA approach.

4.2.2 Future Work

In this thesis, we came across several ideas for future work. These ideas emerged both from assumptions that we had to make in order to limit the scope of this research as well as the observations of the experimental results. What follows are our main suggestions for future research in order to increase the significance of our results and the applicability of the algorithm that we designed.

Other objective functions

In our thesis, we focused on resolving conflicts while minimising the makespan. This is a reasonable objective function in a setting where the supply network under consideration represents a single project. In practice, however, supply networks often represent multiple projects such as different customer orders, whose underlying tasks are interrelated. For such supply networks, a suitable

objective would be to minimize (a sum of) the different makespans corresponding to the different projects, possibly in combination with customer priorities. In order to increase the applicability of the CRA approach, it is of interest to adapt the CRA so that it can be applied to more general supply networks.

Another important objective to consider is the robustness of CRA approach. The CRA approach is said to be robust when the differences between the original and resulting manufacturing plans are small on average. This is of importance as in practice, it is generally undesirable that resolving conflicts leads to a entirely different manufacturing plan, especially when such an approach is applied frequently. This leads to a high unpredictability of the shop-floor tasks to be executed in the near future. A first challenge is to define a proper measure that captures the robustness of the CRA approach. Once such a measure is defined, the next challenge would be to analyse the performance of the CRA approach in terms of robustness and adjust the CRA if needed. One concrete research direction would be to consider priority rules that incorporate the idea of robustness.

Resolving resource conflicts using a more global procedure

The CRA makes greedy, local decisions in forward shifting activities in order to resolve resource conflicts. Each such local decision influences the succeeding local decisions that can be made. It would therefore be of interest to think of a procedure that makes more global decisions, aiming to obtain a better makespan performance. This challenge is in particular challenging as at the same time it should be prevented that the run-time increases significantly.

Exploiting the window architecture in resolving resource conflicts

Resource conflicts are resolved by adding precedences, which was motivated by the order based representation results from Sect. 2.2.2 and the notion of breaking minimal forbidden sets from Sect. 2.2.2. These notions, however, are derived in the context of the RCPSP that uses detailed resource constraints, whereas the WARCPSP uses aggregated resource constraints. This observation served as motivation for backward shifting activities, see Sect. 3.3.7. Nevertheless, our way of forward shifting activities does not explicitly exploit the fact that resource feasibility is related to the resource requirement in time windows. In Lemma 3.4.2, we showed that by adding precedences, the window resource requirement will *eventually* be less than or equal to the window resource availability, but not immediately. Since the window resource requirement and availability are known, it is known how much activity units need to be shifted outside of the window in order to resolve a resource conflict. Consequently, an idea would be to focus on sets of activities to be shifted outside of the window such that the window resource requirement and availability precisely match.

Run-time improvement of the CRA

During the design and implementation of the CRA, we did consider its run-time, but our main focus was its makespan performance. In Obs. 1, we observed that for PSPLIB-120 instances, the median run-time of the CRA approach is in the order of seconds. In Obs. 3, we observed that the run-time increases as the width of the windows increases. We believe that by using a more clever implementation (using both different coding and a different coding language than Python), the run-time can be reduced, increasing the practical applicability of the CRA.

Applying the CRA to practical instances

We analysed the performance of the CRA by applying it to synthetic instances of the PSPLIB library. Due to lack of time and availability, we did not use a real practical instance from the manufacturing industries. In order to be able to generalise performance conclusions to a practical manufacturing planning environment, we suggest the CRA to be tested on such practical instances.

Forward-Backward Improvement

As was mentioned in Alg. 7 and observed in Sect. 4.1.6, the FBI approach generally leads to better results than the CRA approach. This was much to our surprise, as we did not expect that such a relatively simple algorithm, with only one forward and backward pass, would outperform an algorithm that uses intermediate forward and backward shifting of activities. We suggest further research in order to get a theoretical understanding of the FBI algorithm in relation to the WARCPSP, so that its characteristics can be used to improve the CRA.

Moreover, as noticed in Sect. 3.5, the FBI does not have the same online characteristics as the CRA. We believe that the FBI can be adapted so that it becomes applicable in an online setting. One simple adaptation would be to add the notion of lower bounds on the earliest start values of activities.

Comparison with different approaches

Our final suggestion for future work research is the comparison of the CRA with other existing approaches in literature that might be used to resolve resource conflicts. In order to come to a fair comparison, these approaches should have the same characteristics as the CRA, i.e. they should allow a time-dependent level of resource aggregation, they should be able to produce close-to-optimal solutions in a short amount of time and they should be applicable in an online context.

Bibliography

[1] Christian Artigues. "The Resource-Constrained Project Scheduling Problem". In: *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. Chap. 1.

[2] Christian Artigues, Philippe Michelon, and Stéphane Reusser. "Insertion techniques for static and dynamic resource-constrained project scheduling". In: *European Journal of Operational Research* 149.2 (2003), pp. 249–267.

[3] Egon Balas. "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm". In: *Operations research* 17.6 (1969), pp. 941–957.

[4] Egon Balas. *Project Scheduling With Resource Constraints*. Tech. rep. Carnegie-Mellon University Pittsburgh Pennsylvania Management Sciences Research Group, 1968.

[5] Martin Bartusch, Rolf H Möhring, and Franz J Radermacher. "Scheduling project networks with resource constraints and time windows". In: *Annals of operations Research* 16.1 (1988), pp. 199–240.

[6] Colin E Bell and Jaemin Han. "A new heuristic solution method in resource-constrained project scheduling". In: *Naval Research Logistics (NRL)* 38.3 (1991), pp. 315–331.

[7] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete applied mathematics* 5.1 (1983), pp. 11–24.

[8] Fayer F Boctor. "Some efficient multi-heuristic procedures for resource-constrained project scheduling". In: *European journal of operational research* 49.1 (1990), pp. 3–13.

[9] Peter Brucker et al. "A branch and bound algorithm for the resource-constrained project scheduling problem". In: *European journal of operational research* 107.2 (1998), pp. 272–288.

[10] Nicos Christofides, Ramon Alvarez-Valdés, and José M Tamarit. "Project scheduling with resource constraints: A branch and bound approach". In: *European Journal of Operational Research* 29.3 (1987), pp. 262–273.

[11] Edward W Davis and James H Patterson. "A comparison of heuristic and optimum solutions in resource-constrained project scheduling". In: *Management science* 21.8 (1975), pp. 944–955.

[12] F. De Nijs. "Project Scheduling: The Impact of Instance Structure on Heuristic Performance". MA thesis. The Netherlands: Technical University Delft, 2013.

[13] Uriel Feige and Joe Kilian. "Zero knowledge and the chromatic number". In: *Journal of Computer and System Sciences* 57.2 (1998), pp. 187–199.

[14] Birger Franck, Klaus Neumann, and Christoph Schwindt. "Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling". In: *OR-Spektrum* 23.3 (2001), pp. 297–324.

[15] Sönke Hartmann and Dirk Briskorn. "An updated survey of variants and extensions of the resource-constrained project scheduling problem". In: *European Journal of operational research* 297.1 (2022), pp. 1–14.

[16] Thomas Joel Russell Johnson. "An algorithm for the resource constrained project scheduling problem". PhD thesis. Massachusetts Institute of Technology, 1967.

[17] Arthur B Kahn. "Topological sorting of large networks". In: *Communications of the ACM* 5.11 (1962), pp. 558–562.

[18] J.E. Kelley. "The critical-path method: resource planning and scheduling". In: *Industrial scheduling* (1963), pp. 347–365.

[19] James E Kelley Jr. "Critical-path planning and scheduling: Mathematical basis". In: *Operations research* 9.3 (1961), pp. 296–320.

[20] Rainer Kolisch. "Efficient priority rules for the resource-constrained project scheduling problem". In: *Journal of Operations Management* 14.3 (1996), pp. 179–192.

[21] Rainer Kolisch. "Integration of assembly and fabrication for make-to-order production". In: *International Journal of Production Economics* 68.3 (2000), pp. 287–306.

[22] Rainer Kolisch. *Make-to-order assembly management*. Springer Science & Business Media, 2000.

[23] Rainer Kolisch. "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation". In: *European Journal of Operational Research* 90.2 (1996), pp. 320–333.

[24] Rainer Kolisch and Sönke Hartmann. "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis". In: *Project scheduling*. Springer, 1999, pp. 147–178.

[25] Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. "Benchmark instances for project scheduling problems". In: *Project scheduling*. Springer, 1999, pp. 197–212.

[26] Rainer Kolisch and Arno Sprecher. "PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program". In: *European journal of operational research* 96.1 (1997), pp. 205–216.

[27] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. "Characterization and generation of a general class of resource-constrained project scheduling problems". In: *Management science* 41.10 (1995), pp. 1693–1703.

[28] Haitao Li and Mehdi Amini. "A hybrid optimisation approach to configure a supply chain for new product diffusion: a case study of multiple-sourcing strategy". In: *International Journal of Production Research* 50.11 (2012), pp. 3152–3171.

[29] Haitao Li and Keith Womer. "Optimizing the supply chain configuration for make-to-order manufacturing". In: *European Journal of Operational Research* 221.1 (2012), pp. 118–128.

[30] SW McCarthy and KD Barber. "Medium to short term finite capacity scheduling: a planning methodology for capacity constrained workshops". In: *Engineering Costs and Production Economics* 19.1-3 (1990), pp. 189–199.

[31] Pierre-Antoine Morin, Christian Artigues, and Alain Haït. "Periodically aggregated resource-constrained project scheduling problem". In: *European Journal of Industrial Engineering* 11.6 (2017), pp. 792–817.

[32] Ramon Alvarez-Valdes Olaguibel and JoseManuel Tamarit Goerlich. "The project scheduling polyhedron: Dimension, facets and lifting theorems". In: *European Journal of Operational Research* 67.2 (1993), pp. 204–220.

[33] Ramón Alvarez-Valdés Olaguíbel and José Manuel Tamarit Goerlich. "Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis". In: *Advances in project scheduling* (1989), pp. 113–134.

[34] Michael L Pinedo. *Scheduling*. Vol. 29. Springer, 2012.

[35] George W Plossl and Joseph Orlicky. *Orlicky's material requirements planning*. McGraw-Hill Professional, 1994.

[36] A. Alan B. Pritsker and Lawrence J. Watters. *A Zero-One Programming Approach to Scheduling with Limited Resources*. Santa Monica, CA: RAND Corporation, 1968.

[37] Louis Richard Shaffer et al. *The critical-path method*. McGraw-Hill, 1965.

[38] A Sprecher. "A competitive branch-and-bound algorithm for the simple assembly line balancing problem". In: *International Journal of Production Research* 37.8 (1999), pp. 1787–1816.

[39] Hartmut Stadtler. "Supply chain management and advanced planning—-basics, overview and challenges". In: *European journal of operational research* 163.3 (2005), pp. 575–588.

[40] Frederik Stork and Marc Uetz. "On the generation of circuits and minimal forbidden sets". In: *Mathematical programming* 102.1 (2005), pp. 185–203.

[41] Marc Uetz. *Algorithms for deterministic and stochastic scheduling*. Cuvillier Verlag, 2002.

[42] V Valls, MA Perez, and MS Quintanilla. *Heuristic performance in large resource-constrained projects*. Tech. rep. Working Paper. Department D'Estadistica I Invecigacio Operativa, Universitat de Valencia, 1992.

[43] Mario Vanhoucke and José Coelho. "A tool to test and validate algorithms for the resource-constrained project scheduling problem". In: *Computers & Industrial Engineering* 118 (2018), pp. 251–265.

[44] JC Wortman et al. "A review of capacity planning techniques within standard software packages". In: *Production Planning & Control* 7.2 (1996), pp. 117–128.