Roman Čapek

Scheduling with Alternative Process Plans

August 2015

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Control Engineering

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Scheduling with Alternative Process Plans

Doctoral Thesis

Roman Čapek

Prague, August 2015

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics

Supervisor: prof. Dr. Ing. Zdeněk Hanzálek
Supervisor specialists: Ing. Přemysl Šůcha, PhD.

This thesis is dedicated to my family, with true love.

Declaration

This doctoral thesis is submitted in partial fulfillment of the requirements for the degree of doctor (Ph.D.). The work submitted in this dissertation is the result of my own investigation, except where otherwise stated. I declare that I worked out this thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis. Moreover I declare that it has not already been accepted for any degree and is also not being concurrently submitted for any other degree.

Czech Technical University in Prague
Prague, August 2015 *Roman Čapek*

Acknowledgments

First of all, I would like to give my great thanks to my thesis advisers prof. Dr. Ing. Zdeněk Hanzálek and Ing. Přemysl Šůcha, PhD. for their expert guidance and unselfish support throughout my PhD. studies.

The kindest thanks of all belong to my family members for their great support and undying patience during the whole studies.

I am grateful to all my colleagues and friends from our research group for a friendly atmosphere and positive attitude during our collective work.

Last but not least I would like to express my thanks to all people from our department who were helping me and other students to successfully go through all the studies every day.

Thanks to everyone who has helped me during the PhD. studies.

This work was supported by the Ministry of Education of the Czech Republic under the Projects OE09004, ME10039 and Research Program MSM6840770038. This work was also supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994 and by the Technology Agency of the Czech Republic under the Centre for Applied Cybernetics TE01020197.

Czech Technical University in Prague
Prague, August 2015 *Roman Čapek*

Abstract

This thesis is dedicated to the design of practical and efficient models and algorithms for the production processes. The key addressed issue are the alternative process plans, supporting much more flexible definition of the scheduling problems.

With respect to the state of the art in the scheduling area, this thesis aims to cover the gap for the solution approaches that incorporate both the selection of process plan and the fine scheduling within a single model. Consequently, there are two main goals of the thesis - first, to propose a suitable mathematical model capable to cover standard scheduling problems together with the definition of the alternative process plans and second, to design, implement and evaluate algorithms for three different problems with alternative process plans, emerging from real production processes.

The mathematical model for the considered scheduling problems is based on the well known Resource Constrained Project Scheduling Problem (RCPSP) which is combined with the formalism of Nested Temporal Networks With Alternatives (NTNA). Such a model reflects the typical structure of the production processes and it keeps most of the assumptions and constraints from the powerful RCPSP framework. The proposed model involves renewable resources with non-unary capacity, sequence dependent setup times, release times and deadlines of activities and generalized temporal constraints. Thus, it allows very general and flexible definition of the scheduling problems with many realistic constraints.

Three different scheduling problems with alternative process plans are then in more detail. The first studied problem involves negative time-lags and the goal is to minimize the total schedule length. For such a problem we have developed constructive heuristic algorithm where the activities are being scheduled and un-scheduled according to their dynamic priorities. The second studied problem is motivated by the production processes where the goal is to utilize the expensive machines as much as possible and, therefore, the time spent by setting up such machines is minimized. The solution approach is based on the iterative method with separation of a schedule into time-disjunctive parts where the local search is applied. The criterion for the third considered problem is the minimization of the total production cost consisting of the costs corresponding to the selected production operations (activities) and penalization for late jobs. Two different evolutionary based heuristic algorithms are used to solve such a problem and their results are thoroughly compared in extensive performance evaluation.

Since there are no standard benchmarks for the proposed scheduling model, we have used the new generated instances specific for each problem under study as well as the existing instances for the similar problems from the literature. Although the available instances usually cover only a part of our approach, all the designed algorithms showed very good performance. In most cases, the results were competitive or even better when compared to the algorithms designed to the specific sub-problems of the general concept used in this thesis.

Abstrakt

Tato disertační práce se věnuje návrhu a implementaci efektivních modelů a algoritmů pro řešení praktických problémů z oblasti optimalizace výrobních procesů. Pozornost je věnována hlavně problematice alternativních výrobních postupů, které s sebou přinášejí možnost velmi flexibilního zadání pro rozvrhovací úlohy.

S ohledem na analýzu souvisejících prací v oblasti kombinatorické optimalizace je cílem této práce rozšířit portfolio existujících přístupů o řešení, které spojuje výběr konkrétního výrobního postupu a samotné rozvrhování vybraných operací v jednom společném modelu. Práce se tak věnuje především dvěma propojeným tematům - zaprvé vytvoření vhodného modelu pro zvolený typ rozvrhovacích úloh a zadruhé návrhu, implementaci a testování optimalizačních algoritmů pro řešení rozvrhovácích problémů s alternativami, které jsou motivovány realnými výrobními procesy.

Matematický model pro uvažované problémy vychází z notace Resource Constrained Project Scheduling Problem (RCPSP), která je dále rozšířena o definici alternativních výrobních postupů s využitím formalismu Nested Temporal Networks With Alternatives (NTNA). Navržený model odpovídá typické struktuře výrobních procesů a přitom zachovává většinu předpokladů a omezení pro RCPSP. Model zahrnuje obnovitelné zdroje s libovolnou diskrétní kapacitou, přestavbové časy a zobecněná temporální omezení (minimální a maximální časové intervaly mezi začátky operací v rozvrhu). Díky tomu umožňuje navržený model velmi flexibilní přístup k definici úloh pro rozvrhování výrobních procesů s mnoha často používanými omezeními.

Práce se dále věnuje detailně třem různým rozvrhovacím úlohám s alternativními výrobními postupy. První úloha zahrnuje kladné a záporné hrany mezi operacemi, kritériem je minimalizace délky celého rozvrhu. Pro tento problém byla vytvořena konstruktivní heuristika, ve které jsou jednotlivé operace přidávány a odebírány z rozvrhu na základě dynamických priorit. Motivací pro druhou úlohu jsou výrobní procesy, ve kterých hrají zásadní roli drahé stroje, u nichž je potřeba minimalizovat finančně nákladné přestavby. Řešení je v tomto případě založeno na iterativní heuristice, která využívá lokální optimalizaci pro časově disjunktní části rozvrhu. Kritériem ve třetí úloze je minimalizace celkových nákladů spojených s výrobním plánem. Ty jsou dány zaprvé cenou samotných výrobních operací a za druhé penalizací za pozdě dokončené zakázky. Pro řešení jsou vytvořeny dva odlišné evoluční algoritmy, jejichž výkonnost je důkladně porovnána velkým množstvím testů.

Jelikož je model uvažovaný v této práci inovativní a prozatím neexistují žádná stadardizovaná testovací data, bylo pro všechny testy použito dvou zdrojů dat - zaprvé nově vygenerované instance pro každou uvažovanou rozvrhovací úlohu a zadruhé instance podobných problémů z literatury. Přestože tyto instance reflektují jen určitou část námi uvažované problematiky, všechny vytvořené algoritmy ukazují velmi dobrou výkonnost. Ve většině případů jsou jejich výsledky srovnatelné nebo i lepší než výsledky uváděné v literatuře.

Contents

List of Figures

List of Tables

Nomenclature

List of Variables and Constants

n	number of activities
$0, n+1$	dummy activities that start and end the project
m	number of resource types
a, b, i, j, k	general indices
$\mathcal{A}$	set of activities
A^S	set of selected activities
$\mathcal{R}$	set of resource types
R_k	resource type
θ_k	capacity of resource type R_k
p_i	processing time
r_i	release time
$\tilde{d}_i$	deadline
d_i	duedate
c_i	processing cost
w_i	weight
T_i	tardiness
R_i^k	demand of activity i for the resource type R_k
l_{ij}	general temporal constraint (time-lag)
st_{ij}	sequence dependent setup time
in_i	input label of node in graph
out_i	input label of node in graph
v_i	binary decision variable that determines the presence of the activity in the schedule
s_i	start time
z_{iqk}	binary decision variable that determines whether activity i is assigned to a resource unit q of resource type R_k

z_{iu}	binary decision variable that determines whether activity i is assigned to a resource unit under serialized index u
u_{ij}	length of the longest path between activities i and j
x_{ij}	binary decision variable of the ILP model
y_{ij}	binary decision variable of the ILP model
x_{ijk}	binary decision variable of the ILP model
y_{ijk}	binary decision variable of the ILP model
g_{ijk}	binary decision variable of the ILP model
f_{ij}	binary decision variable of the ILP model
UB	high positive constant
C_{max}	schedule length
G	general label for graph
V	set of vertices (nodes)
E	set of edges (arcs)
G_S	graph consisting of selected activities
G^{temp}	complete graph with temporal constraints
G^{prec}	graph with precedence-based temporal constraints
δ_i^+	out-degree of a node
δ_i^-	in-degree of a node
$P_{[i,j]}$	directed path from node i to node j
$\mathcal{B}_{i,j}$	branching of a graph delimited by nodes i and j
B_k	branch of a graph
$pred\,(i)$	set of direct predecessors of node i
$succ\,(i)$	set of direct successors of node i
$\mathcal{M}$	set of all pairs of activities with potential resource conflict
$nestedAlt$	classification for the problems with alternative process plans in a nested form
ST_{SD}	classification for the sequence dependent setup times
gpr	classification for the generalized temporal constraints
$temp$	classification for the generalized temporal constraints
l_{ij}^{min}	classification for the minimal (positive) time-lags
min	classification for the minimal (positive) time-lags
$\alpha\vert\beta\vert\gamma$	standard classification for the scheduling problems
S	schedule
C_{max}^{LB}	lower bound for the schedule length
C_{max}^{UB}	upper bound for the schedule length
$C_{max}^{current}$	current threshold for the schedule length

s_i^{LB}	lower bound for the activity start time
s_i^{UB}	upper bound for the activity start time
$\widehat{r}_i$	auxiliary variable for the release time constraint
$\widehat{d}_i$	auxiliary variable for the deadline constraint
$flex_i$	flexibility of the activity for scheduling
s_i^{temp}	earliest start time with respect to temporal constraints
s_i^{res}	earliest start time with respect to resource constraints
$\mathcal{W}$	time window in the schedule
$time_{LB}$	left border of the time window
$time_{RB}$	right border of the time window
$priority_i$	current priority of activity
$nAdds_i$	number of scheduling attempts for the activity
ψ_k	resource sequence for resource type k
top_i	topological order of activity
$scheduled_i$	denotes whether the activity is currently scheduled
t	time
t_{cpu}	time used by CPU for computation
$feas$	percentage ratio of feasible solutions found by the algorithm
RF	release time factor
DF	deadline factor
RC	resource constrainedness
$\#res$	number of resource types
TOS	total order strength
$\#AB$	number of alternative branchings
$\#APP$	number of alternative process plans
$PPAct$	average number of activities per process plan
AOS	average order strength
NL	maximal level of nested alternative branching
AAS	average activity slack
$O(N)$	linear complexity

Abbreviations

List of Abbreviations

PS	project scheduling
RCPSP	Resource Constrained Project Scheduling Problem
MRCPSP	multi-mode RCPSP
DTCTP	discrete time/cost trade-off problem
APP	alternative process plans
RCPSP-APP	Resource Constrained Project Scheduling Problem with alternative process plans
TNA	Temporal Network with Alternatives
NTNA	Nested Temporal Network with Alternatives
IPPS	integrated process planning and scheduling
CPU	central processing unit
RAM	random access memory
ALT	denotes alternative branching
PAR	denotes parallel branching
AoN	Activity-on-Node network
AoA	Activity-on-Arc network
NP	Non-deterministic Polynomial-time
IRSA	Iterative Resource Scheduling with Alternatives
STOAL	Setup Time Optimization ALgorithm
CP	constraint programming
ILP	integer linear programming
MILP	mixed-integer linear programming
AJSP	job shop scheduling problem with processing alternatives
GA	genetic algorithm
ScS	scatter search
DDE	discrete differential evolution

TABU	tabu search algorithm
TST	total setup time
TPC	total production cost
TWT	total weighted tardiness
SGS	serial generation scheme
RIS	referenced insertion scheme

Goals and Objectives

This thesis is dedicated to the scheduling problems where some parts of the considered processes can be performed in more, **alternative**, ways. Three problems based on the same model are studied and both the exact solution and the heuristic algorithm are developed for each of them. The problems differ in the considered constraints as well as in the objective function, which determines the goal of the optimisation. The goals of this thesis were set as follows:

1. Propose a common representation for the scheduling problems that include alternative processes.
2. For each studied problem, establish a mathematical formulation using the proposed representation.
3. Develop an algorithm to solve large instances for each of the problems.
4. Compare the proposed solution methods with the similar works from the literature.
5. Propose the methodology for evaluation and comparison of different solution approaches.

Chapter 1

Introduction

In the recent years, manufacturing (and other) processes are becoming more and more complex and flexible. On one hand, the companies need to utilize expensive production resources - machines, workforce, additional tools - to make the production effective in terms of their expenses. On the second hand, they need to satisfy all the demands of the current market, which is very dynamic. Therefore, it is very beneficial, if not even necessary, to define the process plans as precisely as possible and then let an artificial system to resolve which operations and in which order will be performed according to the resource environment and the given jobs.

This thesis is focused on the scheduling problems, for which there is a high flexibility in the problem definition. The most of the work is dedicated to the scheduling problems with the alternative process plans, especially the problems related to the production processes. In case of the alternative process plans, the operations can be performed in various ways, using fully automated machines, semi-automated machines or performed manually with some special equipment. Three related problems with the alternative process plans are addressed in detail.

The first studied problem is the scheduling of the wire harnesses production, which involves the alternative process plans, generalized temporal constraints (positive and negative time-lags) and sequence dependent setup times. The goal is to minimize the total schedule length. For the second studied problem, the goal is to maximize the utilization of the expensive machines in the production of electrical contacts and, therefore, the time spent by setting up such resources is minimized. The goal in the third studied problem, motivated by the production in the printing company, is to minimize the total production cost. In this case, the hard constraints (like deadlines) are substituted by the soft constraints that are reflected in the objective function.

1.1 Contribution and Outline

The main contribution of this thesis is the formulation of the novel scheduling problem, where the resource constrained project scheduling problem is extended by the definition of the alternative process plans - denoted as *RCPSP-APP*. Such alternative process plans specify the rules for the selection of activities, i.e. which activities will be present in the schedule and which will be not. Therefore, a new decision variable has to be established and, consequently, the search space is more complex. Although there were several attempts to incorporate the alternative choices into the scheduling process, there is no particular work dedicated to the general concept of the problem as considered in this thesis.

The proposed model encapsulates shared resources with an arbitrary discrete capacity, selection constraints defined via the alternative process plans, generalized temporal constraints among activities and sequence dependent setup times. For the considered problem, we propose a formal representation based on the existing approaches from the literature. Finally, the mathematical formulation of the common constraints is formulated as the mixed integer linear programming model.

The second contribution is the consideration of three different specific problems based on the proposed RCPSP-APP problem. Each particular problem is motivated by the different needs of the production companies and, therefore, an objective function as well as several specific constraints and assumptions are adjusted separately. Therefore, the mathematical model is formulated for each specific problem separately as well.

The third contribution is represent by the heuristic solution approach for each considered problem. The algorithms are designed with intention to solve the large instances for which the exact methods are not able to find the solution of the desired quality in a reasonable time. For the first two problems, we have developed new constructive algorithms and for the third one, two different evolutionary algorithms are adapted. The proposed algorithms are evaluated on a variety of instances, including the new generated benchmarks as well as the existing datasets for the similar problems from the literature.

The fourth contribution lies in an evaluation metric for the characterisation of the instances and in the consequent methodology of testing different algorithms with respect to the proposed evaluation metric. The main focus is paid to the structural properties closely related to the definition of the alternative process plans. Such a metric enables one to distinguish between the effectiveness of the proposed algorithms for different types of instances, reflecting e.g. the ratio of the alternative/parallel parts or the tightness of the temporal constraints. The proposed evaluation metric is a necessary prerequisite for a fair comparison (not only) of the proposed heuristic algorithms, since the complexity of the instance is dependent on many factors, not only on the number of activities.

The main contributions of this thesis are, namely:

a) formulation of the novel scheduling problem with alternative process plans based on the RCPSP formalism,

b) consideration of three related problems with different criteria together with a mathematical formulation,

c) a heuristic algorithm for each considered problem designed for solving of the large scale instances,

d) an evaluation metric for instances and a testing methodology for the comparison of different solution approaches.

The thesis is organized as follows: Section 1.2 provides the literature overview related to the problems considered in this thesis. Chapter 2 is dedicated to the description of the general scheduling model, including the formal classification. The next three chapters contain the specific problems from the area of scheduling with alternative process plans. The RCPSP with alternative process plans and generalized temporal constraints is studied in Chapter 3, where the definition of the evaluation metric for the instances and the testing methodology for different solution approaches is included as a part of the computational experiments. Next, the total setup time minimization objective function is considered in Chapter 4 and finally, Chapter 5 is dedicated to the problem where the total production cost is being minimized. Chapter 6 concludes the work.

1.2 Related Work

To address the problem involving alternative ways how to select and assign activities (operations, tasks) to the schedule, several modeling approaches can be found in the literature. In the most cases, some type of special graph is used to model the presence of alternatives in the scheduling problem. To avoid any misunderstandings, let us assume that the notions *activity*, *operation* and *task* have the same meaning and the term *activity* we will be used in this thesis. Furthermore, to address the presence of alternatives in the scheduling, the term *alternative process plans* will be used in the rest of the work. The benefit of the alternative process plans definition for the production processes is shown in Usher (2003), where the need of having an effective solution methodology is appointed and emphasized by the experimental results.

Beck and Fox (2000) established the *Modified Temporal Graph* with so called *XorNodes*, *AndNodes* and *ActivityNodes* to model the possibility of choice among the alternative process plans that are interconnected via the aforementioned nodes. Each activity has a certain probability to be assigned (selected) into the final schedule and the authors proposed a propagation technique for the probability values through the graph with both the parallel parts (delimited by *AndNodes*) and the alternative parts (delimited by *XorNodes*).

Another approach to model the alternative process plans in scheduling, similar to the Modified Temporal Graph methodology, was presented by Barták (2004) and Barták and Čepek (2007, 2008). The authors used a special type of graph called *Temporal Network with Alternatives*, which is a directed acyclic graph where the nodes represent activities and the arcs correspond to temporal constraints. Logical constraints, which represent alternative process plans, are specified through the input and output labels of each node. If only the structure of the network is considered, i.e. temporal constraints are ignored, we obtain the *Parallel/Alternative Graph* (*P/A Graph*). Both Beck and Fox (2000) and Barták and Čepek (2007, 2008) focused on the representation of the alternative process plans, the construction of the schedule itself is not considered.

Kis (2003) studied a job shop scheduling problem with processing alternatives where the goal is to minimize the makespan. Each job is represented by a special graph consisting of two types of subgraphs - *and-subgraphs* and *or-subgraphs*, which are both composed by more routes. A *route* is a directed path from the first node to the last node of the subgraph. All routes have to be scheduled for each and-subgraph while exactly one route has to be selected for each or-subgraph.

Finally, Shao et al. (2009), Leung et al. (2010) and Li et al. (2010) dealt with the problem of integrated planning and scheduling (IPPS), which is close to the job shop problem with alternative process plans, since each job includes more alternative ways (process plans) to complete the product. The goal is to select a process plan for each job and to schedule job activities such that the schedule length is minimized. The IPPS problem was studied also in Moon et al. (2002) where the problem is extended by the unit loads of products and transportation times among the machines.

Capacho and Pastor (2006, 2008) and Capacho et al. (2009) studied an assembly line balancing problem with alternatives, where certain parts can be processed in several alternative modes and the goal is to balance the workload of the available resources.

1.2.1 Resource Constrained Project Scheduling Problem

The *resource constrained project scheduling problem* (*RCPSP*) is well known NP-hard (see Blazewicz et al. (1983)) problem, with many real applications. Several exact solution procedures have been proposed by Demeulemeester and Herroelen (1992), Mingozzi et al. (1998), Brucker et al. (1998) and Dorndorf et al. (2000). For larger problem instances, heuristic and metaheuristic solution procedures have been proposed, see e.g. an overview published by Kolisch and Hartmann (2006). Other overviews of the problem can be found in Icmeli et al. (1993), Özdamar and Ulusoy (1995), Blazewicz et al. (1996), Herroelen et al. (1998), Brucker et al. (1999a) and Kolisch and Padman (2001).

Herroelen et al. (1999) and Brucker et al. (1999a) summarized the notation of the RCPSP problems and their extensions using the well-known $\alpha|\beta|\gamma$ notation (Blazewicz et al. (1983)). Hartmann and Briskorn (2010) published an extensive survey with many various forms and extensions of the resource constrained project scheduling problem. The concept of activities, temporal constraints, resource constraints and objective functions are discussed and the state of the art literature is summarized.

1.2.2 Extensions of the RCPSP

One of the existing extensions of the RCPSP problem is the *multi-mode resource constrained project scheduling problem* (*MRCPSP*) where each activity can be executed in one of several alternative modes with different processing times and resource demands (see De Reyck and Herroelen (1999); Neumann et al. (2003)). Moreover, multi-mode problem includes also the definition of non-renewable resources in general case. The basic goal of the problem is to determine a mode and a start time for each activity, such that the total duration of the project is minimized.

De Reyck and Herroelen (1999) proposed a local search based methodology for MRCPSP with generalized precedence constraints with objective to minimize the project duration. Neumann et al. (2003) formulated a mathematical model and a general algorithm scheme for the MRCPSP problem. Deblaere et al. (2011) proposed an exact scheduling procedure based on the Branch & Bound algorithm and also proposed a tabu search heuristic for the MRCPSP with a criterion to minimize the project duration. The currently best known search procedure for the problem MRCPSP is the scatter search presented in Van Peteghem and Vanhoucke (2011). An overview of all the available metaheuristic solution procedures for this problem can be found in Van Peteghem and Vanhoucke (2014).

Salewski et al. (1997) considered the RCPSP with mode identity constraints, which is a generalization of the multi-mode case where the set of all jobs is partitioned into disjoint subsets while all activities forming one subset have to be processed in the same mode. Kuster et al. (2006) proposed the extended resource constrained project scheduling problem (*x-RCPSP*), which incorporates the concept of alternative activities. The authors prove that any multi-mode RCPSP can be formulated as an *x-RCPSP*, since each mode of an activity can be represented as an alternative with exactly one activity. The authors focused on the rescheduling problem, which is used for a comprehensive disruption management. Kellenbrink (2012) presented the RCPSP with a flexible project structure (*RCPSP-PS*), which is is a generalization of the RCPSP-APP in terms of the structure and logical constraints. The problem involves the non-renewable resources but there are no additional constraints like time-lags or setup times; considered objective function is the makespan.

Chapter 2

Scheduling Model

This chapter is dedicated to the description of the new proposed model for the scheduling with alternative process plans. The motivation for the research is, in the first place, the scheduling of the production processes which typically involve more than one way how to complete the product. Not only are the resource requirements different, but the processing times, precedence relations and also the number of activities in each process plan can differ in general as well.

The process plan defines a set of activities such that their execution leads to the completion of a product. Each process plan is formed by a set of disjunctive activities where no activity can be included more than once in a process plan. On the other hand, an activity can be included in an arbitrary number process plans. We use the term *alternative process plans* since there are more process plans in the studied problem while only one of them has to be executed. Hence the goal of the scheduling is to choose a subset of all activities that forms one process plan and schedule them according to the given criterion.

Traditional scheduling algorithms, according to Blazewicz et al. (1996); Brucker (2007), assume exactly given set of activities to be scheduled, i.e. only one process plan is defined. In this thesis, the traditional scheduling approach is extended by a definition of alternative process plans, i.e. the traditional time scheduling and the decision which process plan will be executed are both integrated into one problem.

The studied problem is formulated as an extension of the resource constrained project scheduling problem (RCPSP). Although the RCPSP is a well-studied problem, there were only a few attempts to include the alternatives into the scheduling process. However, the alternative process plans can be found as a natural part of the production processes and therefore we have decided to extend the RCPSP problem by the definition of the alternative process plans. The following sections provide a detailed description of the new proposed model for the resource constrained project scheduling problem with alternative process plans.

2.1 Overall Problem Statement

The general problem studied in this thesis is identified by the set of activities, the set of resources, the set of constraints and the objective function. For each of three specific problems in this thesis, there are some differences in the definition, which are described in more detail. This section provides a general overview of the common part of the proposed scheduling model. First, the overall description of the problem is provided, then the detailed definition of the nested temporal networks with alternatives, used for the problem representations, is stated. Subsequently, all the considered constraints are described, the objective function is discussed and finally, the classification of the studied problem is provided.

Let the production consist of n indivisible operations performed on the specified machines according to the process plan. Consequently, there is a set of $n+2$ non-preemptive activities $\mathcal{A} = \{0, \ldots n+1\}$ to be scheduled on a set of m resource types $\mathcal{R} = \{R_1 \ldots R_m\}$ where each resource type $R_k \in \mathcal{R}$ has a discrete capacity $\theta_k \geq 1$, i.e. θ_k resource units are available for resource type R_k. Each activity i is characterized by the processing time $p_i \geq 0$, the release time $r_i \geq 0$ and the resource demand $\mathrm{R}_i^k \geq 0$ for the resource type $R_k \in \mathcal{R}$. Only mono-resource activities are considered in this thesis, meaning that each activity demands exactly one resource, i.e. $\sum_{\forall R_k \in \mathcal{R}: \mathrm{R}_{ik} > 0} (1) = 1$ for all $i \in \{1 \ldots n\}$. The processing time of the activity specifies the time needed for its execution, which must be performed without preemption (interruption). Release time determines the earliest time where the activity can be scheduled. Activities 0 and $n+1$ with $p_0 = p_{n+1} = 0$ and $\mathrm{R}_0^k = \mathrm{R}_{n+1}^k = 0$ for all $R_k \in \mathcal{R}$ denote *dummy* activities such that activity 0 is a predecessor and activity $n+1$ is a successor of all other activities. Precedence relations together with the definition of alternative process plans are specified using an NTNA formalism (see Section 2.3).

In the rest of the thesis, the problem defined in the previous paragraph is addressed as the *resource constrained project scheduling problem with alternative process plans* (*RCPSP-APP*). Since all the activities demand for, at most, one resource type and there are no additional resources, the problem can be addressed also as the *machine scheduling problem* (see Rand (1977); Blazewicz et al. (1991)). Nonetheless, the term RCPSP-APP will be used in this thesis, since the scheduling model is designed with intention to establish a general notation for the problems with alternative process plans.

2.2 Goal of the Scheduling

The goal of the scheduling for the problem described in the previous section is to select a subset $\mathcal{A}^S \subseteq \mathcal{A}$ of all activities (i.e. one process plan) and then to schedule $\mathcal{A}^S$ to a given set of resources while the value of the objective function is minimized.

To represent the schedule, three types of variables are considered:

- $v_i \in \{0, 1\}$ - determines the presence of the activity in the schedule. If $v_i = 1$, then activity i is present in the schedule and it is called *selected* activity; if $v_i = 0$, then the activity i is not in the schedule and it is called *rejected* activity.
- $s_i \in \mathbb{N}^0$ - determines the start time of the activity in the schedule. If the activity is rejected ($v_i = 0$), then its start time is arbitrary and it has no significance.
- $z_{iqk} \in \{0, 1\}$ - determines whether activity i is assigned to a resource unit q of resource type R_k. As for the start time, if activity i is rejected, then $z_{iqk} = 0$.

The values of the of all the variables are mutually constrained by a set set of selection, temporal and resource constraints. The constraints for selection of activities are defined in Section 2.4, the temporal constraints are given in Section 2.5 and finally, the resource constraints are defined in Section 2.6. The objective function, as well some additional constraints, are defined for each considered problem separately.

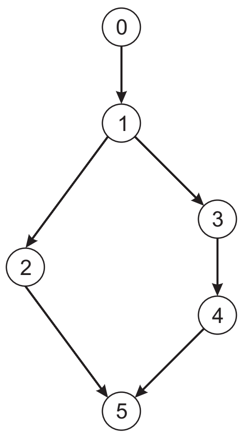
2.3 Problem Structure Representation

The structure of the scheduling problems is, in the most cases, represented by the *Activity-on-Node* (*AoN*) networks. The nodes of the AoN graph represent activities and edges represent precedence relations. On the contrary, in the representation using the *Activity-on-Arc* (*AoA*) networks, activities are represented by the arcs of the network and the nodes represent events, i.e. completion of some activities and, therefore, also the precedence relations. Any problem represented by the AoN (AoA) instance can be transformed to a problem represented by the AoA (AoN) instance, i.e. there is a mutual transformation for any problem. The similarities and differences of both approaches are summarized in Kolisch and Padman (2001). Both representations for the same instance are depicted in Figure 2.1.

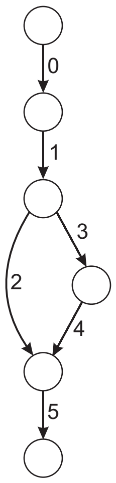
The representation of the problems studied in this thesis is based on the Activity-on-Node model, each activity therefore corresponds to one node. The classic AoN networks are designed for the problems, where all the underlying activities have to be scheduled and, therefore, an extended model have to be used in the case of the alternative process plans. The attempts to propose a modeling approach for the scheduling problems with alternative process plans have been made in Beck and Fox (2000), Kis (2003) and Chryssolouris et al. (1985), but the models are too restrictive and/or not enough general for the representation of the RCPSP-APP problem.

2.3.1 Nested Temporal Networks with Alternatives

Another approach to deal with the (not only) scheduling problems with alternative process plans in general has been proposed by Barták and Čepek (2007), who pro-



(a) AoN network



(b) AoA network

Fig. 2.1: Example of the AoN and AoA representation of the same instance

posed the *Temporal Networks with Alternatives* (*TNA*). The TNA is an acyclic directed graph where nodes represent activities and edges represent temporal constraints.

Each node i of the graph (corresponding to activity i) has an input label in_i and an output label out_i, denoting the type of input and output *branching*, which can be either *parallel* or *alternative*. If there is a parallel branching at the output (input) of node i, then $out_i = 0$ ($in_i = 0$) and, vice versa, if there is a alternative branching at the output (input) of node i, then $out_i = 1$ ($in_i = 1$). If activity i has only one predecessor (successor), then $in_i = 0$ ($out_i = 0$). Furthermore, each node i is assigned a binary value v_i, indicating whether the corresponding activity will be present in the schedule ($v_i = 1$) or not ($v_i = 0$).

According to Barták and Čepek (2007), the problem in assignment of v_i values for the TNA instance is NP-complete in case that some values are predefined. In other words, if some activities are selected a priory, then the decision whether there exists a feasible assignment of v_i values for the whole instance is NP-complete problem. The solution, motivated by the real processes, lies in the more restrictive form of the TNA called *Nested Temporal Networks with Alternatives* (*NTNA*). The assignment of the v_i values for the NTNA instances is proved to be a problem with polynomial complexity (see Barták and Čepek (2008)).

The NTNA is a special form of the TNA, where the parallel and alternative branchings are arbitrary nested one in another but no other interaction among the branchings is allowed. Let $G = \{V, E\}$ be a directed acyclic graph, where $V \equiv \mathcal{A}$ and $E = \left\{\forall (i,j) \in V^2 : i \text{ is a direct predecessor of } j\right\}$. Furthermore, let $V(G)$ be in a topological order, i.e. $i < j$ for all $(i,j) \in E(G)$ (see e.g. Korte and Vygen (2000)). Finally, let $a \in V(G)$ be a node with out-degree $\delta_a^+ > 1$ and $b \in V(G)$ be a node with out-degree $\delta_b^- > 1$. Then a connected component of G delimited by nodes a and b is called a *branching* if and only if all the following properties hold:

- $a < b$
- $a \in P_{[0,b]}$ for each directed path $P_{[0,b]}$ from node 0 to node b in G
- $b \in P_{[a,n+1]}$ for each directed path $P_{[a,n+1]}$ from a to $n+1$ in G
- $out_a = in_b$
- b is minimal for a given a

A branching in the NTNA instance, delimited by nodes a and b, is denoted as $\mathcal{B}_{a,b} = \left\{\forall i \in V(G) : \exists P_{[a,i]} \wedge \exists P_{[i,b]}\right\} \bigcup \{a, b\}$. If for each $a \in V(G) : \delta_a^+ > 1$ there is a corresponding $b \in V(G)$ such that $\mathcal{B}_{a,b}$ is a branching as defined in the previous paragraph, then graph G corresponds to the NTNA instance. Each branching consists of a set of branches $\mathcal{B}_{a,b} = \{B_1 \ldots B_{\delta+(a)}\}$, where B_k denotes the k-th *branch* of such a branching. Each branch $B_k \in \mathcal{B}_a$ is a set of activities that form a connected component of graph G, starting by some successor of node a and ending by the corresponding predecessor of node b.

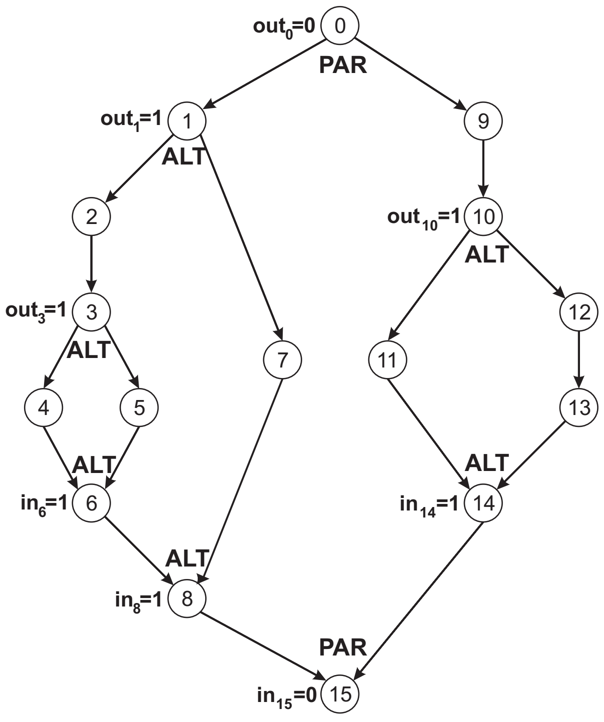


Fig. 2.2: Example of the NTNA instance

An example of the NTNA instance is depicted in Figure 2.2. For a better illustration, parallel branchings are denoted as *PAR* and alternative branchings are denoted as *ALT*. The meaning of the parallel branching is the same as for the scheduling problems without alternatives - after the first activity is scheduled, all the successors have to be scheduled as well. In case of the alternative branching only one successor has to be scheduled, i.e. only one path through the corresponding part of the graph has to be selected. With respect to the scheduling model, the constraints for the selection of activities for each branching $\mathcal{B}_{a,b}$ are:

- both activities a and b are selected/rejected simultaneously,
- all successors of a and predecessors of b in the parallel branching are selected/rejected simultaneously,
- only one successor of a and one predecessor of b are selected if a and b are selected in the alternative branching,
- all successors of a and predecessors of b are rejected if a and b are rejected in the alternative branching,

The constraints for the selection of activities (represented by v_i values) resulting from the NTNA instance are stated in Section 2.4. The presented NTNA formalism is used for the representation of the problem structure for all scheduling problems considered in this thesis.

2.4 Selection Constraints

The selection constraints determine which activities will be selected to be a part of the schedule and which will be rejected. The selection constraints are derived from the NTNA instance as follows:

1. When there is a parallel branching at the input/output of selected activity i ($in_i/out_i = 0$), all its direct predecessors/successors have to be selected. If activity i is rejected, all its direct predecessors/successors have to be rejected.
2. When there is an alternative branching at the input/output of selected activity i ($in_i/out_i = 1$), exactly one of its direct predecessors/successors has to be selected. If activity i is rejected, all its direct predecessors/successors have to be rejected.
3. When there is a simple precedence between activities i and j (i has only one successor j and vice versa j has only one predecessor i), both activities have to be selected/rejected simultaneously.
4. Dummy activities 0 and $n + 1$ have to be always scheduled.

The resulting constraints are:

$$v_i = \sum_{\forall j \in succ(i)} v_j \qquad \forall i \in \mathcal{A} : out_i = 1 \tag{2.1}$$

$$v_i = \sum_{\forall j \in pred(i)} v_j \qquad \forall i \in \mathcal{A} : in_i = 1 \tag{2.2}$$

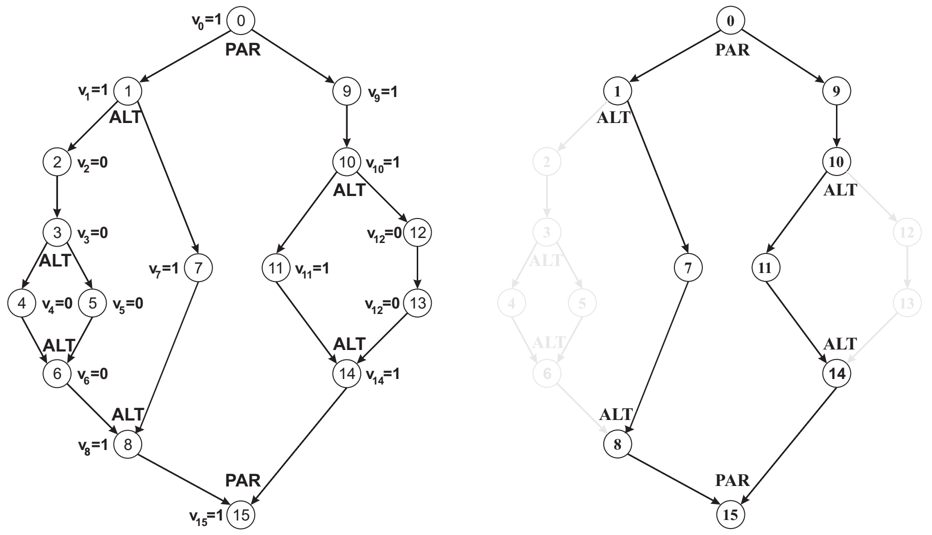
$$v_i = v_j \qquad \forall (i, j) \in \mathcal{A}^2 : out_i = 0 \wedge in_j = 0 \wedge j \in succ(i) \tag{2.3}$$

$$\sum_{i \in \mathcal{A}} v_i \geq 1 \tag{2.4}$$

where :

$succ(i) = \{\forall j \in \mathcal{A} : (i, j) \in E(G)\}, pred(i) = \{\forall j \in \mathcal{A} : (j, i) \in E(G)\}$

Any assignment of v_i values that fulfills the equations (2.1)-(2.4) corresponds to one process plan. The existence of a non-empty solution is always ensured thanks to the nested structure of the NTNA instance (see Barták and Čepek (2008)). The selection constraints are the same for all the scheduling problems considered in this thesis. A feasible assignment of v_i values and the resulting selection of activities corresponding to a feasible process plan is shown in Figure 2.3. The assignment of v_i values and the resulting process plan formed by selected activities is one of the six feasible process plans that can be found for the considered instance.



(a) Assignment of v_i values (b) Selected process plan

Fig. 2.3: Feasible assignment of v_i and the resulting process plan

2.5 Temporal Constraints

The formalism of the Nested Temporal Networks with Alternatives allows also to define the temporal constraints for all precedence-related nodes (activities). A pair of real numbers $[a_{i,j}, b_{i,j}]$ is assigned to each edge $(i,j) \in E(G)$, where $a_{i,j}$ is the minimal time distance of start times of activities i and j in the schedule and $b_{i,j}$ is the maximal time distance of the start times. In this thesis, the constraint for the minimal time distance $a_{i,j}$ is involved in each of the considered problems. More precisely, the concept of *general temporal constraints* in the form of *positive and negative time-lags*, is considered.

The positive and negative time-lags (shortly time-lags) are defined such that $s_i + l_{ij} \leq s_j$ for all $(i,j) \in \mathcal{A}^2$, where $l_{ij} \in \mathbb{R}$ is the length of the time-lag and s_i is the start time of activity i in the schedule. The constraints imposed by $a_{i,j}$ and $b_{i,j}$ in the NTNA instance form a special case of the time-lags, since they are restricted to the precedence-related activities only. In this thesis, we consider $l_{ij} \geq 0$ and $l_{ji} \leq 0$ for all $(i,j) \in E(G)$, i.e. the successor cannot never start before its predecessor in the NTNA instance. If there is no temporal constraint for a pair of activities $(i,j) \in \mathcal{A}^2$, then $l_{ij} = -\infty$.

Some specific assumptions with respect to the time-lags are given for each scheduling problem separately. The common assumption is that there is no cycle with a positive length in the time-lags definition for any process plan. For this purpose, let $\mathcal{A}^S \subseteq \mathcal{A}$ represent a subset of activities corresponding to a feasible process plan. Furthermore, let G^{temp} be a graph with nodes $V\left(G^{temp}\right) = \mathcal{A}^S$ and edges $E\left(G^{temp}\right) = \left\{\forall (i,j) \in V\left(G^{temp}\right) \times V\left(G^{temp}\right)\right\}$ where each edge $e_{ij} \in E\left(G^{temp}\right)$ has the weight (length) equal to l_{ij}. To detect the cycle with a positive length in G^{temp}, the longest paths are calculated by Floyd's algorithm (see e.g. Korte and Vygen (2000)). If G^{temp} contains any cycle with a positive length, then there will be at least one node for which the longest path to itself is greater then zero. On the contrary, if the graph does not contain any positive cycle, then the longest paths for each node to itself is equal or less then zero. We assume that there is no positive cycle for any $\mathcal{A}^S \subseteq \mathcal{A}$. In case that there was a positive cycle for a process plan, there would be no feasible solution for such a process plan.
The common temporal constraints for all the problems studied in this thesis are as follows:

$$s_i \geq r_i - UB \cdot v_i \qquad \forall i \in \mathcal{A} \tag{2.5}$$

$$s_j \geq s_i + l_{ij} - UB \cdot (2 - v_i - v_j) \qquad \forall (i,j) \in \mathcal{A}^2 \tag{2.6}$$

$$UB > \sum_{\forall i \in \mathcal{A}} \max\left(p_i + \max_{\forall j \in \mathcal{A}}(st_{ij}), \max_{\forall j \in \mathcal{A}}(l_{ij})\right) \tag{2.7}$$

Formula (2.5) ensures that no selected activity is scheduled before its release time. Thanks to the high positive constant UB, the release times for rejected activities are always satisfied. Formula (2.6) defines the positive time-lags for all pairs of selected activities. The positive time-lag is considered if and only if both activities are selected and, therefore, the rejected activities do not influence the schedule at all.

2.6 Resource Constraints

The resource constraints are based on the commonly used assumption that each resource unit can process, at most, one activity at the time. In other words, any pair of activities cannot overlap on the same resource unit in the same time. The second straightforward resource constraint is that each selected activity is assigned the appropriate number of resource units of the specified resource type. In other words, the resource demand of each selected activity is satisfied. The resource demands of the rejected activities need not satisfied, since such activities are not in the schedule at all.

In addition to the before mentioned assumptions, we consider the *sequence dependent setup times* (also called *changeover times*) that represent an additional time needed for setting up the resource between the consequently scheduled activities. A setup time $st_{ij} \geq 0$ determines the minimal time distance between the completion time of activity i and start time of activity j if i and j are scheduled consequently on the same resource type and they share at least one unit of such resource type. We presume that the setup times satisfy the triangular inequality $st_{ij} + st_{jk} \geq st_{ik}$ for all $\{i, j, k\} \in \mathcal{A}^3$ (Brucker, 2007).

The resource constraints, including the setup times, are then specified as follows:

$$\sum_{v=1}^{\theta_k} z_{ivk} = \mathrm{R}_i^k \cdot v_i \qquad \forall i \in \mathcal{V}, \forall k \in \mathcal{R} \tag{2.8}$$

$$z_{ivk} + z_{jvk} - 1 \leq 1 - y_{ij} \qquad \forall (i,j) \in \mathcal{M}, \forall k \in \mathcal{R}, \forall v \in \{1 \dots \theta_k\} \tag{2.9}$$

$$-x_{ij} + y_{ij} \leq 0 \qquad \forall (i,j) \in \mathcal{M} \tag{2.10}$$

$$s_i + p_i + st_{ij} \leq s_j + UB \cdot (1 - x_{ij} + y_{ij}) + UB \cdot (2 - v_i - v_j) \qquad \forall (i,j) \in \mathcal{M} \tag{2.11}$$

$$s_j + p_j + st_{ji} \leq s_i + UB \cdot (x_{ij} + y_{ij}) + UB \cdot (2 - v_i - v_j) \qquad \forall (i,j) \in \mathcal{M} \tag{2.12}$$

where :

$$x_{ij}, y_{ij} \in \{0,1\}; \mathcal{M} = \left\{(i,j) \in \mathcal{A}^2 : i < j \wedge \exists k : \mathrm{R}_i^k > 0 \wedge \mathrm{R}_j^k > 0\right\}$$

For the purpose of the resource constraints definition, two auxiliary variables have to be introduced. First, let x_{ij} be a binary decision variable such that $x_{ij} = 1$ if activity i is followed by activity j on the same resource type and $x_{ij} = 0$ otherwise. Second, let y_{ij} be a binary decision variable such that $y_{ij} = 1$ if activities i and j do not share any resource unit on any resource in the schedule and, therefore, the resource constraints need not to be considered and $y_{ij} = 0$ otherwise, i.e. there is at least one resource unit of the given resource type assigned to both i and j. Furthermore, let $\mathcal{M}$ be a set of all activity pairs for which there is a potential resource conflict between the two activities in such a pair.

Equation (2.8) ensures that the resource demand of each selected activity is satisfied while there are no assigned resource units for the rejected activities. Formula (2.9) determines the pairs of activities (values of y_{ij}) for which there might be a resource conflict, i.e. activities that share at least one resource unit of the same resource type. Formula (2.10) only determine the x_{ij} values for all pairs of activities for which there cannot be an actual resource conflict. Finally, the assumption of no-overlapping activities on the same resource unit in the same type is defined using a double inequality (2.11) and (2.12). For each pair of activities i and j competing for the same resource unit of the same resource type, either i precedes j ($s_i + p_i + st_{ij} \leq s_j$) or j precedes i ($s_j + p_j + st_{ji} \leq s_i$).

2.7 Classification of the problem

For the classification of the problem described in the previous text, we use the well known $\alpha|\beta|\gamma$ notation (see Blazewicz et al. (1996)), where α defines the resource environment, β stands for the specification of activities and additional constraints and γ defines the objective function. According to Brucker et al. (1999a), the above described problem can be classified as $PS|nestedAlt, temp, ST_{SD}|-$ where PS stands for the resource constrained project scheduling problem, $temp$ denotes the generalized temporal constraints and ST_{SD} represents the sequence dependent setup times. According to Herroelen et al. (1999), the problem can be specified as $m, 1|nestedAlt, gpr, s_{ij}|-$ where $m, 1$ defines the renewable resources with constant availability in time, gpr represents the generalized temporal constraints and s_{ij} denotes sequence dependent setup times.

For both classifications, the β field is extended by the term $nestedAlt$ to denote the presence of the alternative process plans in the nested form (see Section 2.3). In this thesis, we will address the considered problem as the *resource constrained project scheduling problem with alternative process plans - RCPSP-APP*. Since there are no additional resources, the problem can be addressed also as the *machine scheduling problem*, yet the term RCPSP-APP will be used throughout the thesis.

The objective function is not specified in the classification since the criterion is specified for each of the considered scheduling problems separately. In all the cases, the objective function is a convex function dependent on the selection of activities and their start times and order on the resources.

Chapter 3

RCPSP-APP with positive and negative time-lags

The motivation for the research is the scheduling of production processes which typically involve more than one way how to complete the product. Such alternative process plans occur in the production of wire harnesses, where operations to produce a wire harness can be performed in various ways, using fully automated machines, semi-automated machines or manually operated ones with special equipment. The problem can be formalized as an extension of the $PS|temp, ST_{SD}|C_{max}$ problem. Therefore, we deal with the resource constrained project scheduling problem which is further extended by the positive and negative time-lags, sequence dependent setup times and alternative process plans. Time-lags (also called generalized temporal constraints) are useful to specify the relative time position of two activities in general. Sequence dependent setup times serve to cover the time needed to change the equipment or set up a machine between two different operations. The optimality criterion is to minimize the schedule length. The combination of generalized temporal constraints and logical constraints (in form of alternative process plans) makes the problem even more difficult since we have to introduce new decision variables into the problem.

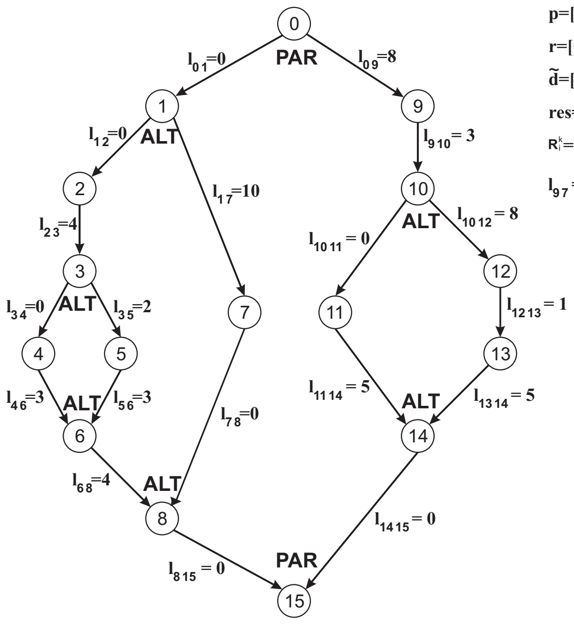
This chapter presents the resource constrained project scheduling problem with alternative process plans (RCPSP-APP) motivated by the real production of wire harnesses. Section 3.1 contains the statement and the mathematical model of the $PS|nestedAlt, temp, ST_{SD}|C_{max}$ problem with the representation based on the RCPSP-APP formalism. The model also considers sequence dependent setup times and generalized temporal constraints (positive and negative time-lags). A heuristic method, where the choice of process plan and traditional scheduling are executed simultaneously, is described in Section 3.2. Computational experiments together with the novel evaluation metric for the instances and the testing methodology for different algorithms are provided in Section 3.3. Section 3.4 concludes the work.

3.1 Problem Statement

The major part of the problem considered in this chapter is defined in the overall problem statement in Chapter 2. Therefore, the problem includes the set of activities $\mathcal{A}$, the set of resources $\mathcal{R}$ and the selection, temporal and resource constraints as defined in Chapter 2. In addition to this, each activity $i \in \mathcal{A}$ has a deadline $\tilde{d}_i \geq 0$, which is the hard constraint for the activity completion time in any schedule. The key part of the problem statement are the positive and negative time-lags that can be defined for any pair of activities.

To represent the problem structure, the formalism of the Nested Temporal Networks with Alternatives is used as described in Section 2.3.1. An example of the NTNA instance with all the necessary data for the scheduling problem is depicted in Figure 3.1. In addition to the temporal constraints imposed by the direct precedence relations, there are two further minimal time-lags $l_{9\,7} = 3$ and $l_{11\,14} = 6$ and one maximal time-lag $l_{9\,0} = -16$. The setup times are given for each resource separately; there are no setup times for resource R_3 since it is dedicated to dummy activities only.

The goal is to minimize the total schedule length, also called *makespan*, which is denoted as C_{max} and is equal to the completion time of the last activity in the schedule. The described problem can be classified as $PS|nestedAlt, temp, ST_{SD}|C_{max}$ or $m, 1|nestedAlt, gpr, s_{ij}|C_{max}$ using the same classification schemes as in Section 2.7. In the rest of the chapter, only the first notation will be used.



p=[0, 8, 2, 1, 3, 2, 1, 5, 6, 4, 3, 7, 4, 3, 6, 0]

r=[0, 0, 0, 6, 6, 6, 6, 5, 0, 0, 2, 5, 5, 5, 0, 0]

$\tilde{d}$=[5, 9, 9, 9, 9, 9, 9, 15, 25, 15, 21, 25, 23, 26, 30, 30]

res=[3, 1, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 1, 2, 3]

R_i^k=[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]

$l_{9\,7} = 3$ $l_{11\,14} = 6$ $l_{9\,0} = -16$

R_1	3	4	8	9	10	12	14
3	0	0	2	2	2	1	1
4	0	0	2	2	2	1	1
8	2	2	0	0	2	1	1
9	2	2	0	0	0	1	1
10	2	2	2	0	0	1	1
12	1	1	1	1	1	0	0
14	1	1	1	1	1	0	0

R_2	1	2	5	6	7	11	13
1	0	0	0	0	2	2	2
2	0	0	0	0	2	2	2
5	0	0	0	0	2	2	2
6	0	0	0	0	0	2	2
7	2	2	2	0	0	0	0
11	2	2	2	2	0	0	0
13	2	2	2	2	0	0	0

Fig. 3.1: Example of the NTNA instance

3.1.1 Mathematical Model

In this section, the mathematical model for the $PS|nestedAlt, temp, ST_{SD}|C_{max}$ problem is formulated. Except the constraints imposed by the deadlines of activities (3.1), all the formulas in the mathematical model below are taken over from Chapter 2. All the parameters, constants and variable domains are the same as in Chapter 2 as well. The objective is to minimize the makespan, which is defined as the maximal completion time over all the activities in the schedule.

$$\min\left(\max_{\forall i\in\mathcal{A}}(s_i+p_i)\right)$$

subject to:

$$(2.1) - (2.12)$$

$$s_i + p_i \leq \tilde{d}_i + UB \cdot (1 - v_i) \qquad \forall i \in \mathcal{A} \tag{3.1}$$

3.1.2 Problem Complexity

Let us focus on the complexity of the problem considered in this chapter. The problem $PS|temp, ST_{SD}|C_{max}$, i.e. the case without alternative process plans, is NP-hard since it is a generalization of the $1|r_j, \tilde{d}_j|C_{max}$ problem (see reduction of this problem from a 3-partition problem in Lenstra et al. (1977)). If the resource constraints are omitted, we have a $PS\infty|temp|C_{max}$ problem, which can be solved in polynomial time (e.g. using linear programming while eliminating the resource constraints). On the other hand, the problem $PS\infty|nestedAlt, temp|C_{max}$, is NP-hard, despite the resource constraints relaxation, see Čapek et al. (2012) for more details. This leads to the observation that the computation of the earliest start times for all activities $i \in \mathcal{A}$ is an NP-hard for the problem $PS|nestedAlt, temp, ST_{SD}|C_{max}$ since the $PS\infty|nestedAlt, temp|C_{max}$ problem is a sub-problem of finding the earliest start times for all activities.

3.2 Heuristic Algorithm

Since $PS|nestedAlt, temp, ST_{SD}|C_{max}$ is an NP-hard problem, the optimal solution can be obtained, in reasonable amount of time, only for small instances. For large instances, we propose a heuristic algorithm that does not ensure finding an optimal solution, but it is able to handle instances with a significantly larger amount of activities. The idea of this algorithm, called Iterative Resource Scheduling with Alternatives (*IRSA*), is based on an IRS algorithm for $PS|temp, st_{ij}|C_{max}$ inspired by software pipe-lining and presented by Rau (1994) and extended by Hanzálek and Šůcha (2009) who focused on the acyclic scheduling problem and introduced so called *take-give resources* into the problem. It is a constructive method where activities are being added to the schedule according to their actual priority or being removed if the partial schedule is not feasible. The input of the algorithm is an instance of the $PS|nestedAlt, temp, ST_{SD}|C_{max}$ problem. The output of the algorithm is a schedule S determined by the selected activities, their start times and assigned resource units, i.e. $S = [s, v, z]$. The main purpose of the proposed heuristic is to deal with the problems where a feasible schedule cannot be found in polynomial time in general case. The optimization of the C_{max} criterion is achieved by the gradual tightening of the constraint for the schedule length.

3.2.1 Initialization

The algorithm starts with the estimation of the bounds for the schedule length. The upper bound is computed as $C_{max}^{UB} = \sum\limits_{\forall i \in \mathcal{A}} max \left(p_i + \max\limits_{\forall j \in \mathcal{A}} (st_{ij}), \max\limits_{\forall j \in \mathcal{A}} (l_{ij}) \right)$ (see Brucker et al., 1999b). The lower bound is computed as $C_{max}^{LB} = s_{n+1}^{LB}$, i.e. the lower bound of the earliest start time of activity $n+1$. For this purpose, let G^{temp} be a directed graph with nodes $V(G^{temp}) = \mathcal{A}$ and edges $E\left(G^{temp}\right) = \left\{(i,j) \in V(G^{temp}) \times V(G^{temp}) : l_{ij} \neq -\infty\right\}$ with weights equal to l_{ij}. Furthermore, let G^{prec} be a directed graph with nodes $V(G^{prec}) = V(G^{temp})$ and $E\left(G^{prec}\right) = \{(i,j) \in E\left(G^{temp}\right) : i$ is a direct predecessor of j in the NTNA$\}$. Then the estimated C_{max}^{LB} is equal to the shortest path length between nodes 0 and $n+1$ in G^{prec} computed by Dijkstra's algorithm (Korte and Vygen, 2000).

In the original IRS algorithm, the priority of an activity is equal to its longest path length to the terminal activity $n+1$. Due to NP-hardness of the longest path lengths computation in our case, we use only the estimation retrieved from G^{temp}, i.e. negative time-lags are omitted. Moreover, we have to distinguish priorities according to alternative process plans. Therefore, the priority of an activity increases with its estimated distance to the end of the schedule and decreases with the length of the alternative branch in which the activity is included. To compute priorities, we first set $aprior_i = c_1 \cdot u_{i,n+1} - c_2 \cdot u_{open,close}$ for each activity i where $u_{i,j}$ is the longest path length between nodes i and j in G^{temp}, the $open$ and $close$ are activities that start and terminate the minimal alternative branch containing activity i and c_1 and c_2 are constants. Minimal alternative branch for activity i is the alternative branch (see Section 3.1) containing activity i such that there is no other alternative branch containing activity i with the lower number of activities. In the example in Figure 3.1, the $open$ and $close$ for activity 5 are activities 3 and 6 respectively. For activity 2, the $open$ and $close$ are activities 1 and 8. Based on the algorithm testing on various instances, the best performance is achieved when the longest path length to the end of the schedule is given higher influence on the priority value (we use $c_1/c_2 = 5/3$). Finally, the priority $priority_i$ of each activity i is set to a value equal to the position of its $aprior_i$ value in the ascending order of all $aprior$ values. In other words, activity with the lowest $aprior$ value will have priority equal to 1, next activity will have priority equal to 2 and the activity with the highest $aprior$ value will have priority equal to $n + 2$.

3.2.2 Main loop

In each iteration of the main loop, the function $findSchedule$ tries to find the schedule with the given upper bound while the number of steps is limited by the parameter $budget$ that is usually set as a number of activities multiplied by the parameter $budgetRatio$. If a feasible schedule is found, all activities are shifted to the left by the label-correcting algorithm (see Brucker and Kunst, 2006) so that the constraints and the order of activities in S are kept. A new upper bound of the schedule length is computed as $C_{max}^{UB} = C_{max}^{current} - 1$ and the next iteration of the loop is performed. If a feasible schedule S is not found for the given schedule length, the algorithm modifies the priority according to the returned partial schedule.

Algorithm 1 IRSA(budgetRatio, maxModifications, instance)

```
compute $C_{max}^{LB}$ and $C_{max}^{UB}$;
set initial $priorities$;
$budget = budgetRatio \cdot n$;
$actualModifications = 0$;
while $C_{max}^{UB} \geq C_{max}^{LB}$
    $S = findSchedule\left(C_{max}^{UB}, priority, budget\right)$;
    if $S$ is feasible
        $s = shiftLeft\,(S)$;
        $C_{max}^{UB} = s_{n+1} - 1$;
    else
        if $actualModifications < maxModifications$
            $priority = modifyPriority\,(priority, S)$;
            $actualModifications = actualModifications + 1$;
        else
            break;
        end
    end
end
```

A general observation for heuristic algorithms is that more incorrect decisions are made at the beginning and, therefore, the priority of the earliest scheduled activities and activities that have been added to the schedule more often is decreased. The function $findSchedule$ is then called for the same upper bound C_{max}^{UB} using the modified priorities. If the schedule was not found and the maximum number of priority modification steps determined by the parameter $maxModifications$ is exhausted, the algorithm returns the best schedule.

3.2.3 Inner loop

In the inner loop of the IRSA algorithm, priorities are updated in the function $updatePriority$ (see Algorithm 2) such that the priority is increased for the activities marked as selected and proportionally decreased to the number of inclusions of the activity into the schedule. This update of priorities allows the heuristic to switch between alternative branches instead of staying in the same selection for the whole run of the algorithm. For each activity i, the priority is updated such that $priority_i = priority_i + 0.5 \cdot v_i - 0.5 \cdot nAdds_i$ where $nAdds_i$ denotes the number of inclusions of activity i to the schedule. Activity k with the highest priority is

Algorithm 2 Inner loop of IRSA

```
$findSchedule\left(C_{max}^{UB}, priority, budget\right)$
    $scheduled = \{\}$ ;
    $nAdds_i = 0 \; \forall i \in \mathcal{A}$;
    $s_i = 0 \; \forall i \in \mathcal{A}$;
    $v_i = 0 \; \forall i \in \mathcal{A}$;
    while $budget \geq 0$
        $priority = updatePriority\,(priority, nAdds, v)$;
        $k = \max\limits_{\forall j \in \mathcal{A}: j \notin scheduled \wedge j \notin rejected} (priority_j)$ ;
        $s_k^{LB} = \max\limits_{\forall j \in scheduled} (s_j + l_{jk})$ ;
        $s_k^{UB} = C_{max}^{UB} - p_k$;
        $[conflicts, s_k] = findSlot\left(k, scheduled, s_k^{LB}, s_k^{UB}\right)$ ;
        $nAdds_k = nAdds_k + 1$;
        $[s, scheduled] = insertActivity\,(k, s_k, conflicts)$ ;
        $v = findSelected\,(v, scheduled)$ ;
        if schedule is complete
            return $S$;
        end
        $budget = budget - 1$;
    end
    return $S$;
end
```

found among the set of not yet scheduled activities and a time window $\langle s_k^{LB}, s_k^{UB} \rangle$ where activity k can be scheduled is computed. The lower bound for start time s_k^{LB} is calculated as the minimum time such that all temporal constraints $s_j + l_{jk} \leq s_k$ for all $j \in scheduled : l_{jk} \geq 0$ are satisfied, where $scheduled$ is a set of activities that forms the current partial schedule. The start time upper bound s_k^{UB} is set to the maximal value such that the activity is completed before the given C_{max}^{UB}.

The function $findSlot$ tries to find the earliest time slot within the given time window with respect to the resource constraints. In other words, the time interval given by s_k^{LB} and s_k^{UB} is explored while searching for a time point where the given activity can be scheduled without violating any resource constraint. Sequence dependent setup times are also considered. If no feasible time position is found, then the time slot is set to s_k^{LB} if the activity is being added to the schedule for the first time. If the activity has been already included into the schedule in previous step, its time slot is set to $s_k^{LB} + 1$ to avoid cycling of the algorithm. The function $findSlot$ then returns all conflicting activities, i.e. activities that cannot be kept in the schedule without violating any resource or temporal constraint with respect to the last included activity.

Activity k is then inserted into the partial schedule and all activities marked as conflicting are removed in order to keep the partial schedule feasible at any time. If an unscheduled activity (i.e. activity actually removed from the schedule) is a member of some alternative branch, then all activities in the same alternative branch are also removed. The list of the selected/rejected activities is then updated; the scheduled activities are marked as selected, activities belonging to the same alternative branch are also marked as selected activities and all activities that cannot be added to the schedule without violating propagation rules from the mathematical model are marked as rejected activities. The selection/rejection of other activities is not decided yet. If each activity is already scheduled or marked as rejected, then the schedule S is complete.

3.2.4 Example of the IRSA Algorithm Progress

Figure 3.2 illustrates one iteration of the IRSA main loop for the instance depicted in Figure 3.1, considering three resource types with capacity equal to one . In the initialization, the algorithm sets $priority = (16\ 15\ 10\ 9\ 7\ 8\ 6\ 11\ 14\ 13\ 12\ 5\ 3\ 2\ 4\ 1)$ and consequently it starts with the addition of activity 0 into the schedule. Then activity 1 is added to the schedule and its start time is set to its lower bound , i.e. $s_1 = 0$ (step 1 in Figure 3.2). Then activity 8 is scheduled and the next not yet scheduled activity with the highest priority is 9, which has to be scheduled to the same resource as activity 8. Its time window is given as $s_9^{LB} = 8$ and $s_9^{UB} = 16$, resulting from $l_{09} = 8$ and $l_{90} = -16$. Within the given time window, there is no space to schedule activity 9 without violation of resource constraints and therefore activity 8 is marked as conflicting in function $findSlot$ and then removed from the

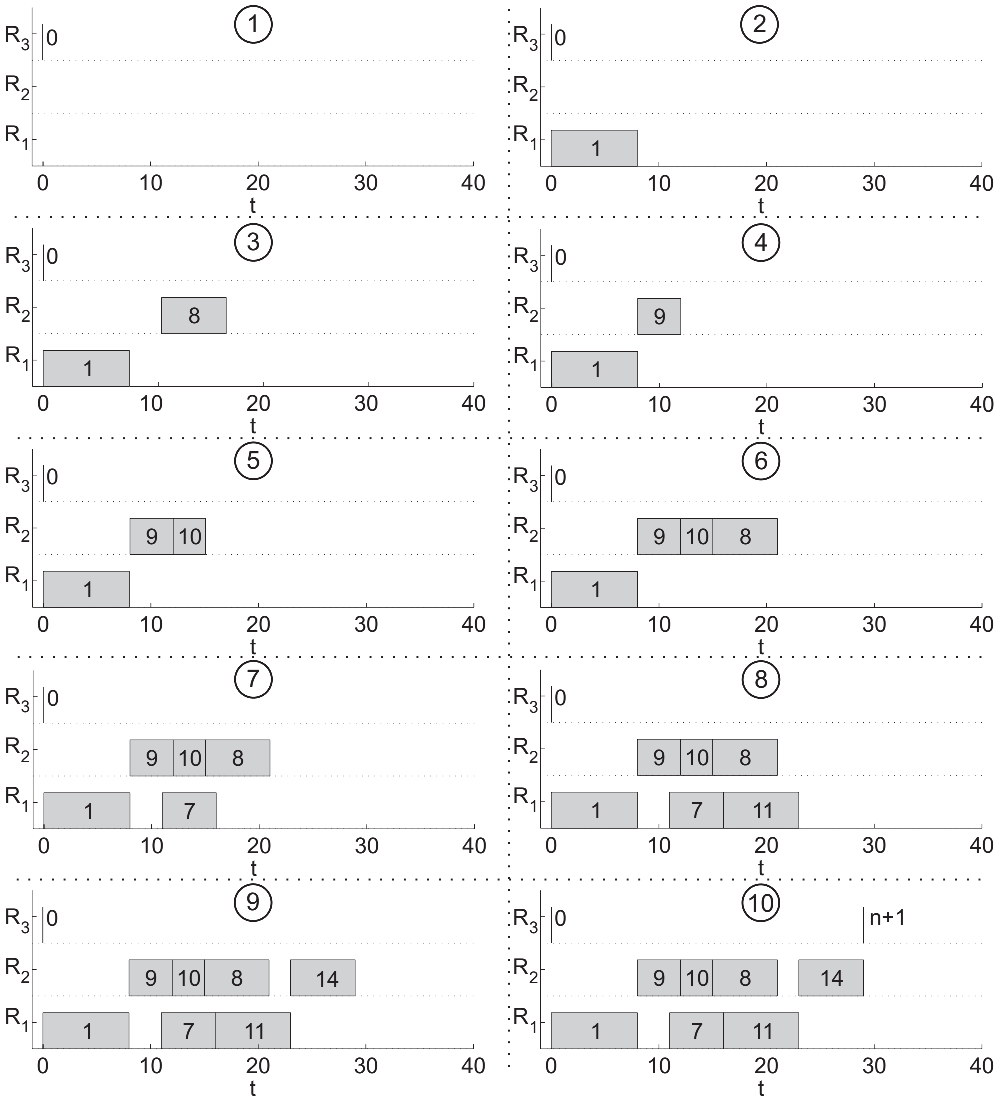


Fig. 3.2: Example of the IRSA algorithm progress

schedule in function $insertActivity$ (step 3). Activity 9 is scheduled instead and its start time is set to 8. In the following step, activity 10 is added and then activity 8 is added back to the schedule. Then the algorithm adds the activities one by one up to the last activity $n+1$ and the schedule is complete, since each activity is marked as scheduled or rejected.

3.3 Computational Experiments

This section presents extensive computational experiments for the problem defined in Section 3.1. First, the performance evaluation for the mathematical model proposed in Section 3.1.1 is shown. Two different solution approaches have been used for the mathematical model, namely the *mixed integer linear programming* (*MILP*) and the *constraint programming* (*CP*).

The optimal solutions obtained by both mathematical solvers are then used for the first evaluation of the IRSA algorithm. The heuristic algorithm is further evaluated on the instances of integrated process planning and scheduling (IPPS) problem from Shao et al. (2009), which is a specific sub-problem of the problem considered in this chapter. Furthermore, the instances of the job shop scheduling problem with processing alternatives from Kis (2003) are used to test slightly modified version of the IRSA algorithm.

Section 3.3.6 is dedicated to the description of the evaluation metric for the characterisation of the instances for the RCPSP-APP problem. The statistical methods used for the comparison of the different solution approaches and for the determination of the important instance properties are described in the same section. Finally, the results of the heuristic algorithm are compared with CP solver and discussed with respect to the proposed evaluation metric.

All experiments were performed on a PC with 2x Intel Core 2 Quad CPU at 2.83GHz with 8GB of RAM. The IRSA algorithm was implemented in C# language and the MILP and CP models have been developed and tested in the IBM ILOG CPLEX Optimization Studio 12.4.

3.3.1 Generated Instances

Up to our knowledge, there are no available standard benchmarks for the considered $PS|nestedAlt, temp, ST_{SD}|C_{max}$ problem. Therefore, randomly generated instances have been used to test both solvers of the mathematical model and the IRSA algorithm. The datasets with 30 (denoted as D30), 50 (D50), 100 (D100) and 200 (D200) activities per instance were generated, each dataset containing 100 random instances. The datasets are formed by very diverse instances, both the structure of the NTNA instance and the attributes of resources and activities are generated from the random distributions with wide range of values.

The number of resource types for each instance is randomly generated from interval $\langle 1, R_{max} \rangle$ where R_{max} is 2 for D30 activities, 3 for D50 activities and 5 for D100 and D200 activities. The capacity of each resource type is randomly generated from interval $\theta_k \in \langle 1, 5 \rangle$. Each setup time is randomly generated from the interval $\langle 5, 10 \rangle$.

The parameters of the activities were generated from the uniform distribution with the following boundaries: processing time $p_i \in \langle 1, 10 \rangle$ and resource demand $\mathrm{R}_i^k \in \langle 1, 3 \rangle$ for one resource type $R_k \in \mathcal{R}$. Release times and deadlines are generated based on the estimation of the schedule length and adjusted by the release time factor RF and the deadline factor DF, similarly to the method used in Vanhoucke et al. (2001). First, the minimal schedule length is estimated as a maximum from two numbers - the critical path length l_{CP} and the schedule length estimate based on the resources availability l_{RA}. Then the release times are generated from the interval $r_i \in \left\langle 0, \frac{max(l_{CP}, l_{RA})}{RF} \right\rangle$ and the deadlines from the interval $\tilde{d}_i \in \langle 0, DF \cdot max\,(l_{CP}, l_{RA}) \rangle$. Finally, the release times and deadlines are sorted in non-decreasing order and assigned to the activities based on the precedence relations from activity 0 towards activity $n+1$.

The critical path length l_{CP} is computed as the minimal schedule length while the resource constraints are relaxed, i.e. only the temporal constraints are considered. Since the considered problem remains NP-hard even if the resource constraints are not considered (see Section 3.1.2), only the non-negative time-lags are considered for the l_{CP} calculation. To calculate l_{RA}, the processing time of each activity is multiplied by its resource demand and the resulting values are summed up for each resource type separately. A schedule length estimate for each resource is then calculated as the total consumption demand over the capacity of a resource. Finally, the highest estimate over all resource types is considered, i.e. $l_{RA} = \max\limits_{\forall k \in \{1 \ldots m\}} \left(\sum\limits_{\forall i \in \mathcal{V}} \frac{p_i \cdot \mathrm{R}_i^k}{\theta_k} \right)$.

3.3.2 Mathematical Model Complexity

To handle the mathematical model formulated in Section 3.1.1, we use two different approaches, namely the mixed integer linear programming (MILP) and the constraint programming (CP). Solvers for both the MILP and the CP problem formulations have been developed and tested in the IBM ILOG CPLEX Optimization Studio 12.4. Both solvers were tested on four datasets D30, D50, D100 and D200, described in the previous section.

Two performance indicators were used in order to determine the respective effectiveness of the solvers. First, the ratio of the optimal solutions with respect to the assigned solution time has been observed, i.e. the number of instances solved to the optimum by the particular solver in a certain time limit is calculated for each dataset. Then the time limit is increased and the same instances are solved again; the total range of the time limit was set from 1 to 300 seconds. Second, the mean value of the

$Dataset$	D30		D50		D100		D200	
t_{CPU}	$MILP$	CP	$MILP$	CP	$MILP$	CP	$MILP$	CP
1	51	29	36	18	9	4	6	3
10	72	56	47	27	17	8	11	5
30	80	76	58	36	22	15	14	9
60	98	97	60	42	25	17	19	14
300	100	100	62	45	30	21	21	19

Table 3.1: The ratio of optimal solutions for the mathematical solvers

objective function over each dataset is observed for both solvers.

The ratio of the instances solved to the optimum depending on the given time limit for both solvers is depicted in Table 3.1 where row $Dataset$ denotes the set of instances, column t_{CPU} contains the time limits for both solvers in seconds and columns $MILP$ and CP denote the ratios of the instances solved to the optimum in percent. The mean objective values for both solvers with respect to the assigned time limit are then shown in Table 3.2 where the abbreviations has the same meaning and the values in columns $MILP$ and CP denote the mean value of the objective function over all instances with the same number of activities.

Although the results achieved by both solvers may seem similar and the dependency on the assigned time limit comparable at first sight, several important facts can be derived from the experiments. First, the ratio of instances solved to the optimum is always higher for the MILP solver. On the contrary, the mean value of the objective function is, in the most cases, lower (i.e. better) for the CP solver. The reason is in the different search strategies for both approaches. The MILP solver is based on a branch and bound method while the CP solver uses the restarted mechanism with the local search optimization. The same holds for the increase in the ratio of optimal solutions and for the mean objective value if the time limit is being increased.

We can conclude that both solvers represent a solution methodology providing very good results for instances with up to 50 activities. For the instances with more than 100 activities, the tested solvers do not represent approach useful for real applications where the response is needed in a short term.

3.3.3 Performance Evaluation of IRSA algorithm

The IRSA algorithm was evaluated using the same set of instances as for the MILP and CP solvers in the previous section. The parameters of the algorithm were set to $budgetRatio = 6$ and $maxModifications = 2$. Two performance measurements were used to test the effectiveness of the IRSA algorithm. First, we consider the number of the instances for which IRSA found a feasible solution and second, the mean difference of the IRSA algorithm from the optimal values is observed. The overall

$Dataset$	D30		D50		D100		D200	
t_{CPU}	$MILP$	CP	$MILP$	CP	$MILP$	CP	$MILP$	CP
1	148	145	242	237	510	501	1002	990
10	139	137	229	228	494	491	982	974
30	136	135	226	223	480	475	973	969
60	133	132	222	220	465	460	968	962
300	132	132	221	218	459	455	965	958

Table 3.2: The mean objective values for the mathematical solvers

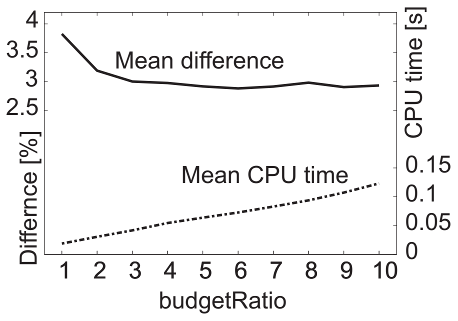
$Dataset$	D30	D50	D100	D200
$feas$ [%]	97	98	95.1	96.8
$diff$ [%]	2.12	2.45	4.24	6.51

Table 3.3: Overall performance evaluation of the IRSA algorithm

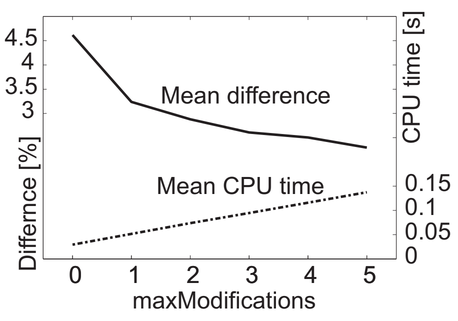
results are summarized in Table 3.3 where $Dataset$ denotes the set of instances, $feas$ stand for the ratio of the feasible solutions and $diff$ denotes the mean difference of the IRSA results from the optimal values for a particular dataset.

Due to the complexity of the mathematical model of the considered problem, the optimal can be found only for a subset of all instances in datasets D100 and D200 in a reasonable time (4 hours time limit). Therefore, the IRSA algorithm is evaluated only on the instances for which the optimal solution has been found using the MILP and CP solvers. The total number of feasible solutions is 82 for the dataset D100 and 64 for the dataset D200.

In conjunction to the overall performance evaluation of the IRSA algorithm, we have tested the influence of the algorithm settings on the quality of the results and on



(a) Budget



(b) Modifications

Fig. 3.3: Performance evaluation of IRSA algorithm and MILP model

n	10	50	100	250	500	1000	2000
t [s]	0.01	0.06	0.12	0.31	1.07	2.55	5.72

Table 3.4: Solving time for IRSA algorithm

the respective solution time. The influence of the $budgetRatio$ parameter is illustrated in Figure 3.3(a) where the mean difference from the optimum and the solution time are depicted in dependence on the given budget for the algorithm. Figure 3.3(b) shows the influence of the $maxModifications$ parameter. The tests were performed using the D100 dataset as proposed in Section 3.3.1.

Finally, we have evaluated the average running time of the IRSA algorithm for the instances with a wide range of the number of activities. The mean solving time for the IRSA algorithm with regard to the number of activities is shown in Table 3.4. For each number of activities, 20 feasible instances with were generated. The parameters of the algorithm were the same as in previous paragraphs, i.e. $budgetRatio = 6$ and $maxModifications = 2$.

3.3.4 Integrated process planning and scheduling

The Integrated process planning and scheduling (IPPS) problem studied in Shao et al. (2009) is used to prove the effectiveness of our algorithms for the scheduling problems containing alternatives. IPPS is again a subproblem of the problem considered in this chapter. The goal is to select and schedule a subset of all activities based on the precedence graph containing alternative routes and alternative machine assignment such that the makespan is minimized.

Instance	1	2	3	4	5	6	7
Shao et al. (2009)	116	116	95	93	116	116	162
IRSA	117	119	98	93	119	117	171

Table 3.5: Comparison of IRSA algorithm with Shao et al. (2009)

In Shao et al. (2009) there are six small instances (1-6) of IPPS and one bigger instance (7) obtained by joining all small instances into one graph. The comparison of the reported objective values and the values obtained by the IRSA algorithm for all seven instances is depicted in Table 3.5. It should be appointed out that the objective value for the first instance indicated in Shao et al. (2009) is not possible, since the optimal value is 117 instead of 116. The mean solution time reported in Shao et al. (2009) is 1 second for small instances, while for the bigger one there is no solution time at all. The algorithm was coded in C++ language and run on a machine with 2.40 GHz Pentium IV. The mean running times for the IRSA algorithms is 12 ms for small instances and 2s for the bigger one.

As can be seen from Table 3.5, the IRSA algorithm is competitive with the evolutionary algorithms proposed in Shao et al. (2009). Therefore we can conclude that the solution methodology is eligible to solve the problems with alternative process plans.

3.3.5 Evaluation on AJSP instances

Finally, we have evaluated the IRSA algorithm on the instances of the *job-shop scheduling problem with processing alternatives* (AJSP) proposed by Kis (2003). We have decided to solve such instances since our problem is the generalized version of the AJSP problem. The results are depicted in Table 3.6 where columns GA, TABU and RAND contain the results found by algorithms proposed by Kis (2003) and column IRSA contains the results found by the IRSA algorithm, *diff* is the ratio of the schedule length found by the given algorithm over the lower bound estimated by the MILP solver and *t* is the average computational time in seconds.

As we can see, the results found by the algorithms proposed by Kis (especially TABU algorithm) are superior than the results found by the IRSA algorithm. On the other hand, the increase in the computational time in dependence on the number of activities is more crucial for algorithms proposed by Kis. The total computational time for each instance is also much lower in case of the IRSA algorithm, although the comparison is not straightforward since Kis (2003) reported that C++ language was used and the tests were performed on a machine with Pentium II 400 MHz.

	GA		TABU		RND		IRSA	
Instances	*diff*	*t [s]*	*diff*	*t [s]*	*diff*	*t [s]*	*diff*	*t [s]*
a01-a03	1.025	3.812	1.021	2.331	1.023	3.308	1.062	0.07
a04-a06	1.042	17.04	1.011	11.38	1.024	16.78	1.096	0.16
a07-a09	1.042	40.52	1.012	30.76	1.095	42.98	1.077	0.39
a10-a12	1.042	78.67	1.005	67.68	1.093	87.04	1.137	0.62
a13-a15	1.020	27.55	1.014	71.29	1.098	29.92	1.251	0.14
a16-a18	1.051	67.57	1.012	49.27	1.135	77.41	1.263	0.27
a19-a21	1.068	124.8	1.015	97.14	1.149	153.1	1.235	0.67
a22-a24	1.072	60.31	1.042	43.23	1.136	72.02	1.299	0.31
a25-a27	1.123	147.8	1.058	131.4	1.203	191.6	1.364	1.11
a28-a30	1.145	274.3	1.025	274.1	1.212	386.3	1.259	1.02
a31-a33	1.152	100.5	1.083	82.76	1.249	130.8	1.341	0.85
a34-a36	1.157	243.6	1.060	253.8	1.261	347.7	1.381	1.93
a37-a39	1.151	457.7	1.036	327.6	1.232	709.5	1.258	2.42

Table 3.6: Comparison of IRSA algorithm with Kis (2003)

The problem assumed in this chapter is more general than the problem described by Kis. The main difference is that positive time-lags are restricted to be equal to processing times of activities and there are no negative time-lags at all in the AJSP instances. We also assume more general definition of alternative process plans where the alternative and parallel branchings can be arbitrary nested one in another. Furthermore, we do not focus on the particular situation where activities are joined in jobs with the specific precedence relations and resource assignment. Finally, there are no sequence dependent setup times in the AJSP problem.

To solve the AJSP instances, we have slightly modified function $findSlot$ in the IRSA algorithm. Each job in the AJSP problem is a sequence of activities where at most one activity can be in process at each time but the order of activities in and-subgraphs is not specified. Therefore, the function $findSlot$ has to check one more constraint during the search for the feasible time position, i.e. the feasible time position of an activity has to satisfy three type of constraints - temporal constraints, resource constraints and job constraints.

3.3.6 Evaluation Metric for Instances

In this section, the evaluation metric for the characterisation of the instances for the problem described in Section 2.1 is proposed. The instances can be described by the resource availability, parameters of activities and structure properties of the corresponding Nested Temporal Network with Alternatives (NTNA) instance. Each property for each instance is represented by a numerical value that is later used for the comparison of different solution approaches.

3.3.6.1 Resource environment

For the evaluation of the generated instances, we use two properties with respect to the resource environment for each instance. First, the **Number of Resource Types** ($\#res$) $\#res$ is the number of resources types for the particular instance, equal to m according to the problem statement in Section 2.1. Second, the **Resources Constrainedness** (RC) is a measure related to the average consumption of resources by activities over all resource types, used e.g. in Demeulemeester et al. (2003). The value of the resource constrainedness for resource type R_k is calculated as $RC_k = \frac{1}{\theta_k} \cdot \frac{1}{n} \cdot \sum_{\forall i \in \mathcal{A}} R_i^k$, i.e. it is an average demand of all activities that require resource type R_k over its capacity. The overall resource constrainedness is then given as an average over all resources $RC = \frac{1}{m} \cdot \sum_{\forall k \in \{1 \dots m\}} RC_k$. A higher number indicates a more resource constrained problem where activities have to be ordered more sequentially on resources than for a lower RC value.

3.3.6.2 Structural properties

The structure of the NTNA instance is generated based on many input factors like the minimum and maximum number of branches in both parallel and alternative branchings, the ratio of the alternative branchings, etc. For the computational experiments, we propose a set of structural properties that can be measured for each generated instance:

The **Total Order Strength** (TOS) is the order strength of the NTNA instance regardless of the types of branchings. The order strength of a directed acyclic graph with n nodes is calculated as the actual number of edges in the transitive closure of the graph over the maximal number of edges, i.e. $TOS = 2 \cdot |E_{closure}| / n \cdot (n-1)$. The transitive closure of a graph is a graph, where nodes i and j are connected by the edge if and only if there is a directed path from i to j in the original graph. An instance with a higher value of the order strength is usually easier to solve since the order of more activities is given in advance. For the instance in Figure 3.1, the value is calculated as $TOS = 2 \cdot 64/(16 \cdot 15) = 0.53$.
The **Number of Alternative Branchings** ($\#AB$) is the actual number of alternative branchings in an instance, i.e. it is equal to the number of nodes in NTNA that have the alternative output label. In the example in Figure 3.1, there are 3 alternative branchings.
The **Number of Alternative Process Plans** ($\#APP$) is the total number of selection-feasible process plans that can be derived for a particular instance. In other words, it is a total number of unique combinations of selected activities that will satisfy the rules for parallel and alternative branchings. The higher number of process plans, the larger the solution space, since there are more possibilities how to create a schedule. There are 6 different process plans that can be found in the example in Figure 3.1.
The **Average Order Strength** (AOS) is computed in a similar way as TOS, but instead of calculating one value of the order strength for the whole instance, AOS represents the average value computed over all process plans separately. To calculate the order strength of a process plan, a subgraph induced by the activities selected in the process plans is taken into account. As can be seen from the experiments in Section 3.3.8, the AOS value is, in most cases, slightly higher than the TOS value. On the other hand, a counter-example where AOS is lower than TOS can be also found. The TOS value for the example in Figure 3.1 is an average value over 6 process plans, resulting in $AOS = 0.62$.
The **Average Number of Activities per Process Plan** ($PPAct$) is given as an average number of selected activities over all possible process plans. Generally, the higher number the more difficult instances, since the resulting RCPSP problem contains more activities. The average number of activities per process plan in Figure 3.1 is $PPAct = 67/6 = 11.17$.
The **Maximal Level of Nested Alternative Branching** (NL) is the highest level of nesting with respect to alternative branchings. For the instances without alternatives, the value of NL is zero. For the instance depicted in Figure 3.1, the value of NL is 2 since the alternative branching determined by activities 3 and 6 is nested in another one.

3.3.6.3 Attributes of activities

For each generated instance, the **Average Activity Slack** (AAS) is calculated as a difference between the latest and the earliest start time of an activity in the schedule. The earliest start time est_i is derived from the release time of activity i and the precedence relations among the activities from the first activity towards the last one. Similarly, the latest start time lst_i is derived from the deadline of activity i and the precedence relations from the last activity towards the first one. The activity slack is then calculated as $AS_i = lst_i - est_i$ and the average activity slack is then given as $AAS = \frac{1}{n} \cdot \sum_{\forall i \in \mathcal{A}} AS_i$.

3.3.7 Evaluation Methodology

The metric composed of TOS, AOS, $\#AB$, $\#APP$, $PPAct$, NL, $\#res$, RC and AAS introduced in the previous paragraphs is used to characterize all instances and find out which type of problem structure is better to solve by which specific algorithm. In other words, our intention is to determine whether each property plays an important role if a different solution approaches are applied. To find out the most important properties that, we first separate the instances into three sets for each dataset - instances where the CP algorithm was able to find a better solution, instances where the IRSA algorithm was better and instances where the objective value was the same for both. Then we use Two-sample t-test for equal means (see Snedecor and Cochran (1989)) to distinguish between statistically important factors and factors that do not have an important influence on the solution quality.

The two-sample t-test for equal means (t-test for short) is a statistical method how to determine whether two sets with the same distribution have the same mean value at a given significance level. In our case, we want to determine whether there is an important relationship between the values of a specific measured property and the solution quality of two different solution approaches. For this purpose, the t-test is used for two sets of instances for each dataset - instances where IRSA found better results and instances where CP is better. The output of the t-test is the so called p-*value* which represents the significance that the mean values of both compared sets are the same. A higher p-value corresponds to a higher significance that the mean values are identical and, vice versa, a low number means that the mean values are different. If the mean values are different, then we can conclude that the tested property has an important influence on the effectiveness of the different solution approaches.

The assumption for the t-test is either a normal distribution or at least sufficient amount of data (at least 30 elements), which is satisfied in our case, since we generate 100 instances per dataset. The calculations within the t-test not only consider the mean values, but also the standard deviation of the values. The results we are looking for are those where the p-value is low, then the mean values are different and, therefore, the property significantly influences the effectiveness of the given approaches.

3.3.8 Experiments for Evaluation Metric

The evaluation metric for the characterization of the instances and the corresponding evaluation methodology proposed in the previous text are tested using the IRSA algorithm and the CP solver. In addition to the datasets defined in Section 3.3.1, two new datasets with 50 activities per instance were generated, differing in the assignment of release times and deadlines. Dataset D50L contains *loose* instances where the release times and deadlines do not form a difficult constraint for finding a feasible solution. On the contrary, release times and deadlines for dataset D50T are very *tight*, which is reflected in the number of feasible solutions found. Release times and deadlines for the former dataset D50 represent the intermediate step between the former two. The rationale to generate new datasets is to examine the dependency of the solution approaches on the temporal constraints in more detail, since they directly influence the ratio of the feasible solutions as well as the value of the objective function.

The settings (i.e. the solving time) for the CP solver were adjusted such that the results are comparable with the results achieved by the IRSA algorithm in the terms of the objective values. Thus, the impact of the properties of the instances can be evaluated for the similar quality of the results. Furthermore, two different search limits of the CP solver and the IRSA heuristic were used for the datasets D100 and D200. Therefore, datasets denoted as D100a and D100b in the following tables contains the results for the same instances, only the settings of the algorithms were changed. The fail limit for the CP solver was increased from 5000 (D100a) to 10000 (D100b) and the parameters of the IRSA algorithm were set to $budgetRatio = 6$ and $maxModifications = 2$ (D100a) and to $budgetRatio = 12$ and $maxModifications = 4$ (D100b). The same situation is then for D200a and D200b, where the same dataset D200 is used.

The overall results of both the CP approach and the IRSA heuristic are shown in Table 3.7. Column *feasible* denotes the number of the feasible solutions (out of 100),

	Constraint Programming			IRSA		
Dataset	$feasible$	$best$	t_{cpu} $[ms]$	$feasible$	$best$	t_{cpu} $[ms]$
D30	100	36	967	97	37	18
D50L	100	33	1121	100	35	49
D50	100	36	1095	98	32	60
D50T	84	5	3066	79	6	59
D100a	79	31	2026	78	34	86
D100b	83	40	4699	79	8	180
D200a	54	12	5969	62	28	199
D200b	61	29	10085	62	7	372

Table 3.7: Results for new datasets

D30	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.46	0.46	1.17	4.7	26.9	0.82	1.26	0.87	229.13
IRSA	0.48	0.46	1.14	5.56	27.61	0.77	1.38	0.68	227.41
p-value	0.51	0.95	0.82	0.55	0.05	0.46	0.01	0	0.59

D50L	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.41	0.41	1.89	11.96	43.72	1.09	1.2	0.9	675.3
IRSA	0.47	0.46	2.11	13.09	41.76	1.19	1.34	0.67	662.31
p-value	0.01	0.01	0.14	0.34	0.01	0.19	0.2	0	0

D50	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.45	0.45	1.83	8.31	43.81	1.05	1.28	0.85	167.82
IRSA	0.46	0.46	2.09	15.87	41.19	1.08	1.29	0.67	164.52
p-value	0.5	0.8	0.1	0	0	0.64	0.77	0	0.37

D50T	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.37	0.42	2.41	18.59	36.68	1.36	1.68	0.87	57.82
IRSA	0.41	0.47	2.75	24.11	32.07	1.75	1.43	0.64	52.71
p-value	0.54	0.39	0.39	0.37	0.09	0.28	0.08	0	0.56

D100a	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.42	0.44	3.48	50.58	84.7	1.5	1.93	0.81	1701.2
IRSA	0.46	0.47	3.88	116.74	81.63	1.52	2.1	0.78	1632.8
p-value	0.22	0.36	0.12	0.01	0.12	0.86	0.17	0.09	0.56

D100b	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.41	0.41	3.64	97.62	83.76	1.48	2.02	0.81	1743.4
IRSA	0.43	0.42	3.82	77.97	83.94	1.53	2.09	0.67	1524.2
p-value	0.52	0.71	0.62	0.39	0.94	0.66	0.69	0	0.2

D200a	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.37	0.37	8.34	1002.3	160.34	2.24	2.13	0.85	2099.9
IRSA	0.39	0.4	8.15	1594.2	159.95	2.18	2.43	0.74	2340.5
p-value	0.47	0.37	0.67	0.21	0.92	0.58	0.2	0	0.09

D200b	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
CP	0.36	0.37	8.05	1485.2	161.24	2.1	2.25	0.8	2200.7
IRSA	0.42	0.42	8.63	2186.6	161.42	2.37	2.2	0.72	2278.1
p-value	0.19	0.21	0.29	0.29	0.97	0.25	0.82	0	0.67

Table 3.8: Evaluation of properties for new datasets

column *best* denotes the number of the solutions where the corresponding solution method found strictly better solution than the other one. Finally, column t_{cpu} shows the computation time in milliseconds. Generally, the IRSA heuristic is faster but it does not significantly improve the solutions with the increasing solution time. The constraint programming solver needs a longer time to obtain the same quality of the solutions, but then, with increasing time limit, it is still able to considerably improve the solutions. The prove can be found in comparison of results for D100a vs D100b and D200a vs D200b.
The evaluation of the properties importance is shown in Table 3.8. For each tested property, we can derive the following conclusion based on the obtained results:

- The total order strength (TOS) and the average order strength (AOS) are closely related in both the values and the influence for the solution approach. The importance of both properties is rather low, with the exception for D50a dataset. In general, we can conclude that the CP approach is slightly better for instances with lower order strength (regardless total or average), i.e. for more parallel instances.

- The number of alternative branchings ($\#AB$) and the number of alternative process plans ($\#APP$) are also quite related to each other, since each alternative branching rises the number of process plans. Both factors have a high influence on the different solution approaches. With a growing number of the alternative branchings and especially with a growing number of alternative process plans, the IRSA heuristic usually becomes more effective than the CP approach. The reason may lie in the limited branching strategy of the CP solver, which is then not able to explore more alternatives of the activities selection.

- The average number of activities per process plan ($PPAct$) seems to be important for the smaller instances, where the constraint programming approach is more effective for a higher number of activities per process plan. For the larger instances, the property looses its importance.

- The maximal level of nested alternative branching (NL) does not show any significant importance for the effectiveness of the solution approaches. If any, the CP approach is better for a lower number, i.e. less nested instances.

- The number of resources ($\#res$) also does not have a high importance for the quality of the solutions of both approaches. On the other hand, the resources constrainedness (RC) became the most important factor with respect to the results of the t-tests over all the instances. The CP approach is always better for a higher value, i.e. for the instances where the activities have higher resource demands and the resources become a more critical constraint of a problem.

- The average activity slack (AAS) is an important factor for 3 datasets while for the others the influence is not conclusive. The CP approach is usually better for a higher average activity slack, but not with a great significance and moreover, for the largest dataset, the opposite progress is shown.

Based on the experiments, we can conclude that the CP approach is more effective for the instances with lower ratio of alternative parts (properties $\#AB$, $\#APP$ and NL), less predefined order of activities in the schedule (properties TOS and AOS) and more constrained resource environment (properties $\#res$ and RC). The impact of the property AAS is not straightforward from the experiments.

The CP approach can be used also for instances with more than 200 activities, but such instances are not included in this thesis since the difference in the solution time to achieve comparable results is becoming too high.

3.4 Conclusion

This chapter is dedicated to the resource constrained project scheduling problem with alternative process plans $PS|nestedAlt, temp, ST_{SD}|C_{max}$, motivated by the production of the wire harnesses in Styl Plzeň. We have decided to represent the structure of the problem by Nested Temporal Networks with Alternatives and for such representation, the mathematical model able to solve, in a reasonable amount of time, instances with up to 50 activities per resource time is presented.

In order to solve larger problems in the nested form, we have developed the heuristic algorithm IRSA. Computational experiments demonstrate good performance of this algorithm with a mean difference from the optimal value of the makespan less than 7%, while solving time for instances with 200 activities within 300 milliseconds. Instances with up to 2000 activities can be solved in the order of a few seconds. Moreover, the instances of two related problems have been used for the algorithm evaluation and the experiments showed that the IRSA algorithm is able to solve much more specific problems with good quality of solutions in very short time.

Finally, we have presented a novel metric for the characterization of the instances of scheduling problems with alternative process plans. Such a metric is used for the comparison of the different solution approaches, namely the constraint programming solver and the IRSA algorithm. The results of the experiments are evaluated by the statistical methods and the important properties are derived and discussed.

Chapter 4

RCPSP-APP under minimization of the total setup time

This chapter is dedicated to the RCPSP-APP problem where the goal is to minimize the *total setup time* (*TST*), which is equal to the sum of the overall performed setup time in the schedule. The main motivation for the research is the manufacturing of the small electrical connectors, where the reconfiguration of the machinery is very costly and the correct order of jobs is crucial. Up to our knowledge, there is no existing solution approach for such a problem and therefore, a new model and a new heuristic algorithm is proposed for the considered problem with the intention to solve large instances with up to 1000 activities.

Sequence dependent setup times (also called *changeovers*) are crucial for the problems where the resources are very expensive in terms of wasting their time by unnecessary setups. Setup times represent the time necessary to reconfigure the resource or to change its functionality. During this time period, no work on the resources can be performed, which can cause the entire process flow to be inefficient. The problem in minimisation of the total setup time is a part of many manufacturing processes (we "sell the machinery time"). Yet the setup times are almost always considered only as a problem constraint, not as a part of the criterion. One of the main goals of this research is to fill the gap in this area, i.e. to propose a generic approach to deal with the minimisation of the total setup time.

Allahverdi et al. (2008) dealt with the setup times in general and published a survey in which many different problems related to the setup times are summarised. The authors also reported on solution approaches and proposed a notation for all of these problems. Yuan et al. (2004) published a study for a metal casting company concerning the minimisation of the total setup costs in which the authors demonstrate the importance of setup times by calculating the savings to the company. Focacci et al. (2000) dealt with the general shop problem with the sequence dependent setup times. The authors proposed a two phase Pareto heuristic to minimise the makespan and the total setup costs. In the first phase, the makespan is minimised and, in the second phase, the total setup costs are minimised, while the makespan is not allowed to get worse. Wang and Wang (1997) focused on a single machine earliness tardiness problem with sequence dependent setup times. The objective function is to minimise the total setup time, earliness and tardiness. Mirabi (2010) proposed a hybrid simulated annealing algorithm for the single machine problem with sequence dependent setup times. The objective function is given by the sum of the setup costs, delay costs and holding costs.

The main contribution of this chapter is the formulation of the novel problem, incorporating the criterion based on the performed setup times into the area of the resource constrained project scheduling problems with alternative process plans. Such a problem has not been studied before in this range. There were only a few attempts to deal with the scheduling problems where the criterion reflects the setup times. The closest problem that can be found in the literature, when compared to the approach studied in this chapter, was published by Focacci et al. (2000) who focused on the job shop problem with the alternative machines while the makespan and the total setup time is minimised. Compared to the problem studied in Focacci et al. (2000), the model proposed in this chapter is developed for more general problems, namely for non-unary resources, deadlines of activities and more complex precedence rules including alternative process plans.

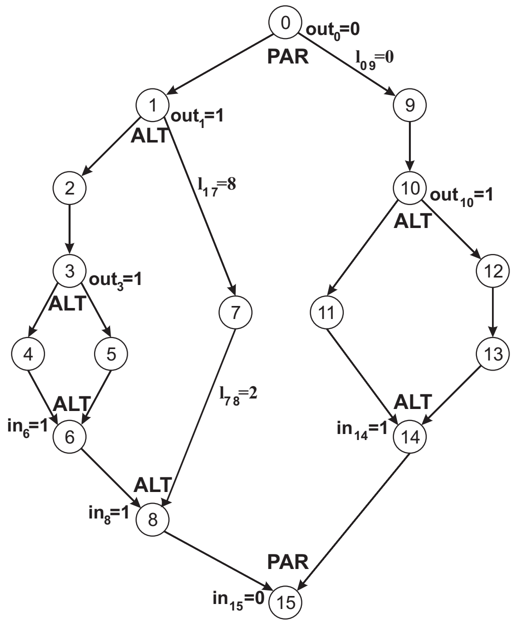
The second contribution lies in the newly developed algorithm able to solve the instances of the RCPSP-APP problem with up to 1000 activities. The effectiveness of the algorithm is evaluated using the datasets published in Brucker and Thiele (1996) while the proposed algorithm outperforms the results presented in Focacci et al. (2000). Moreover, the algorithm presented in this chapter is able to solve instances with 1000 activities within dozens of seconds.

The rest of the chapter is organised as follows: Section 4.1 provides a definition of the considered problem for which the mathematical model is presented in Section 4.2. A new heuristic algorithm is proposed in Section 4.3. Section 4.4 presents the results of the performance evaluation of the developed algorithm and Section 4.5 concludes the work.

4.1 Problem statement

In conjunction to the common definition in Chapter 2, each each activity $i \in \mathcal{A}$ has a deadline $\tilde{d}_i \geq 0$. The temporal constraints are given as the non-negative start to start time-lags restricted to the precedence-constrained activities only, i.e. $l_{ij} \geq 0$ for all $(i, j) \in E(G)$ and $l_{i,j} = -\infty$ otherwise; G represents the NTNA instance as defined in Section 2.5.

The instance of the problem considered in this chapter is depicted in Figure 4.1. Several time-lags are used to demonstrate how the temporal constraints are defined,



p=[0, 3, 2, 1, 3, 2, 1, 7, 2, 3, 2, 3, 2, 3, 2, 0]

r=[0, 0, 0, 6, 6, 6, 6, 5, 0, 0, 2, 5, 5, 5, 0, 0]

$\tilde{d}$=[5, 9, 9, 9, 15, 15, 20, 9, 25, 9, 11, 15, 13, 16, 20, 30]

res=[4, 2, 1, 2, 2, 3, 1, 3, 3, 1, 2, 3, 3, 3, 2, 4]

R_i^k=[0, 2, 2, 1, 2, 3, 2, 1, 1, 2, 2, 3, 2, 2, 1, 0]

R_1	2	6	9
2	0	3	1
6	3	0	3
9	1	3	0

R_2	1	3	4	10	14
1	0	3	3	4	2
3	2	0	2	4	2
4	3	3	0	2	3
10	4	4	2	0	2
14	3	2	3	2	0

R_3	5	7	8	11	12	13
5	0	2	2	0	0	3
7	2	0	0	2	3	3
8	2	0	0	2	3	3
11	0	3	3	0	0	3
12	0	2	2	0	0	3
13	3	3	3	3	3	0

Fig. 4.1: Nested temporal network with alternatives - example

see e.g. time-lag $l_{17} = 8$ that forces activity 7 to start at least 8 time units after the start time of activity 1. All the parameters related to the activities are also included.

The goal of the scheduling process is to minimize the total setup time, denoted as *TST*, which is equal to the sum of all the setup times performed in the schedule. In addition to the variables for the schedule representation defined in Section 2.2 variable $f_{ij} \in \{0, 1\}$ is defined as follows: If activities i and j are scheduled subsequently on the same resource type and they share at least one unit of its resource capacity, then $f_{ij} = 1$; $f_{ij} = 0$ otherwise.

The setup time from activity i to activity j is always considered only once in the objective function, regardless the actual number of the resource units which are shared by both activities. Lets assume that activity i requires three units of a certain resource type and activity j also requires three units of the same resource types. Furthermore, lets assume that activity i is assigned to resource units $\{1, 2, 4\}$ and activity j is assigned to resource units $\{2, 3, 4\}$. Although the activities share two resource units, the setup time from i to j will be added to the value of the objective function only once.

The considered problem is denoted as $PS|nestedAlt, l_{ij}^{min}, ST_{SD}, r_j, \tilde{d}_j|TST$ or as $m1|nestedAlt, min, ST_{SD}, r_j, \tilde{d}_j|TST$ using the same extended notation of Brucker et al. (1999a) and Herroelen et al. (1999), respectively as in Section 2.7. The term r_j stands for the release times and $\tilde{d}_j$ denotes the deadlines.

4.2 Mathematical formulation

The mathematical formulation using the mixed integer linear programming (MILP) for the problem defined in the previous section is formulated below. For a higher efficiency of the model, variable z_{ivk} is substituted by variable z_{iu}, i.e. only one index u is used to reference the assigned resource units of a certain resource type. The mutual conversion between (v, k) and u is given as follows:

$$u = \sum_{q=1}^{k-1} \theta_q + v \tag{4.1}$$

$$k = arg \min_{k} \left\{ \sum_{q=1}^{k} \theta_q \geq u \right\}; v = u - \sum_{q=1}^{k-1} \theta_q \tag{4.2}$$

In addition to variables s_i, v_i, f_{ij} and z_{ivk} (z_{iu}) defined in the previous section, auxiliary binary variables g_{ijk}, x_{ijk} and y_{ijk} are used. Variable g_{ijk} determines whether activities i and j are selected and assigned to the same resource unit k such that i is a direct predecessor of j on such resource unit. Similarly, variable x_{ijk} determines whether activities i and j are selected and assigned to the same resource unit k such that i is an arbitrary (direct or propagated) predecessor of j on such resource unit. Finally, variable y_{ijk} determines whether both activities i and j are assigned to resource unit k.

$$\min \sum_{\forall i \in \mathcal{A}} \sum_{\forall j \in \mathcal{A}} f_{ij} \cdot st_{ij}$$

subject to:

$$(2.1) - (2.7), (3.1)$$

$$\sum_{u=C+1}^{C+\theta_q} z_{iu} = \mathrm{R}_i^q \cdot v_i \qquad \forall i \in \mathcal{A}; \forall q \in \{1 \dots m\}; C = \sum_{j=1}^{q-1} \theta_j \tag{4.3}$$

$$s_j + p_j + st_{ji} \leq s_i + UB \cdot (x_{iju} + 1 - y_{iju}) + UB \cdot (2 - v_i - v_j)$$
$$\forall (i,j) \in \mathcal{A}^2 : i \neq j; \forall u \in \{1 \dots K\} \tag{4.4}$$

$$s_i + p_i + st_{ij} \leq s_j + UB \cdot (2 - x_{iju} - y_{iju}) + UB \cdot (2 - v_i - v_j)$$
$$\forall (i,j) \in \mathcal{A}^2 : i \neq j; \forall u \in \{1 \dots K\} \tag{4.5}$$

$$z_{0u} = 1 \qquad \forall u \in \{1 \dots K\} \tag{4.6}$$

$$z_{n+1\,u} = 1 \qquad \forall u \in \{1 \dots K\} \tag{4.7}$$

$$y_{iju} \geq z_{iu} + z_{ju} - 1 \qquad \forall (i,j) \in \mathcal{A}^2 : i \neq j; \forall u \in \{1 \dots K\} \tag{4.8}$$

$$y_{iju} \leq z_{iu} \qquad \forall (i,j) \in \mathcal{A}^2 : i \neq j; \forall u \in \{1 \dots K\} \tag{4.9}$$

$$x_{iju} \leq y_{iju} \qquad \forall (i,j) \in \mathcal{A}^2 : i \neq j; \forall u \in \{1 \dots K\} \tag{4.10}$$

$$\sum_{j=1}^{n+1} g_{iju} = z_{iu} \qquad \forall i \in \mathcal{A}; \forall u \in \{1 \dots K\} \tag{4.11}$$

$$\sum_{i=0}^{n} g_{iju} = z_{ju} \qquad \forall j \in \mathcal{A}; \forall u \in \{1 \dots K\} \tag{4.12}$$

$$g_{iju} \leq x_{iju} \qquad \forall (i,j) \in \mathcal{A}^2; \forall u \in \{1 \dots K\} \tag{4.13}$$

$$f_{ij} \cdot UB \geq \sum_{\forall u \in \{1 \dots K\}} g_{iju} \qquad \forall (i,j) \in \mathcal{A}^2 \tag{4.14}$$

where :

$$f_{ij}, z_{iu}, g_{iju}, x_{iju}, y_{iju} \in \{0,1\}; K = \sum_{q=1}^{m} \theta_q;$$

The constraints for the selection of the activities and the temporal constraints are taken over from Chapter 2. New formulas (4.3)-(4.14) serve to define the resource constraints in a similar way as for formulas (2.8)-(2.12). The key issue is that new formulated constraints allow to reflect the performed setup times in the objective function.

Formulas (4.4) and (4.5) prevent more activities (from overlapping) on one resource unit in one moment. Equation (4.3) ensures that the number of the assigned resource units is equal to the resource demand for each activity. Equations (4.6) and (4.7) are used to assign dummy activities 0 and $n+1$ to each resource unit of each resource type, which then ease the definition of the constraints related to the setup times. Formulas (4.8) and (4.9) constrain the value of variable y_{ijk} - if both activities are scheduled on the same resource unit, then y_{ijk} is equal to 1; 0 otherwise. Formula (4.10) determines the value of variable x_{ijk} - if both activities i and j are assigned to the same resource unit k, they must be scheduled sequentially. Equation (4.11) forces each activity to have only one direct successor on each assigned resource unit. Similarly, Equation (4.12) forces each activity to have only one direct predecessor on each resource unit. Formula (4.13) prevents the cycles in values of variable g_{ijk} for each resource unit. Finally, Formula (4.14) determines whether a particular setup time has to be taken into consideration in the objective function, i.e. whether activities i and j are scheduled subsequently on the same resource unit.

4.3 Heuristic algorithm

This section is dedicated to the description of the heuristic algorithm designed to solve the large instances of the problem defined in Section 4.1. The goal is to find a schedule determined by the selection of activities (variable v_i), their start times (variable s_i) and their assignment to resources (variable z_{ivk}) such that all the constraints are satisfied and the total setup time (TST) value is minimised.

The basic scheme of the proposed heuristic algorithm, called *STOAL* (Setup Time Optimization ALgorithm), consists of two phases - the initial phase to find any feasible solution and the local search for the improvement of the objective value. The initial phase is inspired by the IRSA algorithm presented in Section 3.2 and the local search, based on a time separation technique, is inspired by the work of Focacci et al. (2000). If a feasible solution is not found (due to the presence of deadlines) in the initial phase, the local search is not started at all and the algorithm is terminated.

4.3.1 Initial solution

In the first phase, the STOAL algorithm tries to find any feasible solution by the gradual construction of a schedule with a simple backtracking scheme for recovering from infeasible solutions. The activities are added into the schedule one by one according to the priority rules, which resolve both the selection of activities and their sequencing on the resources. Since the goal is to find any feasible schedule, the schedule is constructed without considering the value of the objective function. Once a feasible solution is found, the initial phase is terminated. The basic procedure for the initial phase of the algorithm is shown in Algorithm 3.

4.3.1.1 Propagation of release times and deadlines

The first step of the initial solution is the propagation of the release times and the deadlines among all the activities using the structure of the instance. The propagation serves for tightening the absolute time windows of the activities, resulting in more accurate estimates of their priorities for adding to the schedule.

The propagation of the release times among the activities is performed from activity 0 towards activity $n+1$ using the temporal constraints defined by the NTNA instance. The release time for activity i is calculated as $r_i = \max(r_i, \widehat{r}_i)$. If there is an alternative branching ending in activity i ($in_i = 1$), then $\widehat{r}_i = \min\limits_{\forall j:(j,i)\in E} (r_j + l_{ji})$; $\widehat{r}_i = \max\limits_{\forall j:(j,i)\in E} (r_j + l_{ji})$ otherwise.

Algorithm 3 Initial phase of the STOAL algorithm

```
Propagate release times and deadlines
Establish priorities of branches for schedule construction
Add activity 0 to ready set
Set the maximum number of backtracking steps
$while$ (ready set is not empty)
   $i$ = select activity from ready set
   Try to schedule activity $i$ and update release times of activities in ready set
   $if$ (activity $i$ is not successfully scheduled)
      Apply backtracking scheme
      $if$ (maximum number of backtracking steps reached)
         End with failure
      $end\,if$
   $end\,if$
   Update ready set
$end\,while$
Return solution
```

Similarly, the propagation of the deadlines among the activities is performed from activity $n+1$ towards activity 0. The deadline of activity i, using the temporal constraints defined by the NTNA instance, is calculated as $\tilde{d}_i = \min\left(\tilde{d}_i, \widehat{d}_i\right)$. If there is an alternative branching starting in activity i ($out_i = 1$), then $\widehat{d}_i = \max\limits_{\forall j:(i,j)\in E}(\tilde{d}_j - p_j - l_{ij} + p_i)$; $\widehat{d}_i = \min\limits_{\forall j:(i,j)\in E}(\tilde{d}_j - p_j - l_{ij} + p_i)$ otherwise.

The values of the release time and the deadline for each activity are updated using the described propagation technique and for the rest of the algorithm run, the new values are considered.

4.3.1.2 Priorities of branches for schedule construction

After the propagation of the temporal constraints, the priorities of all the activities are calculated. For this purpose, we define the *flexibility* of activity i as $flex_i = \tilde{d}_i - p_i - r_i$. The lower flexibility values correspond to the activities that have a more tight time window and, therefore, they are more critical in the scheduling process. The flexibility is used to determine the selection of the activities as well as to determine the actual order in which the activities are added into the schedule.

The priority of the branches in each alternative branching is determined as follows. Let $\mathcal{B}_{ab} = \{B_1 ... B_{\delta^+(a)}\}$ be the set of all branches of the alternative branching that begins in node a and ends in node b, $\delta^+(a)$ is the out-degree of node a. Each branch

$B_j \in \mathcal{B}_{ab}$ consists of activities that form a subgraph starting by some successor of activity a and ending by the corresponding predecessor of activity b. The average flexibility of all the activities in $\mathcal{B}_{ab}$ is then given as $\overline{flex}_a = average(flex_i)$ for all $i \in B_j$, $B_j \in \mathcal{B}_{ab}$. To estimate the probability that a branch would be successfully scheduled with respect to the temporal constraints, the activities with the flexibility lower than the average flexibility of the alternative branching are the most critical. Therefore, to calculate the total flexibility of branch $B_j \in \mathcal{B}_{ab}$, the following formula is used $flex_{B_j} = \sum(\overline{flex}_a - flex_i)$ for all $i \in B_j : flex_i \leq \overline{flex}_a$.

Once the total flexibility of each branch in the alternative branching is calculated, the branches are sorted in each alternative branching for later schedule construction. To distinguish the order between two branches of the same alternative branching, the total flexibility is used - the branch with a lower $flex_{B_j}$ value has a higher priority. If two branches have the same total flexibility, then the branch with a lower number of activities has a higher priority. Finally, if even the number of activities is the same, we use the branch starting by the activity with the lower number (to keep the deterministic nature of the algorithm).

4.3.1.3 Schedule construction

The schedule is constructed by adding one activity from the ready set in each iteration according to the *serial generation scheme* (*SGS*) described by Kolisch (1996), i.e. each activity is scheduled to the earliest possible time point with respect to the temporal and the resource constraints. The *ready* set is a set of all the activities for which all the required predecessors are already scheduled. The order in which the activities are added to the schedule is determined by their flexibility (defined in the previous section), which is being updated after each change in the schedule. If an activity is scheduled, then it is removed from the ready set and the ready set is further updated as follows: if there is an output alternative branching defined for the current scheduled activity, its direct successor belonging to the branch with the highest priority is added to the ready set; otherwise all direct successors of the current scheduled activity are added to the ready set. At the beginning, the ready set only contains activity 0.

To select a particular activity from the ready set, an attempt to schedule each ready activity is first evaluated and then the activity with the lowest evaluation is scheduled. An attempt to schedule the activity and its evaluation is carried out in the following way: First, activity i from the ready set is taken and the earliest time is calculated as $s_i = \max\left(s_i^{temp}, s_i^{res}\right)$ where $s_i^{temp} = \max\left(r_i, \max\limits_{\forall j:(j,i)\in E} s_j + l_{ji}\right)$ and s_i^{res} is the earliest start time with respect to the resource constraints. The assessment of the schedule attempt is then calculated as $C_i = \min\limits_{\forall j:(i,j)\in E}\left(\tilde{d}_j - p_j - (s_i + l_{ij})\right)$. Finally, the activity with the lowest assessment is selected to be scheduled. We try to schedule the most critical activity first and since the C_i value represents the minimal flexibility of all the successors of activity i, the lower values correspond to more critical activities.

If $s_i + p_i > \tilde{d}_i$ holds for any activity from the ready set, then the activity cannot be scheduled and a backtracking scheme (see Section 4.3.1.4) is applied. The rationale is straightforward - if activity i from the ready set cannot be scheduled currently, it cannot be scheduled even later. Since all the activities from the ready set have to be scheduled sooner or later, there is no way how to obtain a feasible schedule with respect to the partial schedule and the current set of ready activities. Therefore, recovering from such a dead-end is achieved by the backtracking scheme.

Activity i, selected from the ready set, is always assigned at the end of the schedule on the demanded resource type R_k. In other words the activity is assigned to the R_i^k resource units such that it is the last scheduled activity (with the highest start time) on each of the assigned resource units. For this purpose, the algorithm keeps the information about the last activity on each resource unit for each resource type. Let $last_q = \underset{j \in \mathcal{A}: z_{jqk}=1}{arg \max} \{s_j\}$ be the last activity on the q-th unit of resource type R_k and $\mathcal{L} = \{last_1 \dots last_{\theta_k}\}$ be the set of the last activities over all units of resource type R_k. Then the algorithm searches for a set of R_i^k resource units $\mathcal{J} \in \mathcal{L}$ such that $s_i = \max\limits_{j \in \mathcal{J}} \{s_j + p_j + st_{ji}\}$ is minimal. Such a task can be easily accomplished by sorting and selecting an appropriate number of resource units. Finally, activity i is assigned to all resource units from $\mathcal{J}$ and its start time is set to s_i. As soon as activity i is scheduled, the ready set is updated as described before.

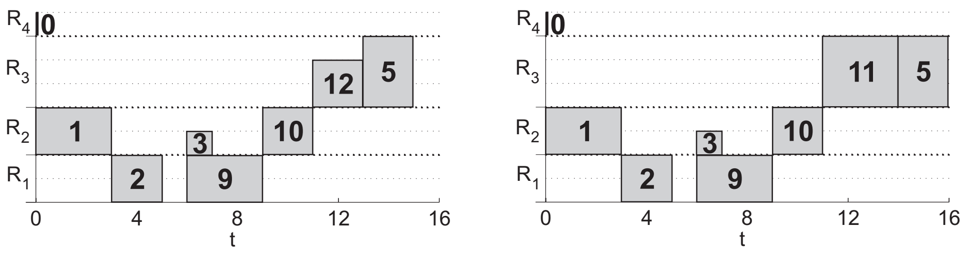
4.3.1.4 Backtracking scheme

The backtracking scheme is used for recovering from the situation when no further activity can be scheduled with the satisfaction of all the defined constraints. The backtracking scheme used in this chapter consists of unscheduling a part of the schedule and updating the ready set and release times of the activities. Then there are two ways how to continue with schedule construction in another direction - first, the selection of the activities can be changed and second, the sequence of the activities on some resources can be modified. After the backtracking scheme is applied, the schedule construction continues in the same way as described in the previous section.

If activity i cannot be scheduled at some point of the schedule construction, then the backtracking method is chosen up to the structure of the NTNA instance (see Section 4.1). If activity i is a part of the alternative branching (i.e. the most nested branching, in which the alternative is included, is the alternative branching), then a change in the selection of activities is used. Otherwise, the change in the sequencing of activities on the resources is used. Even with the backtracking method applied it may happen that, activity i still cannot be successfully scheduled. Then the whole branching, in which the activity is nested, is unscheduled and we apply the backtracking method for the activity that starts such a branching. If further backtracking is not possible (backtracking from activity 0), the entire algorithm ends with failure.

4.3.1.5 Change of the selection of activities

Let activity i be a part of branch B_x in alternative branching that consists of N branches $\mathcal{B}_{ab} = \{B_1 \dots B_x \dots B_N\}$. The change of the activities selection imposed by backtracking for activity i starts by unscheduling all the activities preceding activity i in the same branch, i.e. $\forall j \in B_x$: there is a directed path from j to i in the NTNA instance. Then the first activity of branch B_y is added to the ready set where B_y is the next branch after B_x with respect to the priorities defined in Section 4.3.1.2. If there is no such branch B_y, then the backtracking is not successful and the algorithm continues with the backtracking for the activity that starts the alternative branching $\mathcal{B}_{ab}$. The priorities of the branches in $\mathcal{B}_{ab}$ are then reset to the initial state.



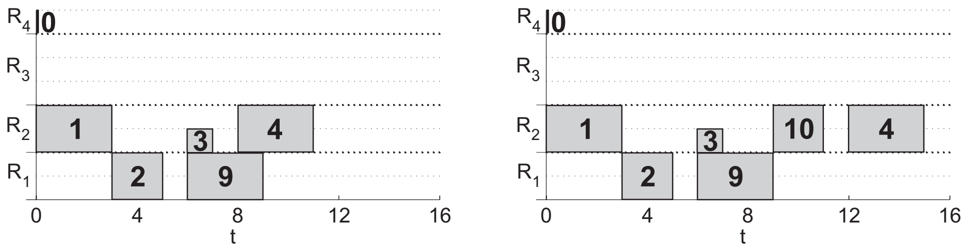
(a) Alt change before (b) Alt change after

Fig. 4.2: Change of the activities selection in the schedule

An example of the backtracking by the change of the selection of activities is depicted in Figure 4.2, which is based on the example presented in Figure 4.1. The next activity from the ready set to be added into the schedule in Figure 4.2a is activity 13 (demanding resource R_3), which cannot be scheduled within its deadline. Therefore, the selection is changed such that the branch formed by activities 12 and 13 is removed from the schedule and the branch containing activity 11 is selected. The result with scheduled activity 11 is shown in Figure 4.2b.

4.3.1.6 Change of the sequence on resources

In the case when activity i is a part of the parallel branching, the change of the activities sequencing on the resources is used as the backtracking method. First, activity j scheduled on the same resource type as is demanded by activity i is found such that there is no directed path from j to i in the NTNA instance and j has the maximal start time. Activity j and all its successors in NTNA are unscheduled (both direct and propagated successors). Activity j is added to the ready set and the schedule on all the resources is updated by shifting the activities to the left, since a part of the resources



(a) Resource change before (b) Resource change after

Fig. 4.3: Change of the sequencing of activities on the resource

capacity is released by unscheduling activity j and all its successors in NTNA. Finally, the algorithm continues with a new attempt to schedule activity i.

An example of the backtracking by the change of activities sequencing on the resource is shown in Figure 4.3, based on the example presented in Figure 4.1 again. The next activity from the ready set to be added into the schedule in Figure 4.3a is activity 10 (demanding resource R_2), but its addition to the last position on the resource would violate its deadline. Therefore, the sequencing of activities on the resource is changed such that activity 4 is unscheduled, activity 10 is scheduled first and then activity 4 is scheduled. The resulting schedule is depicted in Figure 4.3b

Regardless of the type of the backtracking scheme used, the partial schedule is kept feasible and the set of ready activities is up to date. Then the algorithm continues with the schedule construction as described in the previous section. The maximal number of the applied backtracking steps can be set as an input parameter of the algorithm.

4.3.2 Schedule improvement

The aim of the second phase of the STOAL algorithm is to improve the value of the objective function of the schedule found in the initial phase of the algorithm. The basic idea is to divide the whole schedule into more independent parts and then to optimise each part separately, while the rest of the schedule has to stay intact. The time-separation design for schedule improvement has been proposed in Focacci et al. (2000) where the goal is to minimise the makespan and then the total setup time for the general shop problem. The model defined in Section 4.1 is a generalisation of the problem studied in Focacci et al. (2000) and, therefore, the heuristic algorithm described in this section uses only the basic idea of time-separation while most of the algorithm is redesigned for the needs of the problem studied in this chapter.

To improve the value of the objective function, only the sequencing of activities on the resources is modified, the selection of activities is not changed. In other words, the second phase of the algorithm works with the fixed set of selected activities. The basic scheme of the second phase of the STOAL algorithm is depicted in Algorithm 4 where $numberOfRepetitions$ specifies the number of repetitions of the local search procedures over the whole schedule.

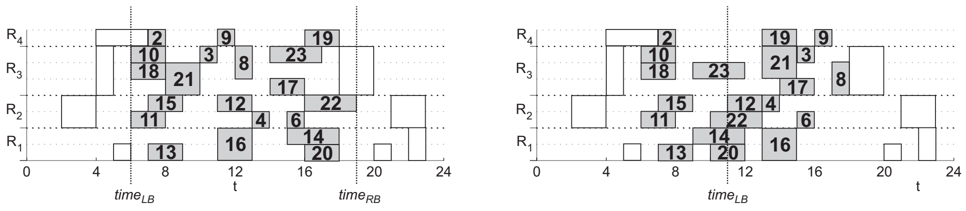
Algorithm 4 Second phase of the STOAL algorithm

```
$for\ (i = 1 \ldots numberOfRepetitions)$
  Determine the first time window
  $while$ (not reached the end of the schedule)
    Determine activities for the time window
    Determine set of ready activities
    Optimise the time window
    $if$ (objective value improved)
      Integrate time window into overall schedule
    $end\ if$
    Determine the next time window
  $end\ while$
$end\ for$
```

4.3.2.1 Time window

The principle of the time-separation technique is to divide the whole schedule into more disjunctive parts, called *time windows*, and to optimise the value of the objective function for each time window such that the rest of the schedule is not changed at all. The time window is formed by a set of all scheduled activities for which the start time and the completion time lie in the same time interval $\langle time_{LB}, time_{RB} \rangle$ where $time_{LB}$ is the time window *left border* and $time_{RB}$ is the time window *right border*. A time window is therefore defined as $\mathcal{W} = \{\forall i \in \mathcal{A} : s_i \geq time_{LB} \wedge s_i + p_i \leq time_{RB}\}$. To determine the left and the right border, the maximal number of activities per resource type in a time window is used, i.e. the activities are being added into the time window until the the predefined number of activities is met for any resource. Using such an approach, all the time windows will be of the similar complexity (a similar number of possible resource conflicts to resolve) even though the absolute time-length of the windows can be quite different.

For each $i \in \mathcal{W}$ the window release time and the window deadline is calculated such that if activity i is scheduled within this interval, the rest of the schedule is not influenced at all. For this purpose, both the temporal and the resource constraints have to be considered. Since the problem studied in this chapter includes non-unary resources, the window release time and deadline of an activity is different for each resource unit. The window release time of activity i for resource unit q is given as $\widehat{r}_{iq} = \max\left(\widehat{r}_i^{\,temp}, \widehat{r}_{iq}^{\,res}\right)$ where $\widehat{r}_i^{\,temp} = \max\limits_{\forall j \notin \mathcal{W}:(j,i) \in E} (s_j + l_{ji})$ and $\widehat{r}_{iq}^{\,res} =$



(a) Time window before

(b) Time window after

Fig. 4.4: Example of the optimisation within the time window

$\max\limits_{\forall j \in \mathcal{A}: s_j < time_{LB} \wedge z_{jq}=1} (s_j + p_j + st_{ji})$. The window release time of the activity i is calculated as the minimum time such that scheduling of activity i will never affect any activity before the current time window. Therefore, only the activities that are prior to the time window are considered for the window release time computation.

Similarly, the window deadline of activity i for resource unit q is given as $\widehat{d}_{iq} = \min\left(\widehat{d}_i^{temp}, \widehat{d}_{iq}^{res}\right)$ where $\widehat{d}_i^{temp} = \min\limits_{\forall j \notin \mathcal{W}:(i,j) \in E} (s_j - l_{ij})$ and $\widehat{d}_{iq}^{res} = \max\limits_{\forall j \in \mathcal{A}: s_j + p_j > time_{RB} \wedge z_{jq}=1} (s_j - p_i - st_{ij})$. As for the window release time, the window deadline of activity i is calculated such that the activity will never influence the schedule after the time window.

After the optimisation of a time window, the next time window is set such that there is one-half overlapping with the current time window in terms of the included activities. Let $\left\{s_1 \ldots s_{|\mathcal{W}|}\right\}$ be the start times of all the activities from the current time window sorted in non-decreasing order, $|W|$ is the number of activities in the time window. Then the left border of the next time window is calculated as $time_{LB} = s_{\lceil |\mathcal{W}|/2 \rceil}$. The right border is then set with respect to the maximal number of activities per resource type.

Figure 4.4 depicts an example of the time window before (Figure 4.4a) and after (Figure 4.4b) the optimisation. The example is not related to the instance depicted in Figure 4.1, a bigger instance is used instead to show a meaningful time window. The time window is delimited by the left border (vertical dashed line at time 6) and the right border (time 19). All the activities that start and end within the time window are included into the optimisation process while the rest of the schedule (activities without numbers) is not allowed to be modified at all. The left border for the next time window is depicted in Figure 4.4b at time 11.

4.3.2.2 Selection from ready set

The schedule construction method is quite similar to the one used in the initial phase, only the rules to choose an activity to be scheduled and also the backtracking scheme are different. The optimisation of the schedule in a time window begins by unscheduling all the activities in this time window. Then the set of ready activities, which can be currently added into the schedule, is established, similar to the initial phase of the algorithm. The set of ready activities contains (in each time moment) all the activities, for which all predecessors are already scheduled. For this purpose, we assume that the selection of the activities is fixed, i.e. the rejected activities are not considered at all. The ready set is updated after each step of the algorithm - after each scheduling/unscheduling of an activity.

To select one activity from the set of ready activities, the algorithm tries to schedule each activity from the ready set and then the activity, which caused the minimal increase in the total setup time, is selected and actually scheduled. The assignment of an activity to the resource type demanded is handled also with respect to the minimal increase of the objective value. Therefore, activity i is always scheduled to the resource units where the setup time is the lowest, with respect to the temporal constraints given by $\widehat{r_{iq}}$ and $\widehat{d_{iq}}$. In case that the activity cannot be scheduled within the specified release time and deadline, the activities are being rescheduled in a different order. If the activity is successfully scheduled within $\widehat{r_{iq}}$ and $\widehat{d_{iq}}$, then it is assured that no temporal constraints are violated considering the schedule outside the time window.

4.3.2.3 Limited discrepancy search

The method selecting an activity from the ready set, described in the previous section, is used in combination with the *limited discrepancy search* (*LDS*). Limited discrepancy search proposed by Harvey and Ginsberg (1995) is a special kind of the branch and bound (B&B) algorithm where the total number of the tested nodes is very limited. Generally, an activity from the ready set to be scheduled is chosen by some heuristic rule and a schedule is constructed while following that rule. In case of the limited discrepancy search, there are some points in the schedule construction (called discrepancies) where the heuristic rule is not used, which can lead to a different schedule. The process of finding the solution for the single time window is repeated several times, while the discrepancy is used in different moments of the schedule construction.

In the branch and bound algorithm, the actual schedule is determined by the current node in the search tree. A *search tree* is a directed acyclic graph (more precisely out-tree graph), where each node corresponds to a (partial) schedule determined by the order in which the activities are added to the schedule, using the serial generation scheme. Each node has the number of successors equal to the number of ready activities with respect to the current schedule. A complete schedule is represented by each leaf of the search tree, the rest of the nodes represent the partial schedules. A full branch and bound algorithm basically enumerates all feasible schedules with possible cuts in the tree dues to an estimation of the upper and lower bounds.

The limited discrepancy search is based on the same principle with the exception

Algorithm 5 Optimisation of the time window schedule

```
$bestSchedule =$ current schedule
$for\, (h = 0 \ldots n)$
    Unschedule $n - h$ activities
    Determine ready activities
    Select activity $i$ by discrepancy on level $h$
    $while$ (activity $i$ can be scheduled)
        Update ready activities
        $if$ (ready set is empty)
            $break$
        $end\, if$
        Select activity $i$ by heuristic rule
    $end\, while$
    $if$ (new best schedule found)
        $bestSchedule =$ current schedule
    $end\, if$
$end\, for$
return $bestSchedule$
```

that instead of constructing the full search tree, a heuristic rule is always used to determine only one successor of each node, which is further expanded. Then, for some predefined nodes in the graph, the discrepancy is used to search in another direction, which means that the heuristic rule is not used and a different direction is chosen. The nodes where the discrepancy is used and also the number of discrepancies can be chosen in a wide range. In our case, we always use, at most, one discrepancy per one schedule construction.

Let the time window consists of n activities. Then the depth of the search tree (the number of decision points) is equal to n. The attempt to construct a schedule follows the heuristic rule (described in the previous section) in each decision point. Then each attempt contains exactly one discrepancy on level $h \in \{1 \ldots n\}$ and the heuristic rule is used in all other nodes. As a consequence there are $n + 1$ attempts to construct a schedule within one time window. Each attempt is evaluated in terms of the objective function and at the end, the best schedule is compared with the original schedule and if the improvement in the objective value is achieved, the time window schedule is embedded into the overall schedule.

The activity to be scheduled in cases of discrepancy is chosen as follows: Instead of selecting the activity, which causes the minimal increase in the objective value, activity i with the lowest value of $\widehat{d_i} - p_i - s_i$ is used. This way, the least flexible

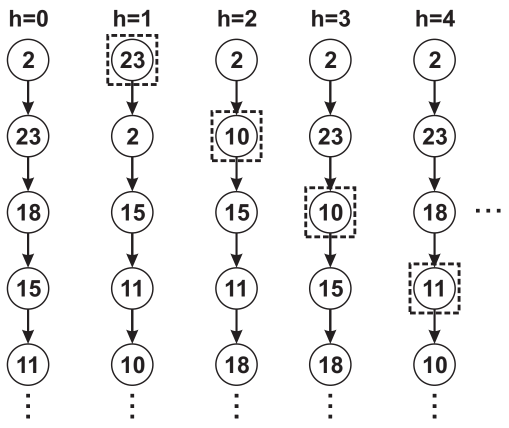


Fig. 4.5: Example of the limited discrepancy search

activity is added into the schedule at a node where the discrepancy is used.

If activity i, selected from the ready set, cannot be scheduled due to a violation of the temporal constraints, the current attempt to construct the schedule is terminated. The algorithm then continues with the next attempt while the level of discrepancy used is increased by one. To save the time wasted by unnecessary scheduling and unscheduling steps, the h-th iteration of the limited discrepancy search always starts from the schedule, where $h - 1$ activities are already scheduled from the previous iteration. The overall scheme of the limited discrepancy search is depicted in Algorithm 5.

An example of the limited discrepancy search for the time window depicted in Figure 4.4 is shown in Figure 4.5. For each iteration of the local search, the discrepancy is used in different level (denoted as h) and, therefore, also the sequence in which the activities are scheduled is modified.

4.4 Performance evaluation

Two sources of instances have been used for the performance evaluation of the algorithm proposed in Section 4.3, designed to solve the problems with alternative process plans. First, the STOAL algorithm is evaluated on randomly generated instances and compared with the IRSA algorithm proposed in Section 3.2. Second, the standard benchmarks of Brucker and Thiele (1996) are used and the results of the STOAL algorithm are compared with the results reported in Focacci et al. (2000). Furthermore, various settings of the STOAL algorithm are discussed and tested on large instances of the problem (up to 1000 activities). The STOAL algorithm was implemented in the C# language and the experiments were performed on a PC with an Intel Core 2 Quad CPU at 2.83GHz with 8GB of RAM.

	Dataset	D30	D50	D100	D200
IRSA	$feas$ [%]	98	100	96	97
	TST	120	254	494	942
	C_{max}	148	243	521	1054
	t_{cpu}	5	36	112	322
STOAL	$feas$ [%]	99	100	98	98
	TST	101	215	427	824
	C_{max}	171	282	618	1263
	t_{cpu}	3	12	77	141
	TST^{impr} [%]	15.84	15.35	13.56	12.53
	C_{max}^{det} [%]	13.46	13.83	15.71	16.55

Table 4.1: Comparison with IRSA algorithm using new random instances

4.4.1 Comparison with IRSA algorithm on random instances

The datasets D30, D50, D100 and D200, introduced in Section 3.3.1, are used to compare the STOAL algorithm with the IRSA algorithm described in Chapter 3. Table 4.1 shows the comparison of the results obtained by the IRSA algorithm and by the STOAL algorithm. Columns represent datasets, rows contain objective values and other performance indicators. Abbreviation $feas$ determines the percentage ratio of feasible solutions found by each algorithm, TST contains an arithmetic average value of the objective function for instances that were successfully solved by both algorithms, $time$ determines the average computational time (in milliseconds) to solve a single instance, TST^{impr} states the improvement of the STOAL algorithm over the IRSA algorithm in terms of the TST value and, finally, C_{max}^{det} denotes the deterioration of the makespan value achieved by the STOAL algorithm over the result achieved by the IRSA algorithm.

The number of feasible solutions found is almost the same for both tested algorithms, but the STOAL algorithm outperforms the IRSA algorithm in both the TST value and the solution time. The fact that the success rate in finding feasible solutions is equal proves that the STOAL algorithm is very effective for the considered temporal constraints, since the IRSA algorithm was developed with the main aim to find any feasible solution.

4.4.2 Comparison with algorithm of Focacci et al. (2000)

For a further evaluation of the STOAL algorithm, the instances of the general job shop problem proposed by Brucker and Thiele (1996) are used. As a reference, the results for such instances reported in Focacci et al. (2000) are considered. The problem studied in Brucker and Thiele (1996) is a sub-problem of the problem defined

	Focacci		STOAL			
Dataset	TST	C_{max}	TST	C_{max}	TST^{impr} [%]	C_{max}^{det} [%]
t2-ps12	1 530	1 445	1 010	1 920	33.99	32.87
t2-ps13	1 430	1 658	1 330	1 872	7.00	18.93
t2-pss12	1 220	1 362	950	1 599	22.13	17.4
t2-pss13	1 140	1 522	1 140	1 610	0	5.78
average	1 330	1 497	1 110	1 825	16.54	18.74

Table 4.2: Comparison with Focacci et al. (2000)

in Section 4.1 since there are no release times or deadlines, no alternative process plans and the resources are considered to be unary. The objective function reported in Focacci et al. (2000) is twofold, first the makespan in minimised and then the total setup time is being minimised without a deterioration of the makespan value.

Table 4.2 shows the comparison of the STOAL algorithm with the one published by Focacci et al. (2000). The meaning of the abbreviations in the table is the same as for Table 4.1. Compared with the algorithm described by Focacci et al. (2000), the STOAL algorithm improved the value of the total setup time by more than 16% in average. The price for the better value of the TST is the higher value of the makespan, by almost 19% in average. Such a trade-off between the makespan and the total setup time shows the good efficiency of the STOAL algorithm proposed in terms of the total setup time criterion.

4.4.3 Configuration of the STOAL algorithm

As described in Section 4.3, there are three main input parameters of the algorithm, which influence both the solution quality and the running time. Namely the maximum number of backtracking steps, the maximum number of activities per resource in a time window and the number of repetitions of the sliding windows local search procedure. To determine the influence of the settings on the quality of the solutions and the running time of the algorithm, the following set of additional experiments was performed.

For each of the algorithm settings, three different values are tested on four datasets, each containing 100 instances. The $medium$ datasets with 50 and 200 activities from Section 4.4.1 together with two further datasets with 500 and 1000 activities per instance are used. The number of resource types m is randomly chosen from interval $\langle 1, 10 \rangle$ for both datasets with 500 and 1000 activities while the rest of the instances properties is the same as in Section 4.4.1.

First, the influence of the maximal number of backtracking steps on the number of the feasible solutions is evaluated. For this purpose, only the initial phase of the algorithm is executed and the number of feasible solutions (out of 100) for each dataset and the average solution time in milliseconds is observed. The maximal

$\#backtracks$	$0.1 \cdot n$		$0.3 \cdot n$		$0.5 \cdot n$	
n	$feas$	$t\ [ms]$	$feas$	$t\ [ms]$	$feas$	$t\ [ms]$
50	54	1	60	4	60	7
200	39	2	62	7	63	16
500	42	10	57	54	61	107
1000	37	24	59	118	62	252

Table 4.3: Influence of the backtracking scheme settings

$\#resActivities$	10		15		20	
n	TST	$t\ [ms]$	TST	$t\ [ms]$	TST	$t\ [ms]$
50	221	15	221	14	223	16
200	663	110	662	112	662	115
500	1500	470	1497	484	1495	501
1000	2677	1140	2672	1325	2665	1578

Table 4.4: Influence of the number of activities per resource

number of backtracking steps, denoted as $\#backtracks$, is calculated as a multiple of the number of activities and for the evaluation, three values have been used: $\#backtracks = \{0.1 \cdot n,\ 0.3 \cdot n,\ 0.5 \cdot n\}$. The overall test results are depicted in Table 4.3 where n denotes the number of activities, $feas$ determines the number of feasible solutions (out of 100) found by the STOAL algorithm and $t\ [ms]$ denotes the average solution time in milliseconds. Based on the results in Table 4.3 we can conclude that the optimal number of backtracking steps lies below half the number of the activities. A further increase of the number of backtracks does not increase the number of feasible solutions.

Second, the influence of the maximal number of activities per resource within a time window, denoted as $\#resActivities$, on the solution quality is determined. The same four datasets as in the previous case are used and three different values are considered: $\#resActivities = \{10,\ 15,\ 20\}$. Table 4.4 contains the results of the evaluation, TST determines the average value of the objective function. For smaller instances, there is no significant influence of the number of activities per resource on the solution quality. For larger instances, the higher number of activities per resource leads to a better solution quality while there is also a minor growth in the solution time.

Finally, the influence of the number of the local search procedure repetitions, denoted as $\#repetitions$, on the solution quality is studied. Again, the same datasets as in both previous experiments are used. As for the previous parameters, three different values are considered: $\#repetitions = \{1,\ 2,\ 4\}$. The results in Table 4.5 show the high importance of the number of the sliding windows repetitions on both the solution

$\#repetitions$	1		2		4	
n	TST	$t\ [ms]$	TST	$t\ [ms]$	TST	$t\ [ms]$
50	224	7	221	14	220	29
200	668	58	662	112	659	213
500	1504	253	1497	484	1491	956
1000	2692	685	2672	1325	2659	2428

Table 4.5: Influence of the sliding windows repetitions

quality and the solution time. Naturally, the higher number of repetitions results in a better quality while the increase in the solution time is more or less linear with respect to the number of repetitions.

4.5 Conclusion

The content of this chapter fills the gap in the literature, where only very few pieces of work have been dedicated to scheduling problems with setup times as a part of the criterion. The setup times are usually considered only as a constraint. The proposed innovative model combines the RCPSP problem with the alternative process plans and the criterion to minimise the total setup time in the schedule. Furthermore, The model includes the release time and deadline for each activity and the non-negative start to start time-lags for precedence constrained activities. For such a model, of the studied problem, the mathematical formulation, using the mixed integer linear programming (MILP), is proposed.

The two-phase heuristic algorithm is then developed to solve the large instances of the considered problem. The goal of the algorithm first phase is to find any feasible solution and the second phase, based on the time separation of the schedule, is dedicated to improve the existing schedule in terms of the total setup time. The STOAL algorithm is compared with the existing approaches for similar problems. The experiments show a very good performance of the STOAL algorithm in both the quality of the solutions and the running time. Finally, various settings of the algorithm and their influence on the obtained results are evaluated using the instances with up to 1000 activities.

Chapter 5

Minimization of the total production cost for the RCPSP-APP

The main motivation for this research is the production process optimisation in a printing company, where the scheduling problem involves alternative process plans and the goal is to minimise the total production cost given by both the processing costs of production operations and the penalties caused by late jobs. In our approach, both parts of the objective function are optimised simultaneously using the concept of alternative process plans. We propose a model and a solution approach that covers alternative process plans as well as the realistic criterion composed of two different parts. To cover the needs of the printing production, we consider resources with non-unary capacities, sequence dependent setup times and generalised temporal constraints in the form of non-negative start to start time lags.

In literature, there are only a few attempts to introduce a tardiness-based objective function into the area of the resource constrained project scheduling problem. Ballestín et al. (2006) studied the RCPSP where the criterion is to minimise the total tardiness of all activities. Vanhoucke et al. (2001) proposed a branch-and-bound algorithm for the RCPSPWET problem that is a resource constrained project scheduling problem with the minimisation of the total weighted earliness-tardiness as a criterion. Franck and Schwindt (1995) mentioned a MRCPSP with the objective to minimise the sum of the earliness and tardiness values. Pinedo and Singer (1999), Essafi et al. (2008), Zhou et al. (2009), Bülbül (2011) and Zhang and Wu (2011) dealt with the job shop scheduling problem, with the total weighted tardiness as an objective function. Naderi et al. (2009) focused on the flow shop problem where the objective function is the minimisation of the total weighted tardiness.

The presence of the cost of activities in the scheduling is mainly represented by the discrete time/cost trade-off problem (DTCTP) where each activity has a finite set of modes given by the duration and processing cost. Shabtay and Steiner (2007) published a survey dedicated to scheduling problems with discretely controllable processing times of activities. Vanhoucke and Debels (2007) presented a metaheuristic solution procedure for the DTCTP and based on the computational results, the authors concluded that, due to the efficient character of the exact algorithm of Demeulemeester et al. (1999), the metaheuristic solutions for the DTCTP can not compete with the truncated solutions found by the exact algorithm.

The first contribution of this chapter is the novel scheduling problem, where the RCPSP problem with alternative process plans (RCPSP-APP) is enhanced by the realistic criterion composed of two different parts, namely the total weighted tardiness and the total processing costs. The combination of the alternative process plans with the objective function reflecting two different sources of the production costs is a natural step towards the demands of the modern production. The approach to minimise the makespan for whatever price used in Chapter 3 can be applied only for special cases with the hard temporal constraints where the main goal is to find any feasible schedule. In this chapter, we consider the general case where the goal is to meet the customer demands (due dates) and, in the same time, minimise the processing costs. The inclusion of both demands in the objective function is supported by the presence of the alternative process plans, which allow to reflect the cost of the actual selection in a straightforward manner.

The IRSA algorithm proposed in Chapter 3 cannot be directly applied to solve the problem considered in this chapter since it does not include due dates and processing costs of activities. Furthermore, the search method in the IRSA algorithm is rather straightforward and strongly dependent on the temporal constraints that go hand in hand with the makespan criterion. Therefore, we use two different heuristic algorithms to solve the considered problem. The second contribution of this chapter is the adaptation and the detailed comparison of two algorithms, while each of them uses unique model and search strategies. To prove the effectiveness of the algorithms, the job shop instances presented in Bülbül (2011) are used as well as the instances of the integrated process planning and scheduling presented in Shao et al. (2009).

The chapter is organised as follows: The problem statement, including the mathematical model, is given in Section 5.1. Section 5.2 is dedicated to the description of two heuristic algorithms for the considered problem. The performance evaluation of both algorithms is given in Section 5.3 and Section 5.4 concludes the work.

5.1 Problem statement

The definition of the problem studied in this chapter is based on the problem statement in Section 2.1. In addition to this, each activity $i \in \mathcal{A}$ has a due date $d_i \geq 0$, tardiness cost $w_i \geq 0$ and processing cost $c_i \geq 0$. All the new activity parameters are reflected in the objective function. In this chapter we consider only non-

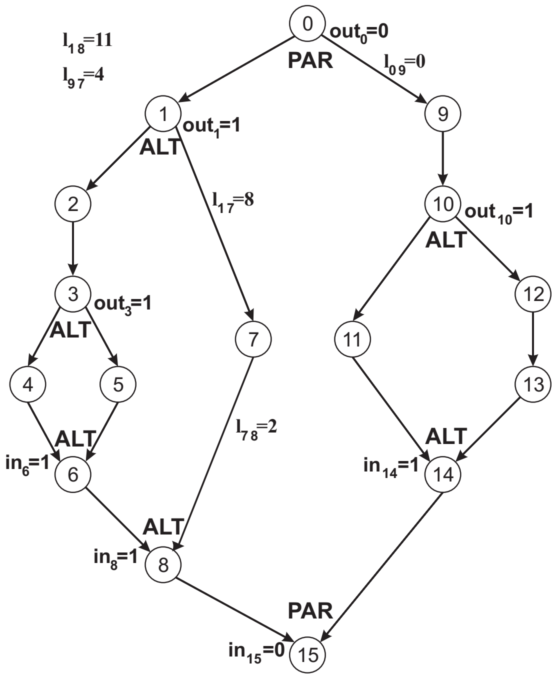


Fig. 5.1: Example of the NTNA instance

negative time-lags $l_{ij} \geq 0$ for all $(i, j) \in \mathcal{A}$; if there is no temporal constraint, then $l_{ij} = -\infty$. The goal of the scheduling is to minimize the total production cost equal to the sum of the total processing costs and the total weighted tardiness $TPC = \sum c_j \cdot v_j + \sum w_j \cdot T_j$ where $T_j = \max(s_j + p_j - d_j, 0)$ is the tardiness of an activity. The considered problem can be denoted as $PS|nestedAlt, l_{ij}, ST_{SD}|TPC$ or $m1|min, \rho_j, \delta_j, nestedAlt, s_{jk}|TPC$ using the same classification schemes as in Section 2.7.

The instance of the problem considered in this chapter is depicted in Figure 5.1. Several time-lags are used to demonstrate how the temporal constraints are defined, see e.g. time-lag $l_{17} = 8$ that forces activity 7 to start at least 8 time units after the start time of activity 1. All the parameters related to the activities are also included.

5.1.1 Mathematical model

The mathematical model for the studied problem is based on the definition given in Chapter 2. In addition to this, formula (5.1) serves to determine the tardiness of each activity. If the activity i is rejected ($v_i = 0$) then the tardiness $T_i = 0$. Note that there is no particular rule for the start times of the rejected activities. A constraint to force the start times of the rejected activities to be equal to zero could be added into the model but it would be of no benefit for the solution. Therefore, the actual values of the start times for the rejected activities are let to be decided by a particular solver search method. The goal is to minimise the total production cost given by the total processing cost and the total weighted tardiness.

$$\text{Min.} \sum_{\forall i \in \mathcal{A}} c_i \cdot v_i + \sum_{\forall i \in \mathcal{A}} w_i \cdot T_i$$

subject to:

$$(2.1) - (2.12)$$

$$T_i \geq s_i + p_i - d_i - UB \cdot (1 - v_i) \qquad \forall i \in \mathcal{A} \qquad (5.1)$$

5.2 Heuristic algorithms

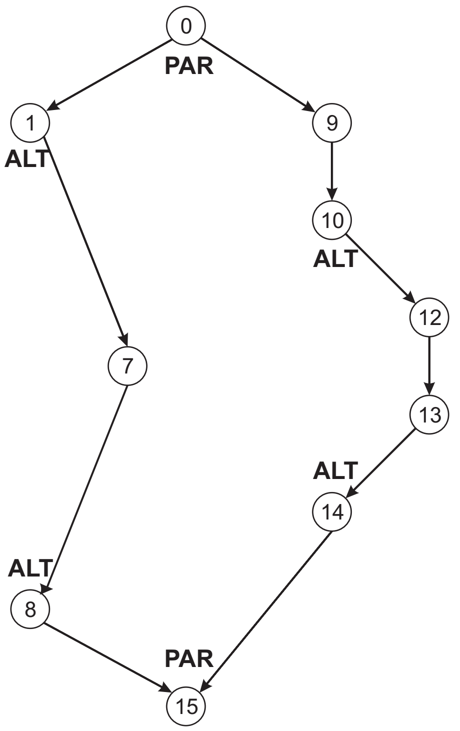
To solve large scale instances of the problem considered in Section 5.1, we have developed two heuristic algorithms, namely the discrete differential evolution (DDE) algorithm and the scatter search (ScS) algorithm. Both algorithms are population-based methods and they can search a large solution space while the local search procedures are used to improve the quality of the solutions found. However, each algorithm uses different solution representations and search strategies.

5.2.1 DDE algorithm

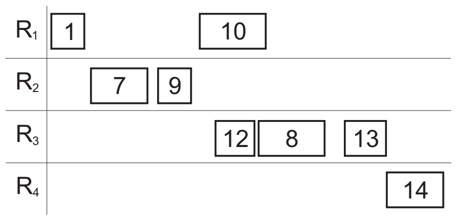
The DDE algorithm used for the solution of the problem considered in this chapter is inspired by the work of Tasgetiren et al. (2009) who used DDE to minimise the total weighted tardiness for the single machine problem. The basic principle of the DDE algorithm is similar to a genetic algorithm - first, the initial population has to be established and then, mutation and crossover operators are performed for the selected individuals and for a defined number of iterations. However, there are some important differences. The emphasis is put on the quality of individual members of the solution population. Therefore, a local search, using the knowledge of the problem, is used to improve individuals. Furthermore, all operations (mutation, crossover and local search) are performed as incremental modifications over an existing schedule instead of generating the schedule from scratch after any change.

5.2.1.1 Individual representation

Each individual represents one schedule, i.e. it contains information about the selection, start times and resource assignment of activities. We intend to utilize local search procedures and the mutation and crossover operators will also employ some routines from the local search procedures. Therefore, the representation of an individual should be designed such that these modifications of the schedule will be performed



(a) Process plan



(b) Solution

Fig. 5.2: Example of solution representation

in an effective way. Furthermore, since the considered problem includes precedence relations, the feasibility of each schedule modification has to be verified as well. For this purpose, two representations of a schedule are used simultaneously.

First, an *activities order* of all selected activities is used. An activities order is an ordered set of selected activities from which the schedule is obtained by the serial schedule generation scheme (see Kolisch, 1996). A feasible activities order is an order that results in a feasible schedule, with respect to both resource and temporal constraints. In other words, a feasible activities order is an order such that if a serial generation scheme is applied while scheduling activities as soon as possible, the resulting schedule is time and resource feasible. An activities order is assigned to each individual.

Second, a set of resource sequences is assigned to each individual. A *resource sequence* $\psi_k = \left(i \in \mathcal{V} : v_i = 1 \wedge \mathrm{R}_i^k > 0\right)$ is an ordered set of selected activities assigned to resource type k. If activity j is the next element in ψ_k after activity i, then j is called a direct resource successor of i and, vice versa, i is called a direct resource predecessor of j. A resource sequence can be understood as a subset of the activities order such that all activities in a resource sequence are assigned to the same resource type. The activities in the resource sequence are in the same order as in the activities order representation. For each resource type, there is a corresponding resource sequence for each individual.

The advantage of simultaneous utilization of both representations is that an activities order serves for a fast and effective feasibility check while the resource sequences are used to eliminate inefficient local modifications of the schedule in mutation and crossover operators and local search methods. A set of activities corresponding to one process plan is depicted in Figure 5.2a and a schedule is shown in Figure 5.2b. The activities order representation of such a schedule is $(0, 1, 7, 9, 10, 12, 8, 13, 14, 15)$ and the representation by the resource sequences is $\{\psi_1 = (1, 10), \psi_2 = (7, 9), \psi_3 = (12, 8, 13), \psi_4 = (14)\}$. The dummy activities 0 and 15 are not included in any resource sequence since they have zero resource demand.

5.2.1.2 Feasibility testing

A typical operation performed in the DDE algorithm described in this chapter is the rescheduling of an activity in the existing schedule from one resource position to another. Representation of a solution only by an activities order would result in many computational operations that change the activities order but not the solution itself, which is very inefficient. Therefore, a set of resource sequences is used to determine a set of possible schedule modifications imposed by a given activity. Incremental updates of an activities order, maintaining its feasibility, are used to check which resource positions are feasible for an activity with respect to the existing schedule as described in the following text.

Let $\mathcal{V}^S \subseteq \mathcal{V}$ be a set of all selected activities in some schedule. Let G_S^{temp} be a directed graph with nodes $V(G_S^{temp}) = \mathcal{V}^S$ and edges $E\left(G_S^{temp}\right) = \left\{(i,j) \in \mathcal{V}^S \times \mathcal{V}^S : l_{ij} \geq 0\right\}$. Furthermore, let G_S^{res} be a directed graph with nodes $V(G_S^{res}) = \mathcal{V}^S$ and edges $E\left(G_S^{res}\right) = \{(i,j) \in \mathcal{V}^S \times \mathcal{V}^S : i$ is a direct predecessor of j in the resource sequence$\}$. Finally, let G_S be a directed graph with nodes $V(G_S) = \mathcal{V}^S$ and edges $E\left(G_S\right) = \left\{E\left(G_S^{temp}\right) \bigcup E\left(G_S^{res}\right)\right\}$. The nodes of G_S correspond to all the selected activities and the edges of G_S represent all the temporal constraints and the constraints imposed by the sequencing of activities on resources. A feasible activities order, corresponding to a feasible schedule, is an assignment of a unique number top_i to each node $i \in V\left(G_S\right)$ such that $top_i < top_j$ for each pair of activities $(i,j) \in E\left(G_S\right)$. In other words, graph G_S corresponding to a feasible activities order has to be acyclic, since a feasible activities order is equivalent with a topological order of nodes $V\left(G_S\right)$.

For fast detection of infeasible schedule modifications, an algorithm for maintaining a topological order under edge insertions published by Spaccamela et al. (1996) is used as follows: Two edges representing the old resource precedences for activity

i are removed and two edges representing the new resource precedences are added to G_S. The new solution results in a feasible schedule if and only if G_S is acyclic, i.e. a feasible topological order can be found for vertices $V(G_S)$. Removal of an edge (see Spaccamela et al., 1996) does not influence the topological order at all. Therefore, only the addition of edges has to be checked with respect to the feasibility of the topological order. To detect whether the addition of an edge will result in a cycle of the graph, only a part of the topological order has to be explored (and possibly updated).

For the purpose of feasibility testing inside the DDE algorithm, we define the method $reinsertActivity(i, old, new)$ which returns true if activity i can be reinserted from resource position old to resource position new with respect to the current solution. In case of a feasible reinsertion, the activities order is updated as well as G_S and the appropriate resource sequence. Otherwise, the function returns false. An amortised running time of $reinsertActivity$ method is $O(N)$ for each edge insertion where N is the number of activities in G_S.

5.2.1.3 Main loop

The pseudo-code of the main loop of the DDE algorithm is depicted in Algorithm 6. First, an initial population is found (see Section 5.2.1.4) and then the mutation (Section 5.2.1.6) and the crossover (Section 5.2.1.7) are performed for each individual for a given number of iterations. Furthermore, a local search, described in Section 5.2.1.5, is performed after each mutation and crossover to improve the quality of each individual. A reference individual for the crossover function is selected using the tournament method, where a certain number of individuals are randomly selected from the population and the best one is given as the reference individual. In this chapter, $\lceil n/4 \rceil$ of individuals are randomly selected from the population. To establish a new generation, the best individual from the following triplet is chosen for each member of the current generation: current *individual*, *mutant_individual* (resulting from the mutation) and *trial_individual* (resulting from the crossover).

5.2.1.4 Initial population

A serial schedule generation scheme (see Kolisch, 1996) is used to establish an initial population. There are two reasons to select the serial scheme: First, it has been proven that utilization of the parallel scheme may lead to non-optimal solution even if the optimal order of the activities to be scheduled is used. The second reason is that using the parallel scheme, the initial solutions would be more similar to each other. On the contrary, the serial scheme adds only one activity into the schedule in each iteration, which leads to more diverse solutions (with randomized selection). Since we need to explore as much solution space as possible, the set of diverse solutions is always more profitable. In the beginning, a set of ready activities V^R is established and the objective value is set to $obj = 0$. Then, one activity is randomly selected from the

Algorithm 6 DDE main loop

```
Establish initial population
best_individual = best of initial population
$for\ iteration = 1 \ldots number\_of\_iterations$
   $for\ each$ individual in the population
      mutant_individual = $mutate$(individual)
      mutant_individual = $localSearch$(mutant_individual)
      reference_individual = $tournamentSelect$(population)
      trial_individual = $crossover$(mutant_individual, reference_individual)
      trial_individual = $localSearch$(trial_individual)
      Select best individual of {individual, mutant_individual, trial_individual}
      Add the best one to the next generation
   $end\ for$
   If the new global best solution is found, assign it to best_individual
$end\ for$
Return best_individual
```

set of ready activities in each iteration of the generation scheme. Once activity i to be scheduled is found, it is marked as selected and its start time is computed as the maximum from two values - the start time with respect to the temporal constraints and the start time with respect to the resource constraints. In order to calculate the start time with respect to the resource constraints, R_i^k (for $k \in \mathcal{R} : \mathrm{R}_i^k > 0$) resource units is assigned to activity i in the first place. To fasten the DDE algorithm, a variable $z_{ivk} \in \{0, 1\}$ is substituted by a variable $\delta_i \in \mathbb{Z}^+$ such that $z_{ivk} = 1$ if and only if $v_i = 1$, $\mathrm{R}_i^k > 0$ and $\delta_i \leq v < \delta_i + \mathrm{R}_i^k$; $z_{ivk} = 0$ otherwise. Instead of assigning particular units of a resource for each activity, only the first unit δ_i of a resource used by activity i is defined and the rest of the activity's resource demand is assigned to the consequent units of a resource. In other words, an activity is assigned to a consecutive set of resource units defined by the first assigned unit δ_i only. Once a set of resource units is assigned to activity i, its start time is calculated as the minimum start time that fulfills the constraint (including setup times) for each assigned resource unit. After scheduling activity i, its tardiness can be immediately calculated as $T_i = \max(s_i + p_i - d_i, 0)$. Consequently, the value of the objective function is updated such that $obj = obj + w_i \cdot T_i + c_i$.

The set of ready activities is then updated. Activity i is removed from V^R and if there is an alternative branching at the output of its direct predecessor in NTNA, a depth first search procedure is performed to find all activities that cannot be set as selected without the violation of rules for the selection of activities (see Constraints

(1)-(4)). Such activities are removed from V^R and marked as rejected. Finally, activities that become ready after scheduling of activity i are added to V^R. Activity i becomes ready if it is not marked as rejected and all the activities $\{j \in \mathcal{V} : l_{ji} \geq 0\}$ are already scheduled or marked as rejected.

If V^R is an empty set, the initial solution is completed and the schedule generation ends. The solution is, so far, represented by the resource sequences only. Therefore, graph G_S is created as described in Section 5.2.1.2 and the initial topological order for activities (vertices of G_S) is found (see e.g. Korte and Vygen (2000)).

5.2.1.5 Local search

The local search method used in this chapter is inspired by the iterated insertion scheme, called RIS, published by Tasgetiren et al. (2009). The basic principle is to find the best resource position of an activity in the fixed order of all other activities. For this purpose, an activity is put on each feasible position in such a sequence and the position with the best value of the objective function of the whole solution is kept. Such a search is performed for each activity in the schedule. The local search only modifies the sequencing of the selected activities on the resources, the selection of the activities itself is not changed at all. Therefore, it only optimizes the total weighted tardiness part of the criterion.

Since we consider a set of non-unary resources, the RIS method is adapted so that it searches for the best activity position in an appropriate resource sequence instead of the whole sequence of the selected activities represented by the topological order. Moreover, due to the presence of the precedence constraints, each activity reinsertion has to be checked with respect to the feasibility of the schedule. Therefore, method $reinsertActivity\,(i, old, new)$ (see Section 5.2.1.2) is used every time activity i is moved from resource position old to another resource position new in resource sequence ψ_k. If the reinsertion of an activity is feasible with respect to the current schedule, the start times and the first assigned units of the activities are updated and the total weighted tardiness of the schedule is simultaneously updated as well. Since the re-generation of the whole schedule is time consuming, we update the start times and the first assigned units only for activities that are actually influenced by reinserting an activity while the start times and first assigned units of all other activities remain unchanged.

5.2.1.6 Mutation

The local search method RIS, described in the previous section, is dedicated to improve the total weighted tardiness part of the objective function only without changing the selection of activities. To explore the solution space with a different selection of activities as well, both the mutation and crossover are mainly focused on the selection change.

To change a selection of activities, one alternative branching is randomly selected in the mutation. A random selection of alternative branching is operated only over the set of the actual selected branchings, i.e. one of their branches is currently selected. Then a selection of activities in the branching is randomly changed, i.e. all activities in the selected branch are unscheduled and a different branch is randomly selected and all activities in the new branch are being scheduled. If another alternative branching is nested in the newly selected branch, a random branch is always selected. Each activity is scheduled on the first feasible position into the current schedule. The feasibility of scheduling activity i is tested by the $reinsertActivity\,(i, -, new)$ method, where the initial position of the activity is not given and the position new is iterated over the resource sequence. Once a feasible position is found for all new activities in the schedule, the schedule is updated in the same way as described in the previous section, i.e. start times and assigned units are updated only for the actual influenced activities.

5.2.1.7 Crossover

The crossover operator in our approach is mainly dedicated to change the selection of the activities. Two given individuals - the *original* individual and the *reference* individual - are combined and the new *trial* individual is generated. To generate the *offspring,* there are two possibilities based on the selection of the *original* and *reference* individuals.

If the activities selection of both parent individuals is the same, then the standard one-point crossover can be applied (see e.g. Shao et al., 2009). If the selection of activities differ between the *original* and *reference* individual, the selection of the *offspring* is established as follows: For each alternative branching where the *original* and *reference* individuals differ, the branch with the lower contribution for the value of the objective function is selected. This way, the solution is modified towards the selection with the lower value of the objective function. Each of the activities, which is added to the schedule by the crossover operator, is scheduled to the first feasible position in the resource sequence demanded by an activity.

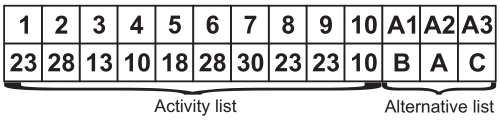
5.2.2 Scatter search algorithm

Scatter search (ScS) is a population-based meta-heuristic, proposed by Glover et al. (2000), in which solutions are intelligently combined to yield better solutions. The scatter search method involves deterministic procedures that can include problem specific knowledge and can, therefore, be implemented in a variety of ways and degrees of sophistication. In this chapter, the scatter search procedure for the problem under study makes use of the biased random sampling in order to obtain a diverse initial population of solution vectors. A solution vector is represented by two lists: an activity list, which determines the sequence in which the activities are scheduled, and an alternative list, which determines which alternative will be chosen. Solution vectors are transformed in schedules using the serial generation scheme. To combine the existing solution, a solution improvement method that calculates the possible theoretical improvement is applied in order to obtain improvements in the objective function. Two local search procedures, one focusing on the activity list and one focusing on the alternative list, are applied - with predefined probability - on the generated schedules in order to decrease the total production cost.

The main difference with the DDE algorithm presented in Section 5.2.1 is threefold: a) the individual representation of the schedule. Where the DDE algorithm is only representing the selected activities in its activity list, the ScS algorithm takes all activities into account, even if these are not selected. Therefore, a second list (the alternative list) is used and needed in order to indicate which alternatives will be chosen. b) the search process. While the DDE algorithm is mainly based on a random combination of individuals, the ScS algorithm is known to intensify its solution by only combining the solutions that are part of the reference set of the best or diverse solutions. c) the local search process. The process in the DDE algorithm makes use of an iterated insertion scheme, which means that each individual will be evaluated several times, while in the ScS algorithm the process to minimise the overall tardiness cost is only applied once, namely the one that will make the largest theoretical improvement in the objective function.

5.2.2.1 Individual representation

In the scatter search algorithm, a population is represented by two lists: an activity list and an alternative list. The activity list determines the sequence in which the activities will be scheduled and is represented by a list of priorities. The alternative list indicates which alternative mode will be chosen for each alternative branching. In Figure 5.3, an example of an individual representation is given. The length of the activity list is determined by the number of activities in the project (n), the length of the alternative list by the total number of alternative branchings ($|A|$).



1	2	3	4	5	6	7	8	9	10	A1	A2	A3
23	28	13	10	18	28	30	23	23	10	B	A	C

Activity list
Alternative list

Fig. 5.3: Individual representation

The modes chosen in the alternative list determine the activities that are being selected. Rather than changing the length of the individual vector depending on the alternatives chosen, each activity in the activity list gets a priority value. The same is true for the nested alternative branchings in the alternative list. Even though one (or more) nested alternative branchings are not chosen, the alternative itself is given an alternative mode. This means that the total length of the representation is always equal to $n + |A|$. Such a representation makes it easier to deal with than the representation with varying lengths. Moreover, it enables the procedure to maintain historical data that can be used in later generations, although it is not used in the current one.

5.2.2.2 Scatter search procedure

The scatter search has a generic structure as outlined in Algorithm 7.

Algorithm 7 Scatter search main loop

```
Diversification Generation Method
best_individual = best of initial population
While Stop Criterion not met
    Subset Generation Method
    Solution Combination Method
    Improvement Method
    Reference Set Update Method
End While
Return best_individual
```

5.2.2.2.1 Diversification Generation Method In this first step, a pool P of $Psize$ solution vectors is generated. In order to obtain a diversified initial population of solution vectors, a random priority is assigned to each activity. The priority value varies between 0 and 100 and will determine the sequence in which the activities will be scheduled, taking into account the precedence relations. It is assumed that activities with a lower priority will be scheduled first. For the generation of the alternative mode list, an alternative mode is chosen for each alternative in such a way that the probability of assigning a mode to that alternative is inversely proportional to the number of times the alternative mode is already chosen. In this way, a diversified initial pool of $Psize$ solution vectors is generated.

5.2.2.2.2 Subset Generation Method Each solution vector is then evaluated by using a serial schedule generation scheme (SGS), which translates the solution vector into a schedule S, taking into account the precedence and resource constraints. Based on the fitness function of each solution, two diverse populations are conducted from the pool P of solution vectors: a set B_1, with the b_1 best solutions of the solution set P and a set B_2, with b_2 diverse solutions. For the subset B_1, a threshold t_1 on the minimal distance between the elements is imposed in pursuit of diversity. The subset B_2 contains the b_2 best solutions from $P \backslash B_1$ that are sufficiently distant from the elements of B_1. The diversity in B_2 is achieved by a threshold t_2 on the smallest distance to any element in B_1 with $t_2 > t_1$. The distance between two solutions is measured as follows:

$$d_{p_1,p_2} = \sum_{i=1}^{n} \begin{cases} 0 & \text{if } seq_i^{p_1} = seq_i^{p_2} \\ 1 & \text{otherwise} \end{cases}$$

Where seq_i^p indicates the sequence number of activity i according to the priority list of population element p. If there are less solutions in B_2 than the predefined number b_2, the set B_2 is filled up with randomly generated schedules, according to the procedure explained in Section 5.2.2.1.

5.2.2.2.3 Solution Combination Method Once the two reference subsets are generated, a new pool of solutions is created by combining pairs of reference solutions in a systematic and controlled way. New solutions are created by combining two elements from the B_1 and B_2 reference set. First, each pair in B_1 is combined to generate two children. In the solution combination phase, the two selected population elements produce a new offspring which inherit parts of their parents characteristics. A new child is generated by randomly selecting an activity r ($r \in [0, n]$), copying all the activities $[0, r]$ from the first solution vector and copying all the other priorities from the second element.

The solution combination method for the alternative list is based on the Harmony Search procedure Geem et al. (2001) and uses a frequency matrix of the alternative lists in the B_1 subset. To assign an alternative mode to an alternative, a random subset element is chosen and the mode for the alternative of that element is assigned to the new element alternative. In order to maintain enough diversification, a probability is used to randomly assign an alternative mode to an alternative.

Next to the combination of the pairs of B_1 elements, offsprings are constructed using the same subset generation method from one element from B_1 and one from B_2. Choosing the two reference solutions out of the same cluster stimulates intensification, while choosing them from different clusters stimulates diversification.

5.2.2.2.4 The Improvement Method After the solution combination method, each new solution vector consists of a newly generated activity list and a newly generated alternative list. Before the evaluation of this new vector is executed, a solution improvement method is applied in order to obtain improvements in the objective function. The applied procedure can be described as follows:

- First, for every pair of activities in the project which are not interrelated by precedence relations, the theoretical improvement in the objective function is calculated if the activities are swapped. This theoretical improvement is calculated as follows:

$$I_T = C_{current} - C_{swap} \tag{5.2}$$

with I_T the theoretical improvement, $C_{current}$ the tardiness cost related to both activities and C_{swap} the tardiness cost if both activities where swapped and stand alone (given the start date of the first activity and the known due dates). This C_{swap} does not take the precedence relation into account, that is why we call it the possible theoretical improvement. All positive theoretical improvement values, as well as the swapped activities are stored.

- Second, one swap is chosen randomly using a weighted probability function. The probability that a swap is chosen is proportional to the theoretical improvement of the swap: the higher the theoretical improvement, the larger the chance that the swap will be chosen.
- Finally, this swap is applied to the solution vector and the schedule generation scheme is applied to calculate the improved solution vector.

The final improvement that will be found in the objective function after this improvement method will not always be as positive as the theoretical improvement has predicted. This is due to the precedence relations of both of the swapped activities with other activities in the schedule which are not taken into account during the calculation of I_T.

Since this improvement method is CPU demanding, especially for an increasing number of activities, a quick/accelerated improvement method is used in the procedure. This method randomly chooses two activities until a positive theoretical improvement value is obtained. If after n consecutive attempts no positive I_T is found, no further changes are applied.

5.2.2.2.5 Reference Set Update Method Each new and improved solution vector is then evaluated and added to the pool P of solution vectors. Out of this pool, the two reference subsets are again generated according to the procedure described above. The algorithm is applied as long as the stop criterion is not met.

5.3 Evaluation

To prove the effectiveness of the proposed solution methodology, we use three data sources for the evaluation of both algorithms. First, the job shop instances proposed by Pinedo and Singer (1999) are used to prove that the algorithms perform well for the total weighted tardiness (TWT) criterion. Second, the instances of the integrated process planning and scheduling proposed by Shao et al. (2009) are used to prove the effectiveness for the problems with alternative process plans. Finally, we generate random instances of the problem defined in Section 5.1 to compare the performance of both algorithms on instances with various structure. Furthermore, the proposed metric for the instances characterisation is used to find the important properties with respect to the solution quality of both algorithms. Both algorithms were implemented in C# language and the experiments were performed on a PC with 2x Intel Core 2 Quad CPU at 2.83GHz with 8GB of RAM.

5.3.1 Mathematical model complexity

Exact solution of the mathematical model presented in Section 5.1.1 can be used only for very small instances. In our previous research (see Chapter 3), it has been shown that the MILP solver is able to solve the instances with 30 activities within one minute. For 50 activities, the number of instances solved to optimality decreased to 60%. The proposed model comprised hard temporal constraints and straightforward objective function - makespan. The absence of hard temporal constraints and the composite objective function considered in this chapter make the exact solution methods far less effective. We have conducted several experiments with IBM ILOG CPLEX MILP solver and the results showed that the solver was able to optimally solve, within one minute time limit, only 65% of instances with 20 activities. For 30 activities, there was only 28% of optimal solutions.

Next to the experiments with the MILP solver, we have used the constraint programming solver as well; the results were presented in Čapek et al. (2013). Due to the restarted search procedure in the ILOG CP solver, the number of instances for which the optimal solution has been found and proved is rather low - 39% for instances with 20 activities and 11% for instances with 30 activities. On the contrary, the average value of the objective function was about 20% better than for the MILP solver within the same time.

To obtain the results of the same overall quality as the heuristic algorithms proposed in this chapter, the exact solvers need $10\times$ more CPU time for instances with 20 activities. For the instances with 50 activities, the exact solvers are no longer able to compete with heuristic approaches even if the time limit is increased to 10 minutes per instance.

5.3.2 Job shop problem

The job shop problem with the TWT criterion represents a special sub-problem of the problem considered in this chapter. Therefore, we can solve the instances presented in Pinedo and Singer (1999) by both of our algorithms. There are three similar datasets, each containing 22 instances with 10 jobs and 10 machines. All datasets consist of the same instances, but with the different assignment of the due date values. The results of our algorithms are compared with the results published in Bülbül (2011) who used a hybrid shifting bottleneck-tabu search heuristic. Table 5.1 summarises the results over each dataset for the DDE algorithm (DDE), the scatter search algorithm (ScS) and the results published by Bülbül (2011) for the G/MAI algorithm with setting (2,2,2,1,1,1,1,1,1,1)-RF (Ref).

The same measurements as in Bülbül (2011) are used for our algorithms. First,

		DDE			ScS			G/MAI
		best	worst	avg	best	worst	avg	
Set 1	Total gap [%]	2.69	13.24	7.92	26.94	31.95	28.94	12.49
	Optimal	12	3	-	0	0	-	3
	New best	6	1	-	0	0	-	3
	Time [s] / NoS	33.41 / -			29.67 / 65 000			31.53 / -
Set 2	Total gap [%]	5.51	36.34	19.89	79.20	99.35	88.21	18.97
	Optimal	18	5	-	0	0	-	11
	New best	0	0	-	0	0	-	0
	Time [s] / NoS	32.04 / -			29.82 / 65 000			30.11 / -
Set 3	Total gap [%]	10.48	55.99	31.07	118	158.69	145.35	37.20
	Optimal	17	11	-	6	6	-	13
	New best	1	0	-	0	0	-	0
	Time [s] / NoS	18.51 / -			29.81 / 65 000			26.32 / -

Table 5.1: Comparison with Bülbül (2011)

we measured the total gap, for each dataset, denoted as $Total\ gap$ that is the percentage difference between the total sum of the objective values of our results and the objective values stated in Pinedo and Singer (1999). Second, the number of solutions with the value of the objective function equal or less ($Optimal$) and strictly less ($New\ best$) than stated in Pinedo and Singer (1999) is calculated. Finally, the average running time of the algorithms in seconds denoted as $Time$ is showed. The number of generated schedules for the ScS algorithm is indicated as NoS. Both algorithms presented in this chapter use some kind of randomization and therefore, each instance is solved five times (using different random seeds) by each algorithm and the best ($best$), the worst ($worst$) and the average (avg) value for each instance is observed. The number of the optimal and new best solutions are not given for the average results of both algorithms, since this information is identical to the worst results (if the value is optimal in the worst case, it has to be the same for all algorithm runs).

The results show very good performance of the DDE algorithm, despite the fact that the job shop scheduling is a very specific sub-problem of the problem considered in this chapter. The average results of the DDE algorithm outperforms the results of the reference algorithm presented in Bülbül (2011) and the best values are very close to the optimal ones. For the first set of instances, the average result of the DDE algorithm is by 4.57% closer to the optimum; for the second set, DDE is worse by 0.92%; for the third set, the DDE algorithm is closer to the optimum by 6.13% than the reference algorithm presented in Bülbül (2011). Moreover, the algorithm was able to find 7 new best solutions compared to the values presented in Pinedo and Singer (1999). Comparison of computational times is not fully representative, since Bülbül (2011) implemented the algorithm in Visual Basic. Nonetheless, we believe that the results in Table 5.1 represent an adequate proof that the DDE and ScS algorithms are more than competitive while the total weighted tardiness criterion is considered in the area of the RCPSP problems.

5.3.3 Integrated process planning and scheduling

The integrated process planning and scheduling (IPPS) problem studied in Shao et al. (2009) is used to prove the effectiveness of our algorithms for the scheduling problems containing alternatives. IPPS is again a special case of the problem considered in this chapter. The goal is to select and schedule a subset of all activities based on the precedence graph containing alternative routes and alternative machine assignment such that the makespan is minimised. In Shao et al. (2009) there are six small instances (1-6) of IPPS and one bigger instance (7) obtained by joining all small instances into one graph. The makespan minimisation can be easily transformed to the minimisation of TWT value by assigning the due date equal to zero for the last node (activity) of the graph and setting the rest of the due dates to a sufficiently large value. The comparison of the reported objective values and the values obtained by our algorithms for all seven instances is depicted in Table 5.2. It should be pointed out that the objective value for the first instance indicated in Shao et al. (2009) is not possible, since the optimal value is 117 instead of 116. It can be seen by constructing the schedule according to the process plan selected in the paper, which leads to the value 117. The average solution time reported in Shao et al. (2009) is 1 second for small instances, while for the bigger one there is no solution time at all. The average running times for our algorithms is 120 ms for small instances and 17s for the bigger one with the same settings. For small instances (1-6), the algorithms always converge to the optimal value. For the bigger instance (7), the value denoted as 7 avg is the average value over ten runs of the algorithms and the value denoted as 7 $best$ is the minimal obtained value of the objective function.

Instance	1	2	3	4	5	6	7 best	7 worst	7 avg
Shao et al. (2009)	116	116	95	93	116	116	-	-	162
DDE algorithm	117	116	95	93	116	116	147	166	157
ScS algorithm	117	116	95	93	116	116	150	171	158

Table 5.2: Comparison with Shao et al. (2009)

Using the algorithms proposed in this chapter, we are able to obtain equal or better values of the objective values in a shorter time than is indicated in Shao et al. (2009). Therefore we can conclude that the solution methodology is eligible to solve the problems with alternative process plans.

5.3.4 Computational Results

To the best of our knowledge, there are no benchmark instances for the problem defined in Section 5.1. Therefore, new random instances were generated for the final performance evaluation of both algorithms. The datasets with 10, 20, 50, 100 and 200 activities per instance were generated, each dataset containing 500 random instances.

	DDE			ScS			
Dataset	f_{mean}	$best$	t_{cpu} $[ms]$	f_{mean}	$best$	t_{cpu} $[ms]$	NoS
D10	484	39	28	487	3	25	100
	484	39	52	487	3	51	200
D20	859	52	115	869	4	102	200
	858	51	237	869	3	213	400
D50	1511	46	237	1519	27	244	500
	1511	45	467	1519	19	483	1000
D100	2667	64	581	2677	67	559	1000
	2665	62	972	2677	64	993	2000
D200	4555	50	2210	4559	181	2186	2000
	4549	56	4219	4559	165	4175	4000

Table 5.3: Results for generated instances

For each instance, we run both the DDE algorithm and the ScS algorithm for 10 times and the average value of the objective function is considered for the evaluation. The overall results for both algorithms over all datasets are summarised in Table 5.3. Two configurations of both algorithms are used to solve the instances. The maximum number of schedule generation steps for the scatter search algorithm is set to 10 times the number of activities in the first case and 20 times the number of activities in the second case. The configuration of the DDE algorithm was adjusted to run for a similar time resulting in the number of individuals equal to 10 and the number of iterations between 30 and 80. The reason to use the running time as the main common measure for both algorithms is that the number of generated schedules, which can be used for the ScS algorithm, is not applicable for the DDE algorithm since there are only few generated schedules that are further updated incrementally. The time saved in the DDE algorithm thanks to the reduction of the repeated schedule generation is dedicated to the incremental schedule updates.

Each label **Dx** in Table 5.3 stands for the dataset with **x** activities per instance while the first row corresponds to the first configuration of the algorithms and the second row to the second configuration with extended stopping criterion. Column f_{mean} represents the average value of the objective function obtained by a corresponding algorithm for a given dataset. Column $best$ contains the number of instances for which the corresponding algorithm found a strictly better solution (for the average value over 10 runs). Column t_{cpu} contains the mean solution time in milliseconds. The number of generated schedules for the ScS algorithm is denoted as NoS.

As can be observed from Table 5.3, the results obtained by both algorithms are very competitive. On one hand, the DDE algorithm was able to find better results from the point of view of the objective function value. On the other hand, the number of strictly better solutions is higher for the ScS algorithm, especially with a growing number of activities per instance. Based on the results we can conclude that a further increase of the solution time for the ScS algorithm will not lead to an improvement in the results. On the contrary, the DDE algorithm showed an improvement of results with a growing solution time. Therefore, we can conclude that the convergence in the objective value is faster with the ScS algorithm while the solution time increase is more rationale for the DDE algorithm when searching for a better solution.

The evaluation of both algorithms with respect to the measured properties of instances is depicted in Table 5.4, where datasets **D50**, **D100** and **D200** are used again. For each dataset and each measured property, the t-test is evaluated while the two sets being compared are the sets of instances where the strictly best result was found by the DDE algorithm (set^{DDE}) and the ScS algorithm (set^{ScS}) respectively. The second set (set^{ScS}) for datasets **D10** and **D20** is too small for a fully conclusive t-test evaluation, those datasets are not considered in Table 5.4. As in Table 5.3, two configurations of each algorithm are tested. The rows with DDE (ScS) label contain an average value of a specific metric over all instances of a given dataset where the DDE (ScS) algorithm found better results. Rows p-$value$ then contain the results of the t-test, i.e. the significance level that the mean values of both sets are equal.

As mentioned before, a lower p-value corresponds to the more important property from the algorithm comparison point of view. Based on the results in Table 5.4, the most important properties with respect to solution algorithm are the number of alternative branchings ($\#AB$) and the number of alternative process plans ($\#APP$). For both properties, the ScS algorithm was better for higher values, i.e. for instances with more alternatives in the selection of the activities. The next property from the importance point of view is the resource constrainedness (RC), where the DDE algorithm was better for higher values corresponding to the instances with more scarce resources. For the average activity slack (AAS), the ScS algorithm was better for instances with tighter time windows of activities. The last properties where a solid importance has been observed are the total and average order strength (TOS and AOS), which are closely related both in the values and in the influence for the solution algorithm effectiveness. For both properties, the DDE algorithm was slightly better for less pre-ordered instances, i.e. those where more decisions related to orders of activities on resources are needed. The rest of the properties do not possess a significant influence for the different solution methods or the results are ambiguous.

The reason for the ScS algorithm being clearly better for the instances with a higher ratio of alternative parts is that more time of the algorithm is dedicated to travel across the solution space based on the evolutionary operators. In the DDE algorithm, a strong emphasis is put on the local search for the TWT part of the criterion and therefore, it is better for instances with less alternative process plans. The proof can also be found in the evaluation of the algorithms on the instances of the job shop problem from Bülbül (2011) where no alternatives are present and the DDE algorithm shows much better performance than ScS.

D50	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
DDE	0.44	0.49	1.85	8.85	36.83	1.26	1.87	0.72	17.48
ScS	0.45	0.53	2.09	11.80	40.37	1.22	1.93	0.61	6.93
p-value	0.66	0.23	0.48	0.47	0.18	0.83	0.80	0.02	0.17
DDE	0.45	0.47	2.00	6.89	38.05	1.22	1.89	0.72	14.24
ScS	0.46	0.52	2.00	11.78	40.47	1.32	1.95	0.64	10.58
p-value	0.60	0.21	1.00	0.27	0.38	0.62	0.82	0.17	0.57

D100	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
DDE	0.48	0.48	3.44	80.59	85.21	1.38	2.81	0.68	93.27
ScS	0.52	0.52	4.55	230.15	83.43	1.49	2.75	0.65	84.55
p-value	0.25	0.23	0.00	0.01	0.44	0.25	0.79	0.20	0.51
DDE	0.48	0.49	3.40	77.39	85.48	1.37	2.85	0.68	95.45
ScS	0.51	0.51	4.39	257.88	85.94	1.44	2.67	0.65	85.02
p-value	0.29	0.35	0.00	0.13	0.85	0.52	0.47	0.22	0.35

D200	TOS	AOS	$\#AB$	$\#APP$	$PPAct$	NL	$\#res$	RC	AAS
DDE	0.46	0.46	6.47	1903.69	167.82	1.73	2.80	0.67	265.27
ScS	0.48	0.47	8.03	7496.32	170.00	1.65	2.99	0.65	234.24
8 *p-value*	0.43	0.53	0.00	0.05	0.50	0.41	0.44	0.46	0.23
DDE	0.46	0.45	6.67	2518.59	169.40	1.72	2.80	0.67	265.74
ScS	0.48	0.48	8.04	7848.33	170.84	1.62	2.84	0.65	251.89
p-value	0.44	0.37	0.00	0.06	0.63	0.31	0.84	0.41	0.32

Table 5.4: Results of metric for new datasets

5.4 Conclusion

In this chapter, we present a new scheduling problem that combines an alternative process plans definition and a realistic objective function composed of two parts, related to both the processing costs and the meeting of the due dates. The scheduling model is based on the resource constrained project scheduling problem with alternative process plans and formulated as the integer linear programming problem. For the proposed model, two evolutionary algorithms with distinct search strategies are developed. Both algorithms showed a good performance for the related problems, especially the discrete differential evolution algorithm which is fully competitive with the existing methods for much more specialized problems.

The algorithms were evaluated using a novel metric for the characterisation of the instances properties. To find the most important properties with respect to the effectiveness of the solution methods, the Two-sample t-test for equal means is used. Such an evaluation strategy can be used for any metric or solution approach while the relative importance of a specific property can be straightforwardly derived from the result of the t-test. Consequently, the proposed evaluation method can be easily adapted to any scheduling problem where more solution approaches are to be compared.

The result of the comparison of two developed heuristic approaches is that incremental updates with a local search used in the DDE algorithm is better for problems with more parallel structure and scarce resources. On the contrary, problems containing more alternative parts (more alternative process plans) were solved better by the scatter search algorithm where the main emphasis is put on the evolutionary operators and wide travel across the solution space.

Chapter 6

Conclusion

This chapter concludes the work and summarizes the achievements with respect to the Goals and Objectives Chapter .

6.1 Main Achievements and Contributions

The first achieved contribution of this thesis lies in the novel mathematical model for the production scheduling with alternative process plans. Thanks to the utilization of the Nested Temporal Networks with Alternatives (NTNA), the model respects the natural structure of the production processes, where certain parts can be produced in more alternative ways, yielding the same final product in the end. Moreover, it preserves the assumptions and constraints of the Resource Constrained Projects Scheduling Problem (RCPSP) that is the most commonly used scheduling framework for the production scheduling problems. Consequently, the resulting RCPSP-APP (RCPSP with alternative process plans) model offers very flexible definition of the production scheduling problems while it supports the utilization of a broad variety of solution approaches for the RCPSP problems. The power of the proposed mathematical model is demonstrated on three specific problems. Although the problems differ in assumptions, constraints and objective function, the common mathematical formulas from Chapter 2 can be used for all the problems without any loss.

The second contribution is represented by the design and implementation of four different heuristic algorithms for three considered problems. The solution approach for each considered problem was selected considering the specific constraints and objective function. The solution for the problems with hard constraints and rather straightforward criterion is based on fast deterministic heuristics with limited budget (number of iterations). On the contrary, the approach for the last problem (easy to find a feasible solution but the objective is more complex) utilizes population based methods in order to explore the search space is more directions simultaneously. Due to the fact that the general problem considered in this thesis has not been studied in such an extent so far, there are no standard benchmarks available. Nonetheless, based on the results in each chapter, we can conclude that the heuristic algorithms proposed in this thesis are able to compete with the specialized algorithms for the specific subproblems. In some cases, the results were even better that the best known solutions so far. In addition to the experiments for the heuristic algorithms, the performance of two different exact methods, namely constraint programming and mixed integer linear programming, is evaluated as well.

The third contribution of the thesis is the novel evaluation metric for the instances of the problems with alternative process plans. This metric is used together with standard statistic methods to find out the important properties of the instances that have the main influence on the performance of different algorithms. The main focus is naturally paid to the structural properties of the instances, especially to the properties bound to the definition of the alternative process plans. The evaluation metric is used in Chapter 3 and Chapter 5 to distinguish the effectiveness of more solution approaches with respect to the type of the instances.

6.2 Revision of Goals and Objectives

The fulfillment of the stated goals and objectives is summarized below.

1. The goal to propose a representation for the scheduling problems with alternative processes was satisfied in Chapter 2, where the common model for the rest of the work is established. The formulation reflects the state of the art in the scheduling area and fills the gap in the existing approaches for the problems with alternatives.

2. A mathematical formulation for three studied problems based on the proposed representation is stated in Chapter 3, Chapter 4 and Chapter 5, respectively. Each chapter presents extensions and/or modifications of the common part from Chapter 2 that are specific for the current scheduling problem. In addition to the formulation itself, Chapter 3 contains the comparison of the exact solution methods, namely the MILP and CP approaches. The results showed that the effectiveness of both solvers is comparable; on the one hand MILP approach is better in proving optimal solutions, on the other hand the value of the objective function with the increasing solution time converges faster in case of CP solver.

3. The goal to develop the solution methods for large instances is fulfilled by the design and implementation of four heuristic algorithms, all implemented in the C# language. Chapter 3 describes the constructive heuristic algorithm with an un-scheduling step that is designed with intention to solve the instances with hard temporal constraints (release times, deadline and positive-negative time-lags). The algorithm is able to solve instances with 2000 activities in less than

10 seconds and when compared to the constraint programming, it consumes approximately 30x less time to achieve the solutions of the same overall quality.

The STOAL heuristic algorithm developed for the problem considered in Chapter 4 is based on the fast search for any feasible solution in combination with the local search for the time-disjunctive parts of the schedule. Thanks to the fact that the negative time-lags are not present in the problem, the STOAL algorithm is able to run even faster than the before mentioned IRSA algorithm. It is able to solve the instances with up to 1000 activities in less than 3 seconds.

Finally, there are two different population based algorithms designed for the problem presented in Chapter 5. Both algorithms are comparable in both solution time and objective function for the random generated instances and are able to solve problems with a few hundreds of activities within 5 seconds. Note that the exact solvers for the considered problem are able to effectively handle only instance with up to 40 activities; for larger instances the exact methods are no longer applicable, mainly due to the composite criterion.

4. To satisfy the objective that lies in the comparison of proposed algorithms with existing approaches from the literature, we have carefully selected at least on similar problem with available datasets for each implemented algorithm. The performance of the IRSA algorithm is first evaluated on a small dataset for a specific sub-problem from Shao et al. (2009). The results achieved by the IRSA algorithm are slightly worse in criterion but much better in the computation time. For the comparison with the second source of datasets from Kis (2003), the IRSA algorithm was extended to handle additional constraints considered in the paper. The results showed that IRSA needed only 1% of the CPU time compared to the algorithm proposed in the paper. The average value of the objective function was 15% worse in case of IRSA if compared to the best of three algorihtms proposed in Kis (2003).

   The STOAL algorithm was compared with the work of Focacci et al. (2000) using four available datasets. The CPU time is not indicated in the paper and, therefore, only values of the objective function were compared. The STOAL algorithm were able to find the results with the total setup time (TST) lower by more than 16%. It should be noted that there is almost the same deterioration for the schedule length, which was a part of the criterion in the paper but it is not in our approach. Therefore, we can conclude that on one hand the STOAL algorithm is very effective for TST criterion. On the other hand its nature does not consider other parameters of the schedule (like makespan) to be important for the scheduling process.

   The last two algorithms, dedicated to deal with the total production cost criterion, were first evaluated on the same dataset of Shao et al. (2009) as for the IRSA algorithm. Even though the original criterion was the minimization of the schedule length, both the discrete differential evolution (DDE) and the scatter search (ScS) algorithms outperformed the paper results in the criterion as well as in the computational time. Finally, the datasets from Pinedo and Singer (1999) were used and the results were compared with the approach presented in Bülbül (2011). The problem considered in the papers forms only a very specific sub-problem of the problem considered in this thesis, yet the DDE algorithm was able to find the results that are more than 3.5% better when compared to Bülbül (2011) .

5. The last goal of the thesis related to the evaluation methodology for the problems with alternative process plans is satisfied in Section 3.3.7. We have explored many properties that are related to the structure, resource environment and attributes of activities from which nine most important were extracted and used as the metric for evaluation of the instances. The relative importance of specific properties for the performance of different solution approaches (constraint programming versus heuristic algorithm) is then evaluated by the standard statistical methods. As a result, we can conclude that CP approach is performing better for the instances with more constrained resources. On the contrary, the IRSA algorithm is better if the number of alternative process plans increases. The same methodology is used also for the comparison of the two population based algorithms in Chapter 5.

6.3 Concluding Remarks

As stated in the previous section, all the goals and objectives set for the thesis were successfully achieved. The proposed model and solution approaches for three different problems with alternative process plans extends the scheduling theory by the new type of scheduling problems with a high flexibility. Based on the number of citations referencing our first published paper within a short period, it is apparent that the research in the area of alternative process plans will be dynamic in the future. There are many challenging issues - developing new algorithms, considering additional constraints, generation of new public instances etc.

Bibliography

Abdolshah, M., 2014. A review of resource-constrained project scheduling problems (rcpsp) approaches and solutions. nternational Transaction Journal of Engineering, Management, & Applied Sciences & Technologies 5, 253–286.

Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y., 2008. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187, 985–1032.

Ballestín, F., Valls, V., Quintanilla, S., 2006. Due Dates and RCPSP. Springer-Verlag New York, Inc.. volume 92 of *International Series in Operations Research & Management Science*. chapter Justification Technique Generalizations. pp. 79–104.

Barták, R., 2004. Integrating planning into production scheduling: A formal view, in: Workshop on Integrating Planning into Scheduling at ICAPS-O4, pp. 1–8.

Barták, R., Čepek, O., 2007. Temporal networks with alternatives: Complexity and model, in: Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference (FLAIRS), Florida, USA, AAAI Press. pp. 641–646.

Barták, R., Čepek, O., 2008. Nested temporal networks with alternatives: recognition and tractability, in: Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Ceara, Brazil, ACM. pp. 156–157.

Beck, J.C., Fox, M.S., 2000. Constraint-directed techniques for scheduling alternative activities. Artificial Intelligence 121, 211–250.

Blazewicz, J., Dror, M., Weglarz, J., 1991. Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research 51, 283–300.

Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J., 1996. Scheduling Computer and Manufacturing Processes. Springer-Verlag New York, Inc.

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A., 1983. Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, 11–24.

Brucker, P., 2007. Scheduling Algorithms. Springer-Verlag New York, Inc.

Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E., 1999a. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112, 3–41.

Brucker, P., Hilbig, T., Hurnik, J., 1999b. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. Discrete Applied Mathematics 94, 77–79.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107, 272–288.

Brucker, P., Kunst, S., 2006. Complex Scheduling. Springer-Verlag New York, Inc.

Brucker, P., Thiele, O., 1996. A branch & bound method for the general-shop problem with sequence dependent setup-times. Operations Research Spectrum 18, 145–161.

Bülbül, K., 2011. A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. Computers & Industrial Engineering 38, 967–983.

Capacho, L., Pastor, R., 2006. The ASALB problem with processing alternatives involving different tasks: Definition, formalization and resolution, in: International Conference Computational Science and Its Applications (ICCSA), Glasgow, UK, Springer. pp. 554–563.

Capacho, L., Pastor, R., 2008. ASALBP: the alternative subgraphs assembly line balancing problem. International Journal of Production Research 46, 3503–3516.

Capacho, L., Pastor, R., Dolgui, A., Guschinskaya, O., 2009. An evaluation of constructive heuristic methods for solving the alternative subgraphs assembly line balancing problem. Journal of Heuristics 15, 109–132.

Čapek, R., Hanzálek, Z., Bužková, L., 2013. Constraint programming and evaluation methods for scheduling with alternative process plans, in: Proceedings of the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems, LORIA Campus Scientifique. pp. 35–37.

Čapek, R., Šůcha, P., Hanzálek, Z., 2012. Production scheduling with alternative process plans. European Journal of Operational Research 217, 300–311.

Chryssolouris, G., Chan, S., Suh, N., 1985. An integrated approach to process planning and scheduling. {CIRP} Annals - Manufacturing Technology 34, 413–417.

De Reyck, B., Herroelen, W., 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 119, 538–556.

Deblaere, F., Demeulemeester, E., Herroelen, W., 2011. Reactive scheduling in the multi-mode rcpsp. Computers & Operations Research 38, 63–75.

Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38, 1803–1818.

Demeulemeester, E., Herroelen, W.S., Elmaghraby, S.E., 1999. Optimal procedures for the discrete time/cost trade-off problem in project networks. European Journal of Operational Research 88, 50–68.

Demeulemeester, E., Vanhoucke, M., Herroelen, W., 2003. A random network generator for activity-on-the-node networks. Journal of scheduling 6, 17–38.

Dorndorf, U., Pesch, E., Phan-Huy, T., 2000. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. Management Science 46, 1365–1384.

Essafi, I., Mati, Y., Dauzere-Péres, S., 2008. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers & Industrial Engineering 35, 2599–2616.

Focacci, F., Laborie, P., Nuijten, W., 2000. Solving scheduling problems with setup times and alternative resources, in: Artificial Intelligence Planning Systems 2000 Proceedings (AIPS), AIPS. pp. 1–10.

Franck, B., Schwindt, C., 1995. Different resource constrained project scheduling models with minimal and maximal time-lags. Technical Report. Universitat Karlsruhe.

Geem, Z.W., Kim, J.H., Loganathan, G., 2001. A new heuristic optimization algorithm: Harmony search. Simulation 76, 60–68.

Glover, F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking. Control and Cybernetics 29, 653–684.

Hanzálek, Z., Šůcha, P., 2009. Time symmetry of project scheduling with time windows and take-give resources, in: 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA), Dublin, Ireland, Springer. pp. 239–253.

Hartmann, S., Briskorn, D., 2010. A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207, 1–14.

Harvey, W.D., Ginsberg, M.L., 1995. Limited discrepancy search, in: Proceedings IJCAI 95, pp. 607–613.

Herroelen, W., De Reyck, B., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers & Operations Research 25, 279–302.

Herroelen, W., De Reyck, B., Demeulemeester, E., 1999. A classification scheme for project scheduling, in: Weglarz, J. (Ed.), Handbook of Recent Advances in Project Scheduling. Kluwer Academic Publishers, Dordrecht, pp. 1–26.

Icmeli, O., Erenguc, S.S., Zappe, C.J., 1993. Project scheduling problems: A survey. International Journal of Operations & Production Management 13, 80–91.

Kellenbrink, C., 2012. First Results on Resource-Constrained Project Scheduling with Model-Endogenous Decision on the Project Structure. Springer International Publishing. chapter Scheduling and Project Management. Operations Research Proceedings, pp. 429–434.

Kellenbrink, C., Helber, S., 2015. Scheduling resource-constrained projects with a flexible project structure. European Journal of Operational Research 246, 379–391.

Kis, T., 2003. Job-shop scheduling with processing alternatives. European Journal of Operational Research 151, 307–322.

Kobayashi, M., Hirano, Y., Higashi, M., 2014. Optimization of assembly processes of an automobile wire harness. Computer-Aided Design and Applications 11, 305–311.

Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90, 320–333.

Kolisch, R., Hartmann, S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174, 23–37.

Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega The International Journal of Management Science 29, 249–272.

Korte, B., Vygen, J., 2000. Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics 21). Springer-Verlag, Berlin Heidelberg New York Tokyo, First edition.

Kuster, J., Jannach, D., Friedrich, G., 2006. Handling alternative activities in resource-constrained project scheduling problems, in: Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, AAAI Press. pp. 1960–1965.

Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, 343–362.

Leung, C.W., Wong, T.N., Maka, K.L., Fung, R.Y.K., 2010. Integrated process planning and scheduling by an agent-based ant colony optimization. Computers & Industrial Engineering 59, 166–180.

Li, X., Zhang, C., Gao, L., Li, W., Shao, X., 2010. An agent-based approach for integrated process planning and scheduling. Expert Systems with Applications 37, 1256–1264.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1998. An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation. Management Science 44, 714–729.

Mirabi, M., 2010. A hybrid simulated annealing for the single-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times and costs and dynamic release of jobs. The International Journal of Advanced Manufacturing Technology 54, 795–808.

Moon, C., Kim, J., Hur, S., 2002. Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. Computers & Operations Research 43, 331–349.

Naderi, B., Zandieh, M., Shirazi, M., 2009. Modeling and scheduling a case of flexible flowshops: Total weighted tardiness minimization. Computers & Industrial Engineering 57, 1258–1267.

Neumann, K., Schwindt, C., Zimmermann, J., 2003. Project scheduling with time windows and scarce resources. Springer-Verlag Berlin Heidelberg.

Niu, B., Bi, Y., Chan, F., Wang, Z., 2015. Srbfo algorithm for production scheduling with mold and machine maintenance consideration, in: Intelligent Computing Theories and Methodologies. Springer International Publishing. volume 9226 of *Lecture Notes in Computer Science*, pp. 733–741.

Özdamar, L., Ulusoy, G., 1995. A survey on the resource-constrained project scheduling problem. IIE Transactions 27, 574–586.

Ángeles Pérez, Quintanilla, S., Lino, P., Valls, V., 2014. A multi-objective approach for a project scheduling problem with due dates and temporal constraints infeasibilities. International Journal of Production Research 52, 3950–3965.

Pinedo, M., Singer, M., 1999. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. Naval Research Logistics 46, 1–17.

Rand, G.K., 1977. Machine scheduling problems: Classification, complexity and computations. European Journal of Operational Research 1, 206.

Rau, B.R., 1994. Iterative modulo scheduling: an algorithm for software pipelining loops, in: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, ACM. pp. 63–74.

Salewski, F., Schirmer, A., Drexl, A., 1997. Project scheduling under resource and mode identity constraints: Model, complexity, methods and application. European Journal of Operational Research 102, 88–110.

Shabtay, D., Steiner, G., 2007. A survey of scheduling with controllable processing times. Discrete Applied Mathematics 155, 1643–1666.

Shao, X., Li, X., Gao, L., Zhang, C., 2009. Integration of process planning and scheduling - a modified genetic algorithm-based approach. Computers & Operations Research 36, 2082–2096.

Snedecor, G.W., Cochran, W.G., 1989. Statistical Methods. Iowa State University Press; Eighth Edition.

Spaccamela, A.M., Nanni, U., Rohnert, H., 1996. Maintaining a topological order under edge insertions. Information Processing Letters 59, 53–58.

Tasgetiren, M.F., Pan, Q.K., Liang, Y.C., 2009. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. Computers and Operations Research 36, 1900–1915.

Usher, J.M., 2003. Evaluating the impact of alternative plans on manufacturing performance. Computers and Industrial Engineering 45, 585–596.

Van Peteghem, V., Vanhoucke, M., 2011. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. Journal of Heuristics 17, 705–728.

Van Peteghem, V., Vanhoucke, M., 2014. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. European Journal of Operational Research 235, 6272.

Vanhoucke, M., Debels, D., 2007. The discrete time/cost trade-off problem: Extensions and heuristic procedures. Journal of Scheduling 10, 311–326.

Vanhoucke, M., Demeulemeester, E., Herroelen, W., 2001. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. Annals of operations research 102, 179–196.

Wang, D., Goodchild, A., Li, X., Wang, Z., 2014. Double girder bridge crane with double cycling: Scheduling strategy and performance evaluation. Journal of Applied Mathematics 2014, 1–12.

Wang, L., Wang, M., 1997. A hybrid algorithm for earliness-tardiness scheduling problem with sequence dependent setup time, in: Proceedings of the 36th Conference on Decision & Control, IEEE. pp. 1219–1223.

Yuan, X.M., Khoo, H.H., Spedding, T.A., Bainbridge, I., Taplin, D.M.R., 2004. Minimizing total setup cost for a metal casting company. Winter Simulation Conference 2, 1189–1194.

Zhang, R., Wu, C., 2011. A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardinessobjective. Computers & Industrial Engineering 38, 854–867.

Zhou, H., Cheung, W., Leung, L.C., 2009. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. European Journal of Operational Research 194, 637–649.

Curriculum Vitae

Roman Čapek was born in Pardubice, Czech Republic, in 1983. He received his master of science degree in Cybernetics and Control Engineering at Faculty of Electrical Engineering, Czech Technical University in Prague (CTU) in 2008. The master thesis with title *Scheduling and Visualization of Manufacturing Processes* was focused on the mathematical modeling of scheduling problems in manufacturing and visualization of the schedules in virtual reality. In the same year, 2008, he started his Ph.D. studies at the Department of Control Engineering at CTU Prague. Besides the research in the main topic called *Scheduling with Alternative Process Plans* he participated on the projects Elektro kontakt (project Eureka), Adaptive scheduling and Optimization algorithms (Czech Science Foundation) and Centre for Applied Cybernetics I-III (responsible for work packages 10 and 21). His main area of interest is combinatorial optimization applied for the scheduling of production processes.

His teaching activities in CTU involved courses of Logic Control and Optimization in Intelligent Systems, where he participated also on the content of the lectures. He has also supervised a lot of student projects and bachelor and diploma theses.

Roman Čapek presented his research results at several international conferences, e.g. *MISTA* (top conference for the combinatorial optimization), *PMS* (Project Management and Scheduling) and *EURO* (European Conference on Operational Research). The most important results were published in the international impacted journal *European Journal of Operational Research* (EJOR - IF 2.911), another one is under review in and *Computers & Operations Research*. Moreover, the results published in EJOR journal were included also in *Handbook of Project Management and Scheduling*, published under Springer.

Roman Čapek was the lead author of two software applications used by Styl Plzeň and Procter&Gamble, where the results of the theoretical research were exploited for the optimization of the manufacturing processes.

Czech Technical University in Prague
Prague, August 2015 *Roman Čapek*

List of Author's Publications

All of the author's publications are directly related to the topic of the thesis. They are separated into five groups as follows.

Publications in Journals with Impact Factor

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Production scheduling with alternative process plans. *European Journal of Operational Research*, 217(2):pages 300–311, 2012. ISSN 0377-2217. doi: doi:10.1016/j.ejor.2011.09.018. URL `http://www.sciencedirect.com/science/article/pii/S0377221711008460`. Accessed: 2015-08-18. **Co-authorship 50 %, cited by** (Kellenbrink and Helber (2015); Ángeles Pérez et al. (2014); Wang et al. (2014)) – **indexed in Web of Science**, **cited by** (Kobayashi et al. (2014); Abdolshah (2014); Niu et al. (2015)).

Roman Čapek, Zdeněk Hanzálek, and Přemysl Šůcha. Scheduling with alternative process plans under minimisation of the total setup time. *Computers & Operations Research*, 2015. In major revision. **Co-authorship 50 %**.

Publications in Reviewed Journals

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Production scheduling with alternative process plans. *European Journal of Operational Research*, 217(2):pages 300–311, 2012. ISSN 0377-2217. doi: doi:10.1016/j.ejor.2011.09.018. URL `http://www.sciencedirect.com/science/article/pii/S0377221711008460`. Accessed: 2015-08-18. **Co-authorship 50 %, cited by** (Kellenbrink and Helber (2015); Ángeles Pérez et al. (2014); Wang et al. (2014)) – **indexed in Web of Science**, **cited by** (Kobayashi et al. (2014); Abdolshah (2014); Niu et al. (2015)).

Roman Čapek, Zdeněk Hanzálek, and Přemysl Šůcha. Scheduling with alternative process plans under minimisation of the total setup time. *Computers & Operations Research*, 2015. In major revision. **Co-authorship 50 %**.

Patents

There are no patents related to the thesis.

Publications indexed in Web of Science

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Production scheduling with alternative process plans. *European Journal of Operational Research*, 217(2):pages 300–311, 2012. ISSN 0377-2217. doi: doi:10.1016/j.ejor.2011.09.018. URL http://www.sciencedirect.com/science/article/pii/S0377221711008460. Accessed: 2015-08-18. **Co-authorship 50 %, cited by** (Kellenbrink and Helber (2015); Ángeles Pérez et al. (2014); Wang et al. (2014)) – **indexed in Web of Science**, **cited by** (Kobayashi et al. (2014); Abdolshah (2014); Niu et al. (2015)).

Other Publications

International Conference Papers

Roman Čapek. Visualization and simulation in scheduling. In *Poster 2008*, pages 12–14. Czech Technical University in Prague, 2008. URL https://ojs.cvut.cz/ojs/index.php/ap/article/view/977. Accessed: 2015-08-18. **Co-authorship 100 %**.

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Production scheduling with alternative process plans. In *EURO XXIII Bonn - Book of Abstracts*, page 278. The Association of European Operational Research Societies, 2009. Accessed: 2015-08-18. **Co-authorship 33 %**.

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Alternative process plans in wire harnesses production. In *15th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–8. IEEE, 2010. doi: 10.1109/ETFA.2010.5641230. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=5641230. Accessed: 2015-08-18. **Co-authorship 40 %**.

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Minimizing total weighted tardiness and total processing costs for the scheduling with alternative process plans. In *Proceedings of the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pages 561–563. Nottingham: University of Nottingham, 2011. URL http://www.schedulingconference.org/previous/publications/

displaypub.php?key=2011-561-563-A&filename=mista.bib. Accessed: 2015-08-18. **Co-authorship 50 %**.

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. Scheduling with alternative process plans and the total weighted tardiness criterion. In *Proceedings of the 10th Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 180–182. Institute for Theoretical Computer Science in Prague, 2011. Accessed: 2015-08-18. **Co-authorship 50 %**.

Roman Čapek. Resource constrained project scheduling problem with alternative process plans and total changeover cost minimization. In *Proceedings of the 25th European Conference on Operational Research (EURO 2012)*, pages 183–183. The Association of European Operational Research Societies, 2012. Accessed: 2015-08-18. **Co-authorship 100 %**.

Roman Čapek, Vincent Van Peteghem, and Zdeněk Hanzálek. Minimization of the total production cost for the RCPSP with alternative process plans. In *Proceedings of the 13th International Conference on Project Management and Scheduling*, pages 110–113. Operations Management Research Group of the Faculty of Business and Economics, 2012. ISBN 978-908-140-994-0. Accessed: 2015-08-18. **Co-authorship 33 %**.

Roman Čapek, Zdeněk Hanzálek, and Lucie Bužková. Constraint programming and evaluation methods for scheduling with alternative process plans. In *Proceedings of the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 35–37. LORIA Campus Scientifique, 2013. Accessed: 2015-08-18. **Co-authorship 33 %**.

Oleh Sobeyko, Roman Čapek, Zdeněk Hanzálek, and Lars Mönch. An instance generator for scheduling problems with alternative process plans. In *Proceedings of the EURO—INFORMS 26th European Conference on Operational Research*, page 9. Sapienza University of Rome, 2013. Accessed: 2015-08-18. **Co-authorship 25 %**.

Remaining Publications

Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek. *Scheduling of Production with Alternative Process Plans*, pages 1187–1204. Handbook on Project Management and Scheduling Vol.2. Springer International Publishing, 2015. ISBN 978-3-319-05914-3. doi: 10.1007/978-3-319-05915-0. URL http://link.springer.com/book/10.1007%2F978-3-319-05915-0. Accessed: 2015-08-18. **Co-authorship 50 %**.

Michal Kutil, Přemysl Šůcha, Roman Čapek, and Zdeněk Hanzálek. *Optimization and Scheduling Toolbox*, pages 239–260. Matlab - Modelling, Programming and Simulations. Sciyo, 2010. ISBN 978-953-307-125-1. URL http://

cdn.intechopen.com/pdfs/11619.pdf. Accessed: 2015-08-18. **Co-authorship 25 %**.

Roman Čapek. Preliminary Doctoral Thesis: Scheduling with Alternatives. Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic, 2010. Accessed: 2015-08-18. **Co-authorship 100 %**.

Roman Čapek, Michal Kutil, Přemysl Šůcha, and Zdeněk Hanzálek. Locusta - aplication for support of production planning and scheduling, 2010. Authorized software. Accessed: 2015-08-18. **Co-authorship 25 %**.

Michal Kutil, Přemysl Šůcha, Roman Čapek, and Zdeněk Hanzálek. Torsche scheduling toolbox for matlab (version 0.4.0), 2010. Authorized software. Accessed: 2015-08-18. **Co-authorship 25 %**.

Roman Čapek and Zdeněk Hanzálek. Project scheduling with alternative process plans under different constraints and criteria. Prof. Dr. Lars Mönch, FernUniversität in Hagen, 2010. Unpublished Lecture. Accessed: 2015-08-18. **Co-authorship 75 %**.

Czech Technical University in Prague
Prague, August 2015 *Roman Čapek*

This thesis is focused on the scheduling problems with alternative process plans. Its goals were set as follows:

1. To propose a common representation for the scheduling problems that include alternative processes.

2. To establish a mathematical formulation using the proposed representation for each studied problem.

3. To develop an algorithm to solve large instances for each of the problems.

4. To compare the proposed solution methods with the similar works from the literature.

5. To propose the methodology for evaluation and comparison of different solution approaches.