FACULTEIT ECONOMIE EN BEDRIJFSKUNDE

MULTI-PROJECT SCHEDULING

THE APPLICATION OF A DECOUPLED SCHEDULE GENERATION SCHEME AND A GAME MECHANIC

Aantal woorden/ Word count: 18.656

Rob Van Eynde
Stamnummer/ Student number : 01205632

Promotor/ Supervisor: Prof. dr. Mario Vanhoucke

Masterproef voorgedragen tot het bekomen van de graad van:
Master's Dissertation submitted to obtain the degree of:

Master of Science in Business Engineering

Academiejaar/ Academic year: 2016 - 2017

UNIVERSITEIT GENT

FACULTEIT ECONOMIE EN BEDRIJFSKUNDE

MULTI-PROJECT SCHEDULING

THE APPLICATION OF A DECOUPLED SCHEDULE GENERATION SCHEME AND A GAME MECHANIC

Aantal woorden/ Word count: 18.656

Rob Van Eynde
Stamnummer/ Student number : 01205632

Promotor/ Supervisor: Prof. dr. Mario Vanhoucke

Masterproef voorgedragen tot het bekomen van de graad van:
Master's Dissertation submitted to obtain the degree of:

Master of Science in Business Engineering

Academiejaar/ Academic year: 2016 - 2017

UNIVERSITEIT GENT

Foreword

Different people have contributed to this dissertation, they deserve a word of thanks. Pieter Leyman provided useful feedback and insights during the first year of writing and experimenting. I am particularly grateful towards Jeroen Burgelman, who was a valuable guide during the second year. He introduced interesting literature, he helped focussing the research and was a frequent sounding board. Last, Bram Bossuyt tested out the basics of the game mechanic with me, which provided initial insights to start with.
Thank you.

Contents

List of Figures

List of Tables

List of Algorithms

Definitions and abbreviations

General definitions	
J	number of projects in a multi-project instance
j	project index: $j \in [1, J]$
p_j	project j
N_j	number of activities in p_j
i	activity index: $i \in [1, N_j]$
a_{ij}	activity i of project j
K	number of renewable resources
k	renewable resource index: $k \in [1, K]$
R_k	maximum available units of resource type k per time unit
R'_{kt}	remaining available units of resource type k at time t
t	time unit index
d_{ij}	duration of a_{ij}
r_{ijk}	demand for resource type k by a_{ij} per time unit
w_{ijk}	total demand for resource type k by a_{ij}
K_{ij}	number of resource types for which the demand by a_{ij} is nonzero
CP_j	resource unconstrained critical path duration of p_j
CP_{max}	$\max_{j \in J}(CP_j)$
ES_{ij}	earliest start time for a_{ij} according to the critical path of p_j
LS_{ij}	latest start time for a_{ij} according to the critical path of p_j
E_{ij}	earliest precedence and resource feasible start time of a_{ij} according to the current partial schedule

Definitions chapter 3	
APD	average percent delay, the objective function
S	number of intervals over which the project duration is split
s	interval index: $s \in [1,S]$
d_s	duration of interval s
L	number of schedule agents
l	schedule agent index: $l \in [1, L]$
SA_l, SA_w	schedule agent l, the winning schedule agent
CA	coordinating agent
ps	partial schedule
P_l	the set of projects assigned to SA_l
$\mathcal{C}_l$	completed set of SA_l
$\mathcal{D}_l$	decision set of SA_l
$\mathcal{I}_l$	ineligible set of SA_l
$\mathcal{E}_l$	eligible set of SA_l
$\mathcal{S}_l$	schedulable set of SA_l
s_{lk}	the current amount of resource type k on the stack of SA_l
q_l, q'_{ls}, q''_l	probabilities of SA_l to receive resources during a game round
LB_q	lower bound on q_l, q'_{ls}, q''_l
f_k	the maximum amount of resource k allocated to an agent during a game round
W_l	total resource demand of SA_l
W_{ls}	total resource demand of SA_l during interval s
TWK_j	total resource demand of the scheduled activities of p_j
$\mathcal{T}_{ij}$	resource tightness of a_{ij} in the current partial schedule
$\mathcal{C}_{ij}$	amount of competing activities of a_{ij} in the current partial schedule
$\mathcal{P}_{ij}$	priority value of a_{ij}
M_l	performance of SA_l
o_l, o_w	schedule order sent by SA_l, winning order selected by CA
$v(o_l)$	value of o_l
co_l	compensating order sent by by SA_l
δ_l	minimal deterioration of makespan for SA_l if o_w would be scheduled
η	parameter for selective pressure of random sampling
τ, κ	selection parameters tightness evaluation
ω	probability parameter PBQU
ν	trigger probability negotiation mechanism

Definitions chapter 4	
C_j	complexity measure of p_j, defined by Browning and Yassine (2010a)
OS_j	order strength of p_j
NARLF	indicator for the resource loading of a multi-project
MAUF, σ^2_{MAUF}	indicators for the resource utilization of a multi-project
n	amount of generated multi-project instances per combination of project parameter values
H, M, L	indices for order strength, referring to High, Medium and Low order strength respectively
W_k	total demand for resource type k by the multi-project instance

Definitions chapter 5	
$\mathcal{C}, \mathcal{D}, \mathcal{I}$	completed, decision and ineligible set of the SSGS, respectively
(x, y, z)	elitism, crossover and mutation percentages for the genetic algorithm

Abbreviations	
APD	Average percent delay
APR	Activity priority rule
ARLF	Average resource loading factor
AUF	Average utilization factor
GA	Genetic algorithm
LBA	Lower bound activity
LBQU	Load based quarry updating
MAS	Multi-agent system
MAUF	Modified average utilization factor
MAXTWK	Maximum total work content
MPSGS	Multi-project schedule generation scheme
NARLF	Normalized average resource loading factor
OS	Order Strength
PBQU	Performance based quarry updating
PPR	Project priority rule
PR	Priority rule
PSGS	Parallel schedule generation scheme
RCMPSP	Resource-constrained multi-project scheduling problem
RCPSP	Resource-constrained project scheduling problem
RS	Random sampling
SASP	Shortest activity of the shortest project
SGS	Schedule generation scheme
SSGS	Serial schedule generation scheme

Part I

Introduction

Chapter 1

Introduction

An ever increasing amount of companies organize their work in project structures. A project can be described as a set of activities that have to be executed, each having a duration and a renewable and/or non-renewable resource demand (e.g. amount of workers required or monetary cost, respectively). Furthermore, precedence relations between some of these activities exist, i.e. an activity can only be started when all its predecessors are completed. As companies have a limited amount of resources, some activities will have to be delayed. In general, the scheduling problem is subject to two kinds of constraints: the resource constraints due to limited capacity and the precedence constraints as formulated above. The problem of finding a solution that minimizes or maximizes an objective function (e.g. the makespan of the project) taking into account the formulated constraints is called the *resource-constrained project scheduling problem* (RCPSP). However, a lot of companies have a *portfolio* of projects that are simultaneously active (e.g. construction companies having multiple building sites). This has led to a new branch of research, regarding the *resource-constrained multi-project scheduling problem* (RCMPSP). This problem is similar to the RCPSP, but considers multiple projects that have to be scheduled in concurrence. Precedence relations exist between activities of a project, but there are none between projects. One or more resource types are shared among projects (e.g. workers can be assigned to different construction sites). Multiple extensions to the basic RCMPSP exist, two of them will be discussed more in detail in chapter 2. However, this research will address the basic RCMPSP, i.e. all projects are available at the first time instance, no preemption or task splitting is allowed, there are no setup times for activities, there are only renewable resources, every activity has a single mode of execution

and a constant resource demand over time.

This dissertation will provide three main contributions to the existing research. First, a schedule generation scheme is designed specifically for the multi-project context. Second, a game mechanic is implemented in the scheduling process, resulting in an efficient solution method with a strong performance relative to benchmark methods. Third, a multi-project generator and data set are proposed and the existing multi-project parameters are critically evaluated.

Chapter 2

Literature review

The RCMPSP can be approached as a single-project or as a multi-project. The former adds a dummy start and end activity, resulting into one overarching project network. The latter addresses each project as a seperate instance with its own characteristics. The two approaches may result in different schedules (Kurtulus and Davis, 1982). Objective functions can be classified accordingly: those who evaluate the performance of the portfolio as a whole and those who take into account the performance of the constituting projects. This master dissertation will focus on makespan minimization. In this field, Lova and Tormos (2001) formulate two objective functions: mean project delay (MPD) and multi-duration increase (MDI). Browning and Yassine (2010b) propose the measures average percent delay (APD) and percentage delay (PD). MPD and APD are multi-project measures, while MDI and PD are more relevant in the single-project approach. As the RCMPSP will be addressed as a multi-project here, APD will be used as objective function. The remainder of this chapter is structured as follows: first, different solution methods for the RCMPSP are discussed. These include the schedule generation schemes with complementing priority rules, meta-heuristics (focusing on genetic algorithms) and multi-agent systems (MAS). Second, the dynamic extension of the RCMPSP is reviewed. The last paragraph provides an overview of multi-project generation attempts and data sets.

Priority rule-based heuristics consist of two parts: a schedule generations scheme (SGS) and a priority rule (PR). In general, the SGS selects an eligible activity (i.e. whose predecessors have been scheduled) and schedules it at the first time instance where no resource or precedence constraints are violated. When multiple activities are eligible, a certain PR is used to select the

activity to be scheduled first (Kolisch, 1996b). The author analyzes the differences between the Serial Schedule Generation Scheme (SSGS) and the Parallel Schedule Generation Scheme (PSGS). He states that "The parallel scheduling scheme searches in a smaller solution space than the serial scheduling scheme, but with the severe drawback that, when considering a regular performance measure, the solution space might not contain the optimal solution." (Kolisch, 1996b). Following this reasoning, the SSGS will be used as base for the genetic algorithm. Kolisch and Meyer (2006) implement a hybrid SGS. By setting the parameter θ the size of the search space can be determined. If $\theta = 0$, the scheme will function as a PSGS, if $\theta = 1$ it will behave as a SSGS. The parameter can take all intermediate values, a higher θ will result in a larger part of the search space being evaluated.

Priority rules can be classified along different dimensions, e.g. the information they use, the number of passes that are executed, the amount of information processed and its dynamic or static nature (Browning and Yassine, 2010b; Kolisch, 1996a). Browning and Yassine (2010b) compare the performance 20 PR's of that have been used in research on different objective functions. They note that PR's considering the information of the separate projects in one multi-project instance perform better on minimizing the average delay of all projects. According to their study, the priority rules MINWCS and TWK-LST perform well in general. Several articles identify MAXTWK and SASP as the PR's giving the best results for minimization of average project delay (Kurtulus and Davis, 1982; Kurtulus, 1985; Lova et al., 2000; Lova and Tormos, 2001). Lawrence and Morton (1993) propose an R & M scheduling policy that takes into account information about resource demand, slack and network structure to minimize weighted tardiness. This PR outperforms other rules that have performed well in previous research and proved to be robust regarding different project parameters (among others: AUF, OS, RF, etc.). The existing PR-based heuristics have a flaw: they are not really adapted to a multi-project context. Although some PR's incorporate project specific information (e.g. SASP, cfr. infra), they combine activity and project information in one measure. This may cloud the activity selection process. This dissertation will formulate an answer to the postulated problem by introducing a SGS that decouples project and activity information.

A second approach to solving the RCMPSP involves meta-heuristics. Shtub et al. (1996) consider an environment where projects are executed in a repetitive manner (e.g. the construction

of naval patrol boats). The authors apply three different methods: simulated annealing, a genetic algorithm and a pair-wise swap algorithm. However, the focus of the article is on learning effects and the work schedules of different crews. Kumanan et al. (2006) propose a genetic algorithm for the RCMPSP. The problem is represented by a permutation encoding, a single point crossover and a swap mutation are used. However, the heuristic that is used to create a schedule from the representation string is not a SGS in the strict sense. It is similar to the PSGS, but among others no priority rule is explicitly chosen. Gonçalves et al. (2008) develop a genetic algorithm for the dynamic RCMPSP. The representation of the problem incorporates priority values and delay times for every activity; and release dates for every project. The reproduction step is a combination of an elitist strategy and a crossover operator. The mutation operator replaces the worst performing individuals by newly generated ones. This representation and decoding are not applicable to the static RCMPSP, but the general concepts of the evolutionary strategy will be implemented in the genetic algorithm of this paper.

The third solution method that will be discussed are the multi-agent systems. MAS's introduce a different approach to the RCMPSP. They are often used to address the Decentralized RCMPSP, where it is assumed that not all information is centrally available. This results in complex interactions between individual project managers, which can be modelled using agents. MAS's are "more robust, flexible and fault tolerant than traditional systems ... are easier to program ... [and] can often solve problems faster (by exploiting parallelism)" (Knotts et al., 2000). The authors categorize agents according to their sophistication on a spectrum from purely reactive to purely deliberative. They model a system where every activity is represented by an agent and apply it to the MultiMode, Resource-Constrained Project Scheduling Problem (MMRCPSP). There is a central 'Blackboard' that stores relevant information (e.g. resource and precedence constraints, available resources) that is accessible by every agent. Basic agents select a certain mode and request the required resources, which they are granted according to their respective priority values. Advanced agents evaluate multiple modes following a more complex logic. Confessore et al. (2007) allocate one agent (project manager) to every project and institute one coordinator agent, who distributes resource time slots among the managers. The coordination process is modeled as a combinatorial auction. All managers formulate a bid for certain slots with an according price. These bids are processed by the

coordinating agent, who allocates the slots to highest bidders. The agents that did not receive any slots reformulate their bidding, adapted to the new situation. This process is iterated until all agents have the necessary slots to schedule their activities. Homberger (2007) combines a Restart Evolutionary Strategy (RES) with a MAS. Every project has one Schedule Agent, who negotiates with a coordinating Mediator. The Mediator allocates resources to the Schedule Agents, the latter use these to create a schedule for their project (using the RES). Excess resources are sent back to the Mediator who communicates the global excess. All Schedule Agents reschedule their network assuming they have all excess resources and evaluate the improvement of the schedule. Based on this information, the Mediator allocates (parts of) the excess to the Schedule Agents, who reschedule their network using the extra capacity. This process is iterated until the average makespan cannot be reduced further. Lee et al. (2003) apply a MAS guided by a market-based control mechanism to the Distributed Multiple Projects (DMP) environment. The focus is on project control. Local markets are established to solve conflicts that arise from disruptions in the initial schedule. Agents bid for resource time slots in which they can schedule their activity. The authors propose a new mechanism to solve the market-based discrete resource allocation problem. Instead of adapting prices for resource time slots based on their demand, the prices are adjusted according to the precedence conflicts that occur from that demand. Homberger and Fink (2017) devise two generic negotiation mechanisms for the decentralized RCMPSP and apply it to a scheduling problem with two agents. Each agent is responsible for a set of activities and wants to maximize the discounted cash flow of his own activities, resulting from the schedule of the multi-project. By using money transfers, an agent can convince others to accept certain changes in the schedule. The effectiveness of these mechanisms is evaluated in a context where agents do not always behave truthfully. When information is decentralized, this can be a realistic situation. With mechanism NM-SP-2, the dominant strategy for both agents is to behave truthfully. This property is valuable in situations where players are able to lie in order to increase their own gain. Most of the research cited above develops systems that work in a deterministic way, i.e. a given starting situation will always result in the same outcome. The mechanism introduced in this dissertation tries to model the decision process of agents alike that of players in a (board)game. The agents will evaluate different possible options and make a decision, which will be based on a combination of randomness and available information.

The schedules resulting from multiple executions may thus be different.
The static RCMPSP can be extended to the dynamic RCMPSP. According to Dumond and Mabert (1988) the dynamic context is valid if new projects join the portfolio over time and there is uncertainty about future resource demands. The authors identify two types of decisions, i.e. planning and control decisions. The former involve the communication of a due date for the project to external parties. The latter incorporate the scheduling, where resources are assigned to activities in order to meet the proposed due date. Multiple alternatives for the two stages are evaluated. They identify the combination of the FIFS scheduling heuristic and the SFT due date setting rule as a strong performer on due date related performance measures. Dumond (1992) proposes a finite scheduling algorithm that is able to schedule projects in a dynamic environment. He evaluates the impact of resource availability levels on different time related performance measures. The research concludes that resource availability at levels above 160% provides no additional reduction in makespan or lateness. When resources become more constrained, the choice of a scheduling heuristic will have a significant impact on the performance. Bock and Patterson (1990) elaborate on resource preemption policies for newly arriving (mainly maintenance) projects and their impact on the long term development projects, which may incur rework loops. When a new high priority project arrives, resources of lower priority work in process are preempted and assigned to the former. They find that FIFS/SFT dominates other combinations in general. However, when limited resource preemption is allowed, the performance of MINSLK[DD] and MINLFT[DD] is not significantly different from FIFS. Tsubakitani and Deckro (1990) develop a scheduling and control model for the housing industry. The authors evaluate the Average Utilization Factor (AUF) and Average Resource Loading Factor (ARLF) of the multi-project to select the appropriate PR (SASP), which is used in combination with a scheduling algorithm similar to the PSGS. Two control features are introduced: the first one calculates the free slack of each activity, the second one receives input of actual information and provides an updated perspective on the current situation. Both routines assist the manager in controlling the projects. Not all research that incorporates extensions like due dates or release dates explicitly addresses the dynamic RCMPSP. Gonçalves et al. (2008) introduce a genetic algorithm that incorporates project release dates and activity delay times. The algorithm is not completely applicable to the dynamic context as the earliest release dates of every project are known in advance.

Lawrence and Morton (1993) compare activity costing and resource pricing heuristics with an iterative schedule updating procedure to minimize weighted project tardiness. This dissertation will address the static RCMPSP, but derived insights and concepts could be extended and applied to the dynamic RCMPSP.

Different authors have generated multi-project instances to test their proposed solution strategies. Some (Gonçalves et al., 2008; Homberger, 2007) combine single-project instances from benchmark sets like J120 from Kolisch et al. (1998). However, no analysis is made about the underlying problem parameters. Kurtulus and Davis (1982) propose two summary measures to categorize multi-projects: the ARLF and the AUF. The authors generate 77 problem instances along these two dimensions, but no further detail is given about the generation procedure. Browning and Yassine (2010a) present the first random network generator for RCMPSP instances. The input parameters for the generator are Normalized Average Resource Load Factor (NARLF), Modified Average Utilization Factor (MAUF), Variance of MAUF, Variance of NARLF, and a complexity measure C_j. However, some aspects of the generator show room for improvement, as will be shown in chapter 4. The network generator in this dissertation will build on RanGen1, a generator for RCPSP instances (Demeulemeester et al., 2003) and combine it with procedures from (Browning and Yassine, 2010a). RanGen1 was augmented to RanGen2 by Vanhoucke et al. (2008). This generator can create instances from a broader range of topological network indicators than its predecessor. (N)ARLF and (M)AUF are the most prevalent multi-project parameters in research. However, this research will show that they are not flawless.

Part II

Methodology

Chapter 3

New approaches to multi-project scheduling

This chapter will introduce two new scheduling methods for the RCMPSP. Before discussing these approaches, the objective function on which they will be evaluated is defined. As stated in chapter 2, Browning and Yassine (2010b) base their analyses on two measures: Average percent delay (APD) and Percentage delay (PD), described by the following equations:

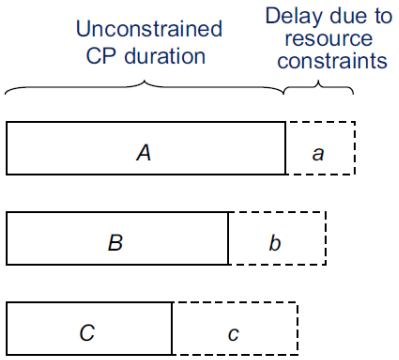
$$\text{Average percent delay} = \frac{(a/A) + (b/B) + (c/C)}{3} \tag{3.1}$$

$$\text{Percentage delay} = \frac{\text{Max}(A + a, B + b, C + c) - \text{Max}(A, B, C)}{\text{Max}(A, B, C)} \tag{3.2}$$

A, B, C are the resource unconstrained CP durations of the three projects while a, b, c are the delays due to resource constraints, as shown in figure 3.1. APD evaluates the delay of every project in the portfolio and aggregates them into one measure. PD makes abstraction of the underlying projects and only looks at the delay of the portfolio. If one project has large delays this will inflate PD, even though all other projects may have small delays. As APD treats the multi-project more as a combination of projects, it will be used as objective function in this dissertation.

The contributions derived in this chapter are twofold. First, a Multi-Project Schedule Generation Scheme (MPSGS) is proposed. It is an extension of the SSGS, with decoupled project and activity selection decisions. Second, the application of a game mechanic on the MPSGS is discussed. Additionally, the behaviour of players in this game is augmented. The remain-

Figure 3.1: An example multi-project instance



Source: Browning and Yassine (2010b)

der of this chapter is structured as follows: Section 3.1 discusses the MPSGS, section 3.2 elaborates on the structure of the basic game mechanic. Subsections 3.3.1 and 3.3.2 discuss augmentations of the decision process of schedule agents in order to model player-like behaviour. Subsections 3.3.3 and 3.3.4 propose adaptations of the resource allocation by the coordinating agent.

3.1 MPSGS, a decoupled SGS

This section will propose a multi-project schedule generation scheme (MPSGS) that allows the decoupling of activity and project selection decisions. To the best of the author's knowledge, no other SGS explicitly does this. Existing SGS's evaluate the activities in the eligible set (i.e. the activities for which all predecessors are scheduled) of all projects and select the one with the best priority value $\mathcal{P}_{ij}$. Some of the PR's incorporate project information, but a multi-project instance is still treated as one aggregated network and not as seperate networks. Apart from new priority rules, there is no difference between the SGS for the RCPSP and for the RCMPSP. Take the priority rule Shortest Activity of the Shortest Project (SASP) as an example, which selects the activity with minimal

$$\mathcal{P}_{ij} = d_{ij} + CP_j. \tag{3.3}$$

It incorporates activity (d_{ij}) and project (CP_j) specific information, but the two types are not evaluated separately. As a consequence, it does not select the shortest activity of the shortest

project. Take two projects p_1, p_2 with $CP_1 = 20, CP_2 = 22$. Let $a_{i1}(d_{i1} = 5)$ and $a_{i2}(d_{i2} = 2)$ be the activities with the shortest duration in the eligible set of p_1 and p_2 respectively. The resulting priority values are $\mathcal{P}_{i1} = 25$ and $\mathcal{P}_{i2} = 24$. If one would select the shortest activity of the shortest project, one would choose a_{i1}, as $CP_1 < CP_2$. However, if one would use the SASP priority rule, one would choose a_{i2} as $\mathcal{P}_{i1} > \mathcal{P}_{i2}$. The latter selection mechanism is present in the traditional SGS's, while the former (decoupled selection) is incorporated in the MPSGS. First, for every project, the activity with the best priority value is selected. Second, the project with the best priority value is chosen, the according activity is scheduled.

A decoupled selection may not only change the meaning of existing priority rules, it also enables the creation of new ones. For instance, Longest Activity of the Shortest Project (LASP) cannot be implemented in a regular SGS. In equation 3.3, the first term should be maximized while the second term should be minimized. As a consequence, it is unclear whether $\mathcal{P}_{ij}$ should be minimized or maximized. As long as the objectives for both activity and project information point in the same direction (e.g. SASP) there is no problem. However, this limits the possibilities of creating priority rules. The MPSGS allows setting different selection rules for activities and projects. Above, the structure of the MPSGS is given in

Algorithm 1: MPSGS

```
ps = ∅
while not all activities scheduled do
    for l in L do
        SA_l selects a_ij with highest priority in D_l
        SA_l sends schedule order o_l for a_ij
    if any orders sent then
        CA selects order with highest v(o_l), i.e. the winning order o_w
        CA schedules activity a_ij of o_w at time E_ij in ps
        All agents update information based on the new partial schedule
return ps
```

pseudocode. Note that it is very similar to the structure of the schedule game in section 3.2. Let ps be the partial schedule, i.e. a schedule in which not all activities are added yet. Before the first iteration ps is empty, after the last iteration it contains all activities of all projects. Every project is assigned a Schedule Agent (SA_l), he is responsible for selecting an activity

of his project. Each SA_l has three disjoint sets containing activities. The Completed set ($\mathcal{C}_l$) contains all activities that are already added to the partial schedule. The Decision set ($\mathcal{D}_l$) contains the activities for which all predecessors are in $\mathcal{C}_l$, which means they can be scheduled. For the activities in the Ineligible set ($\mathcal{I}_l$), not all predecessors are scheduled yet, they cannot be added to the schedule at the current time. $\mathcal{C}_l \cup \mathcal{D}_l \cup \mathcal{I}_l$ contains all activities of the projects assigned to SA_l. Note that $\mathcal{C}_l$ corresponds to the *scheduled set* and $\mathcal{D}_l$ to the *decision set* from Kolisch et al. (1995). As long as not all activities are scheduled, iterations of the mechanism are executed. In every iteration, every agent selects a_{ij} with the highest *activity* priority value from his project and sends a schedule order o_l for that activity. This o_l contains information about the activity, the requesting agent and its earliest precedence and resource feasible start time (E_{ij}) according to ps. The order also has a value $v(o_l)$, which refers to the priority value of the *project*[1]. A Coordinating Agent (CA) collects the o_l of all schedule agents and selects the one with the best $v(o_l)$. This is the winning order o_w, the according SA_l is called the winning agent. The activity of this order is added to the partial schedule at E_{ij}. Then, two kinds of information are updated. First, $\mathcal{C}_l$, $\mathcal{D}_l$ and $\mathcal{I}_l$ of the winning agent are updated. Second, the E_{ij} are updated for all $a_{ij} \in \mathcal{D}_l$, $\forall l$. As the availability of resources have changed during certain time slots, the E_{ij} of some activities may be delayed. In every iteration of the MPSGS, one activity is added to the partial schedule. The orders of the losing agents are discarded, in the next iteration every SA_l sends a new order. Multiple activity and project priority rules (APR and PPR respectively) were implemented and tested, the different rules are listed in table 3.1.

Note that the use of the words project and agent (and their indices j and l) are equivalent here as every agent is responsible for exactly one project (P_l contains one project, $\forall l$). The total work content of a project (TWK_j) is defined as follows:

$$TWK_j = \sum_{i \in \mathcal{C}_l} \sum_{k=1}^{K} w_{ijk}, \tag{3.4}$$

i.e. the resource demand of all activities that are in the completed set. The performance of an agent is given by the equation below:

$$M_l = \frac{|\mathcal{C}_l|}{\sum_{j \in P_l} N_j}. \tag{3.5}$$

[1]If $v(o_l)$ would be equal to the priority value of the activity, the MPSGS would reduce to the SSGS.

Table 3.1: Activity and project priority rules

Code	Name	Definition
Activity Priority Rules		
LA	Longest Activity	Max(d_{ij})
SA	Shortest Activity	Min(d_{ij})
MAXWK	Maximum Work Content	Max($\sum_{k=1}^{K} w_{ijk}$)
MINWK	Minimum Work Content	Min($\sum_{k=1}^{K} w_{ijk}$)
MAXSLK	Maximum Slack	Max(Max($LS_{ij} - E_{ij}, 0$))
MINSLK	Minimum Slack	Min(Max($LS_{ij} - E_{ij}, 0$))
Project Priority Rules		
LP	Longest Project	Max(CP_j)
SP	Shortest Project	Min(CP_j)
MAXTWK	Maximum Total Work Content	Max(TWK_j)
MINTWK	Minimum Total Work Content	Min(TWK_j)
MAXCPL	Maximum Complexity	Max(OS_j)
MINCPL	Minimum Complexity	Min(OS_j)
BA	Best Performing Agent	Max(M_l)
WA	Worst Performing Agent	Min(M_l)

$M_l = 0$ when none of the activities of SA_l are scheduled yet, $M_l = 1$ in the opposite case. Note that the rules related to slack, total work content and agent performance are dynamic, i.e. $\mathcal{P}_{ij}$ of activities may change when the partial schedule changes. The other priority rules are invariant with regard to the partial schedule.

When referring to a priority rule in the MPSGS, it will be expressed in the format A-B where A is the APR and B the PPR. For instance, SA-SP refers to the selection of the shortest activity of the shortest project and SASP refers to the priority rule used in previous research, defined by equation 3.3. In order to evaluate interaction effects between the APR and PPR, the performance of each combination is evaluated in section 6.1. Preliminary tests have shown that MINSLK-SP outperformed the other combinations. As the scheduling methods in the remainder of this chapter bear multiple similarities to the MPSGS, they will also use this PR when necessary.

3.2 A game mechanic

The mechanism of a schedule game is inspired by the dynamics of (board) games. In some of these (e.g. Settlers of Catan or Power Grid), players have to build an empire by gathering resources and negotiating with each other. On an abstract level, these games are played in multiple rounds, each consisting of two main phases: (1) a resource gathering phase, and (2) a decision phase. In the first phase, players receive resources based on their strategy and earlier decisions. Some games also allow negotiating about and trading of resources during this step. In the second phase, the players make decisions about how to invest their resources. This process can be based on a multitude of factors (e.g. their resource position and that of their competitors, their overall strategy, etc.). Not only the individual decisions are important, the interaction between the strategies of the different players has an impact on their respective performances too. A distinction can be made between games with and without fairness mechanisms. The former type has explicit rules or implicit dynamics that give additional opportunities to players who perform badly during a certain stage, making it still possible for them to win. Games of the latter type do not have this property, which allows for larger performance gaps between the best and worst performing players. As a result, losing players may get stuck in a vicious circle. In this case, their bad current position negatively affects the possibility to perform well in future rounds. Even games with fairness mechanisms cannot guarantee that this situation will never happen, but the probability of occurrence is lower.

The general concepts of games discussed above will be incorporated in a scheduling scheme (called a scheduling game or game mechanic). The structure of the basic game is shown in pseudo code below, the elaboration will follow its outline. It bears similarities to the MPSGS, but additional mechanisms are added. An iteration refers to a full execution of the game, resulting in a schedule for the multi-project. A round of the game refers to one iteration of the while-loop in algorithm 2. During one game iteration, multiple rounds will be executed. For every problem instance, multiple iterations of the game will be executed. The first iteration is a simplified variant, its resulting schedule is used to estimate initial values for (among others) the makespan of individual projects. Subsequent iterations will use this information. As a degree of randomness is present in the mechanism, the result of any two iterations may be different, so it is advisable to execute multiple iterations.

Algorithm 2: Basic game

```
1  ps = ∅
2  while not all activities scheduled do
3      CA allocates resources to a schedule agent
4      for l in L do
5          SA_l selects a_ij with highest priority in E_l ∪ S_l
6          if a_ij ∈ S_l then
7              SA_l sends schedule order for a_ij
8      if any orders sent then
9          CA selects o_l with highest v(o_l), i.e. the winning order o_w
10         CA schedules activity a_ij of o_w at time E_ij in ps
11         All agents update information based on the new partial schedule
12 return ps
```

3.2.1 Resource allocation (Line 3)

Every SA_l is again responsible for one project and has a resource stack s_{lk} per resource type k. This stack stores the resources that he receives during the game. When the game starts, $s_{lk} = 0,\ \forall l, k$. At the beginning of each round, a resource assignment procedure is executed. The allocation of resources to the agents is based on the vector $\{q_1, ..., q_L\}$, called the quarry. It is established as follows: first, the total resource demand of each agent is calculated:

$$W_l = \sum_{j \in P_l} \sum_{i=1}^{N_j} w_{ijk}. \tag{3.6}$$

Then, the probabilities q_l are computed:

$$q_l = \frac{W_l}{\sum_{l=1}^{L} W_l} \tag{3.7}$$

A lower bound is set on q_l in order to avoid negative or very small values for some agents. In this case it is set to $LB_q = \frac{1}{4 \cdot L}$. The assigned resources are not linked to a specific time window, but should be seen as a currency. Once an agent has enough resources, he can trade them with the coordinating agent (CA) for a timeslot to schedule an activity (cfr. infra).

Three allocation mechanisms were tested. In approach (1), one agent is randomly selected using the selection probabilities $\{q_1, ..., q_L\}$. When an agent is selected, he receives an amount of resources f_k per resource type, his stack increases to $s_{lk} + f_k,\ \forall k$. For each allocation mechanism, f_k is set equal for every resource type: $f_k = f,\ \forall k$. Approach (2) allocates resources

to every agent per round. The amount of resources that A_l receives, is a random amount in the interval $[0, f \cdot q_l]$. On average, agents with higher q_l will receive more resources, but this is not necessarily true in every round. In approach (3), a vector of size V is constructed. Every spot in the vector stores a tuple of agents. Each agent is randomly added to $V \cdot q_l \cdot 2$ different tuples of the vector ($q_l \cdot 2$ has an upper bound of 0.8), so on average 2 agents are present per tuple. Each game round, all agents in a randomly selected tuple receive f units per resource type.

3.2.2 Activity selection (Line 4-7)

An activity will always be present in exactly one of four disjoint sets. The Completed ($\mathcal{C}_l$) and Ineligible ($\mathcal{I}_l$) have the same definition as in section 3.1. However, the decision set from section 3.1 is split in two: the Schedulable ($\mathcal{S}_l$) and Eligible ($\mathcal{E}_l$) set. $\mathcal{S}_l$ consists of the activities for which all predecessors are in $\mathcal{C}_l$ and $s_{lk} \geq w_{ijk},\ \forall k$. For the activities in the eligible set $\mathcal{E}_l$, the former condition also holds, but $s_{lk} < w_{ijk}$ for at least one k. Separate instances of these four sets are assigned to every agent. After the allocation of resources, every agent evaluates whether $s_{lk} \geq w_{ijk},\ \forall k$ holds for any $a_{ij} \in \mathcal{E}_l$. If so, these activities are migrated from $\mathcal{E}_l$ to $\mathcal{S}_l$. Then SA_l searches a_{ij} with the highest priority value in $\mathcal{E}_l \cup \mathcal{S}_l$. If this $a_{ij} \in \mathcal{S}_l$, he sends a schedule order o_l to the coordinating agent for that activity. This order has the same structure as in section 3.1. If $a_{ij} \in \mathcal{E}_l$, the agent needs more resources to send the order and will take no action at the current time.

As stated in section 3.1, MINSLK-SP performed the best during preliminary tests, so it will be used in the scheduling game. This means that schedule agents give priority to activities with the least slack and that $v(o_l)$ is equal to the CP duration of the corresponding project. The coordinating agent will select the order with the lowest $v(o_l)$.

Next to the deterministic variant, an adaptation of MINSLK-SP is proposed, based on random sampling. This is implemented to better approach the behavior of players in a game, where decisions often are not based on one deterministic rule. The approach formulated in Kolisch et al. (1995) is followed. Let ρ_{ij} denote the regret value of a_{ij}, defined by:

$$\rho_{ij} = \begin{cases} \mathcal{P}_{ij} - \mathcal{P}_{min} & \text{if } \mathcal{P}_{ij}\text{is maximized,} \\ \mathcal{P}_{max} - \mathcal{P}_{ij} & \text{if } \mathcal{P}_{ij}\text{is minimized,} \end{cases} \tag{3.8}$$

where $\mathcal{P}_{min}$, $\mathcal{P}_{max}$ are respectively the minimal and maximal priority value of all activities in $\mathcal{E}_l \cup \mathcal{S}_l$ of the agent that is responsible for a_{ij}. The selection probability of a_{ij} is calculated using

$$\psi_{ij} = \frac{(1+\rho_{ij})^\eta}{\sum_{a_{ij}\in\mathcal{E}_l\cup\mathcal{S}_l}(1+\rho_{ij})^\eta} \ , \tag{3.9}$$

η is a parameter defining the pressure on the selection. The higher η, the higher the probability that activities with a high regret value will be selected. Every time SA_l has scheduled an activity, he first updates $\mathcal{E}_l \cup \mathcal{S}_l$. Afterwards, he randomly selects the new activity that will receive priority, using ψ_{ij}. Activities with a good $\mathcal{P}_{ij}$ still have a high probability of being selected, but there is no guarantee that the best will always be selected.

3.2.3 Order selection and scheduling (Line 8-11)

All schedule agents reevaluate their eligible and schedulable sets in parallel. As a consequence, multiple schedule orders can be sent in one round. If this is the case, the coordinating agent ranks the orders according to their value $v(o_l)$. Then he selects the best ranked one and adds the according a_{ij} to the partial schedule at time E_{ij}. Subsequently he subtracts $\{w_{ij1}, ..., w_{ijk}\}$ from the resource stack of the winning agent SA_w.

After an activity is scheduled, the agents update their available information. First, SA_w moves the scheduled activity a_{ij} from $\mathcal{S}_l$ to $\mathcal{C}_l$. Second, he places activities from $\mathcal{I}_l$ in $\mathcal{E}_l$ or $\mathcal{S}_l$ if all predecessors are scheduled in the new partial schedule. Third, he migrates activities from $\mathcal{S}_l$ to $\mathcal{E}_l$ for which $s_{lk} \geq w_{ijk}, \ \forall k$ is no longer valid. This is possible as the available resources in his stack have decreased in the scheduling step. At last, all agents update the $E_{ij}, \ \forall \ a_{ij} \in \mathcal{E}_l \cup \mathcal{S}_l$. As a new activity has been scheduled, the availability of resources (R'_{kt}) in the scheduled time slots will have decreased. There may not be enough resources left for other activities having E_{ij} in that time window. For these activities, the schedule agent has to find the new E_{ij}.

3.3 Extensions to the basic game

In section 3.2, the agents can be classified as reactive. They wait for the allocation of resources and send a schedule order for the activity with the highest priority value (i.e. they follow a single rule). To move agents into the direction of more deliberative action, their decision process is augmented. Four extensions are implemented to make the strategy of agents less unidirectional. Subsections 3.3.1 and 3.3.2 are addressed at the level of the schedule agents, subsections 3.3.3 and 3.3.4 are addressed at the level of the coordinating agent.

3.3.1 Tightness evaluation

In this section a trade off mechanism between long term and short term improvements is implemented. On the long term side, an agent will always wait until he has the resources to schedule the activity with the highest priority in $\mathcal{E}_l \cup \mathcal{S}_l$. On the short term side, an agent will also evaluate all activities in $\mathcal{S}_l$, i.e. those that can be immediately scheduled with the current resource stack. The evaluation is performed on the first time window in which the activity can be scheduled, i.e. $[E_{ij}, E_{ij} + d_{ij}[$. To make this evaluation, two factors are taken into account: (1) the resource tightness of the window and (2) the competition from other activities.

The resource tightness for a_{ij} is calculated as follows:

$$\mathcal{T}_{ij} = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=E_{ij}}^{E_{ij}+d_{ij}-1} \frac{r_{ijk}}{R'_{kt}}. \tag{3.10}$$

R'_{kt} is the amount of resource type k that is still available at time point t. When $\mathcal{T}_{ij} \to 1$, the resources in the time window become more constrained for a_{ij}. In the extreme case where $\mathcal{T}_{ij} = 0$, there will always be enough resources available for a_{ij}. We can conclude that the higher $\mathcal{T}_{ij}$, the higher the probability that a_{ij} will be delayed if another activity is scheduled in its time window.

The second factor is the measure of competition from other activities. The count of competing activities for any a_{ij} is defined as follows

$$\mathcal{C}_{ij} = \sum_{g=1}^{J} \sum_{h=1}^{N_g} (a_{gh} : E_{gh} \in [E_{ij}, E_{ij} + d_{ij}[\wedge (g \neq i \vee h \neq j)), \tag{3.11}$$

i.e. an activity a_{gh} counts as competing when its E_{gh} lies in the earliest precedence and resource feasible time window of a_{ij}. If a_{gh} would be scheduled first, it is possible that E_{ij}

has to be delayed. The total tightness of an activity is $\mathcal{T}_{ij} \cdot \mathcal{C}_{ij}$. The tightness based selection mechanism of agent SA_l follows this outline:

Algorithm 3: Tightness evaluation

1 $T = \emptyset$; initialize τ, κ
2 a_{top} = activity with the best priority value in $\mathcal{E}_l \cup \mathcal{S}_l$
3 **for** *all* a_{ij} *in* $\mathcal{S}_l$ **do**
4 **if** $\frac{\mathcal{T}_{ij} \cdot \mathcal{C}_{ij}}{\mathcal{T}_{top} \cdot \mathcal{C}_{top}} \geq \tau$ **then**
5 $T = T \cup a_{ij}$
6 $\phi = \exp\left(\frac{-|T|/|\mathcal{S}_l|}{\kappa}\right)$
7 **if** *rand(0,1)* $\leq \phi$ **then**
8 **if** $a_{top} \in \mathcal{S}_l$ **then**
9 Select a_{top}
10 **else**
11 Select $a_{ij} \in T \setminus \{a_{top}\}$ with the highest $\mathcal{P}_{ij} \cdot \mathcal{T}_{ij} \cdot \mathcal{C}_{ij}$

τ is a threshold for the tightness values, a higher value will result in less activities being added to T. If $\tau = 1$, only activities experiencing at least as much competition as a_{top} are added to T. In line 6 the probability ϕ of selecting a_{top} is determined (which means that SA_l does not change his strategy). This probability is influenced by the count of activities in T and by κ. If a lot of activities experience high competition (i.e. a high $|T|$), the probability of selecting an alternative activity becomes higher. If κ increases, ϕ will increase. Note that line 11 only applies when the PR is a maximization rule. In the opposite case, the activity with the lowest $\frac{\mathcal{P}_{ij}}{\mathcal{T}_{ij} \cdot \mathcal{C}_{ij}}$ should be selected. Both the priority and tightness value are incorporated in the selection of an alternative activity. In this mechanism, two parameters are available for tuning: τ and κ.

3.3.2 Negotiation mechanism

The second approach to augmenting the behavior of the schedule agents will introduce a negotiation mechanism. The SA's communicate with each other through the CA. The latter makes the final decision regarding the negotiations. The procedure will activate after line 9 in algorithm 2. The general structure is inspired by the mechanism proposed in Agnetis et al. (2015).

Step 1: When CA has selected the winning order (o_w), he communicates this to all schedule agents.

Step 2: All schedule agents that did not send the order (losing agents) will create two local schedules, building on the current partial schedule. In the first one, they schedule all their remaining activities under the assumption that no other agents can add any activity. In the second, they schedule the activity of o_w first and afterwards again create their own local schedule, neglecting the actions of other agents. The increase in makespan of the latter in comparison to the former is denoted as δ_l. This is the deterioration of the minimal makespan for SA_l if the activity from o_w would be added to the partial schedule. Then, all losing agents construct a compensating order co_l for the activity in $\mathcal{E}_l \cup \mathcal{S}_l$ with the highest priority and send this to CA.

Step 3: CA evaluates these orders based on their value $v(co_l)$, which is given by the following formula (assume that a_{ij} is the activity linked to co_l)[2]:

$$v(co_l) = \frac{\delta_l}{\sum_{k=1}^{K} \max(\pi_k(w_{ijk} - s_{lk}), 1)}. \tag{3.12}$$

CA selects co_l^*, the order with the highest $v(co_l)$. Orders with a high possible makespan reduction (δ_l) will have a higher probability to be selected. However, if an agent still needs a lot of resources to schedule a_{ijk}, $v(co_l)$ decreases. The resources are weighed with their cost π_k, as defined in Lawrence and Morton (1993):

$$\pi_k = \frac{1}{R_k} \cdot \sum_{j=1}^{J} \sum_{i=1}^{N_j} w_{ijk} \ , \forall k. \tag{3.13}$$

CA sends co_l^* to the winning schedule agent.

Step 4: The winning agent creates two local schedules to evaluate the impact on his minimal makespan if co_l^* would receive priority over o_w, (this is similar to step 2). He then communicates the deterioration δ_w to CA.

Step 5: If $\delta_w < \delta_l$, CA will give priority to co_l^*. This means that the corresponding activity is scheduled before the one of o_w. The agent that sent co_l^* may need additional resources before it can be scheduled, as he could select an activity from $\mathcal{E}_l$. CA allocates the required resources, but all other agents receive the same amount to retain a fair treatment. If $\delta_w \geq \delta_l$, only the activity from the original o_w is scheduled.

[2]Note that $v(co_l)$ is different from $v(o_l)$ defined in section 3.2

This negotiation mechanism introduces some kind of fairness. When the scheduling of an activity would have a serious detrimental impact on the makespan of an agent, he still has the opportunity to propose a compensating order. However, this compensating order will not always be given priority, this would disrupt the normal scheduling process too much. In order to reduce computational intensity, this negotiation mechanism is not triggered every time an order is sent. The probability that it is activated is denoted by ν, the only parameter that can be tuned.

3.3.3 Load Based Quarry Updating (LBQU)

In the basic game, CA allocates resources to SA's using the probabilities q_l. These calculations are based on the resource demand over the whole portfolio duration (equation 3.7). However, the resource demand of an agent may vary over the course of the schedule. Dynamically adapting q_l to this situation may improve the general performance of the game mechanism. Therefore, CP_{max} is split into S equal time intervals and the total resource demand over the critical path duration per agent per interval $s \in S$ is calculated:

$$W_{ls} = \sum_{t=a}^{b} \sum_{j \in P_l} \sum_{i=1}^{N_j} \sum_{k=1}^{K} r_{ijk} X_{ijt}. \tag{3.14}$$

Assume that interval s spans $t \in [a, b]$. $X_{ijt} = 1$ if a_{ij} is active at t according to the resource-unconstrained critical path schedule, $X_{ijt} = 0$ otherwise (cfr. equation 4.3 for more detail). This equation is similar to 4.7, but here it is aggregated over all resource types and split per agent. The adapted probability $\hat{q}_{ls}$ of agent l during interval s is given by

$$\hat{q}_{ls} = \frac{W_{ls} \cdot S}{W_l} q_l. \tag{3.15}$$

The resource load during an interval is compared to the average resource load over the whole portfolio length. SA_l will have a higher $\hat{q}_{ls}$ during intervals where his resource demand is relatively high and vice versa. However,

$$\sum_{l=1}^{L} \hat{q}_{ls} = 1, \ \forall s \in S \tag{3.16}$$

will not necessarily hold as there is no guarantee that in any s the decrease in resource demand of some agents exactly levels out the increase of others. Therefore, $\hat{q}_{ls}$ has to be rescaled such

that 3.16 is valid again:

$$q'_{ls} = \frac{\hat{q}_{ls}}{\sum_{l \in L} \hat{q}_{ls}}, \ \forall l \in L, s \in S. \tag{3.17}$$

Once the probabilities are obtained, the intervals s have to be rescaled. The initial interval lengths d_s are based on CP_{max}. However, due to resource constraints the makespan may be longer than the critical path duration. In the initial iteration of the schedule game, q_l are used instead of q'_{ls}. The resulting schedule is used to calculate the inflation of the makespan (in comparison to CP_{max}), the d_s are increased with this same factor. In subsequent iterations of the game, $\{q'_{1s}, ..., q'_{Ls}\}$ are used with the rescaled interval lengths. During the course of one iteration, the probabilities are adapted dynamically. Let s be the current interval that is used for q'_{ls}. If at any time, more than half of the activities in $\mathcal{E}_l \cup \mathcal{S}_l$ of all agents have E_{ij} in $s+1$ or later, the probabilities will be adapted to $q'_{l,s+1}$.

The main parameter that can be set for this mechanism is the amount of intervals S. More intervals will provide more granularity in the resource load information and should increase the performance. However, it is expected that the performance will improve in a digressive manner. For instance, increasing S from 1 to 2 should provide more useful information than increasing S from 9 to 10.

3.3.4 Performance Based Quarry Updating (PBQU)

A second way of adapting q_l is by dynamically evaluating the current performance of the SA's. $\{q''_1, ..., q''_L\}$ denotes the quarry after the performance based adaptations. The performance M_l of agent l is defined by equation 3.5. It takes the ratio of all activities in the completed set and the total number of activities that SA_l is responsible for. Every time an activity is scheduled, $\{q''_1, ..., q''_L\}$ is revised. All agents are ranked in descending order according to M_l. Let r_l be the respective rank of an agent. The adapted probabilities are calculated as follows:

$$q''_l = \max(q_l - \Delta_{max} + (r_l - 1) * \Delta, LB_q) \tag{3.18}$$

with

$$\Delta_{max} = \frac{1}{L \cdot \omega} \tag{3.19}$$

and

$$\Delta = \frac{2 \cdot \Delta_{max}}{L - 1}. \tag{3.20}$$

Δ_{max} is the maximum deviation from q_l for any agent. $q_l'' = q_l - \Delta_{max}$ if $r_l = 1$ and $q_l'' = q_l + \Delta_{max}$ if $r_l = L$. For all other SA, the deviation will lie in the interval $]-\Delta_{max}, \Delta_{max}[$. Note that the only parameter available for tuning is ω. By setting it, one determines Δ_{max} and Δ. The smaller ω, the more the q_l'' will be influenced by M_l. 3.18 does not allow setting a q_l'' below the lower bound LB_q. This restriction is imposed in order to avoid that probabilities become very small or even negative. This PBQU introduces a different fairness mechanism as: agents that currently are not performing well get a higher probability of receiving resources and vice versa.

Chapter 4

Dataset generation

Multiple generators for RCPSP instances exist (Demeulemeester et al., 1993; Kolisch et al., 1995; Demeulemeester et al., 2003). Research regarding the generation of RCMPSP instances is less prevalent. Multiple authors have created datasets without taking into account resource usage or network complexity measures (cfr. supra). Browning and Yassine (2010a) create a generator for RCMPSP problems. They discuss four important input parameters: the complexity C_j [1], the Normalized Average Resource Loading Factor (NARLF) and the Modified Utilization Factor (MAUF) and its variance σ^2_{MAUF}. First these four parameters and an alternative for the complexity (Order Strength) will be discussed, then a generation procedure will be proposed.

4.1 Parameters

To understand the first parameter (C_j), the concept of tiers is introduced. In a network with N activities, they define a tier "...as a subset of the N activities (a) with no arcs between them, and (b) that depend only on activities from lower tiers." (Browning and Yassine, 2010a). The authors use this concept to analyze the degree of serialism: more tiers mean a more serial network and vice versa. Note that it bears similarities to the indicator Progressive Level (PL), defined by Elmaghraby (1977). The resulting complexity of a project j is expressed as

$$C_j = \frac{A' - A'_{min}}{A'_{max} - A'_{min}} \tag{4.1}$$

[1] The authors use the notation C_l as they indicate a project with index l. It is adapted here to j for consistency reasons.

where A' is the number of non-redundant arcs, A'_{min} and A'_{max} respectively the lower and upper bound on A' in a network with N_j nodes and T tiers. An arc $\langle h, j \rangle$ connecting activities h and j in a network is redundant if a series of arcs $\langle i_0, i_1 \rangle, ..., \langle i_{s-1}, i_s \rangle$ exist with $i_0 = h, i_s = j$ and $s \geq 2$ (Kolisch et al., 1995).

The generation procedure using C_j has multiple drawbacks, which will be discussed in section 4.2. The parameter Order Strength (OS) will be used as alternative measure for network complexity. Mastor (1970) defines it as the number of transitive and non-transitive arcs (not including those connected to the dummy start or end node) divided by the theoretical maximum number of arcs in a network. E.g. in a network with three nodes (X,Y,Z) and two arcs $\langle X, Y \rangle$, $\langle Y, Z \rangle$, these two arcs are non-transitive arcs. The precedence relationship $\langle X, Z \rangle$ is implied by the former two arcs and is called transitive. The latter is not explicitly incorporated in the network, but is taken into account for the calculation of OS. RanGen 1, proposed in Demeulemeester et al. (2003), is capable of generating networks without redundant arcs based on a given OS without reducing the space of feasible networks that it evaluates.

The second parameter is NARLF. Kurtulus and Davis (1982) define the ARLF to indicate whether the total resource requirements of a project mainly occurs in its first or second half. To compute it, the resource-unconstrained critical path is constructed first. Then the following equations are used:

$$Z_{ijt} = \begin{cases} -1 & t \leq CP_j/2, \\ 1 & t > CP_j/2 \end{cases} \tag{4.2}$$

$$X_{ijt} = \begin{cases} 1 & \text{if activity } i \text{ of project } j \text{ is active at time } t, \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

$$ARLF_j = \frac{1}{CP_j} \sum_{t}^{CP_j} \sum_{k=1}^{K_{ij}} \sum_{i=1}^{N_j} Z_{ijt} X_{ijt} \left(\frac{r_{ijk}}{K_{ij}} \right) \tag{4.4}$$

$$ARLF = \frac{1}{J} \sum_{j=1}^{J} ARLF_j \tag{4.5}$$

Browning and Yassine (2010a) propose an adapted Normalized ARLF (NARLF):

$$NARLF = \frac{1}{J * CP_{max}} \sum_{j=1}^{J} \sum_{t=1}^{CP_j} \sum_{k=1}^{K_{ij}} \sum_{i=1}^{N_i} Z_{ijt} X_{ijt} \left(\frac{r_{ijk}}{K_{ij}} \right). \tag{4.6}$$

Equation (4.4) calculates the ARLF of every individual project relative to its own CP duration. If the durations of the projects differ significantly, this measure provides misleading results (Browning and Yassine, 2010b). (4.6) partially solves this issue, as the resource distribution of every project is normalized over the duration of the whole portfolio. However, this adaptation is not sufficient (cfr. section 4.3).

The third parameter is MAUF. Kurtulus and Davis (1982) propose the AUF to indicate the resource demand of the portfolio relative to the available resources. Once again, the resource-unconstrained schedule is created. CP_{max} is divided in S intervals. Then, the required amount of resource k over an interval $s \in S : t \in [a, b]$ is computed by

$$W_{sk} = \sum_{t=a}^{b} \sum_{j=1}^{J} \sum_{i=1}^{N_i} r_{ijk} X_{ijt}. \tag{4.7}$$

The AUF is calculated as follows:

$$AUF_k = \frac{1}{S} \sum_{s=1}^{S} \frac{W_{sk}}{R_k d_s} \tag{4.8}$$

where d_s is the duration of interval s. If $AUF_k > 1$, the demand for resource k is –on average– higher than the available amount. The AUF of a problem with K resource types is

$$AUF = Max(AUF_1, ..., AUF_K). \tag{4.9}$$

Kurtulus and Davis (1982) use intervals that can have different lengths, which may obscure the situation (Browning and Yassine, 2010b). To counter this, Browning and Yassine (2010a) average the resource utilization over equal intervals and call their measure the Modified AUF (MAUF). Equations 4.7 - 4.9 still hold, only S is determined differently.

Using only (M)AUF may cause biases, as it neglects information about less constrained resource types. To get a view on the contention of all resources, Browning and Yassine (2010a) introduce the variance of the utilization:

$$\sigma^2_{MAUF} = \frac{\sum_{k=1}^{K} (MAUF - MAUF_k)^2}{K} \tag{4.10}$$

This is an atypical variance, as it measures the variance from the maximum. A higher σ^2_{MAUF} will occur if the resource contention of other resource types deviates further from the most constrained one. Ceteris paribus, a higher σ^2_{MAUF} should result in less delays (Browning and Yassine, 2010b).

4.2 Generation procedure

Multiple objections can be made about the generation procedure of Browning and Yassine (2010a). First, postulating a required amount of tiers inhibits the claim of generating *strongly random* networks. A generator is strongly random when it generates networks "...at random from the space of all feasible networks with a specified number of nodes and arcs" (Demeulemeester et al., 2003). By specifying an amount of tiers, the space from which networks are drawn may be smaller than the full feasible space. Second, the generator starts with specifying for each activity in which tier it should be present. However, the resulting project network does not guarantee that the actual tier of an activity is the same as its specified tier. When reviewing some of the generated networks, it often occurred that activities with a specified tier > 1 had an actual tier $= 1$ as they had no predecessors. The concept of tiers is introduced as a measure of serialism of a network. However, intances appear to be a lot more parallel than they should based on the specified amount of tiers. This leads to the conclusion that the generator cannot guarantee the postulated serialism of a network[2]. Third, the complexity measure defined by the authors is no good predictor of network complexity. The authors generate networks with $C_j = 0.14$ and $C_j = 0.69$. For 84 generated networks of each complexity level, the OS was calculated. As shown in table 4.1, the resulting OS for the two complexity levels is not that different, the more complex networks still have a very low Order Strength. The variations at a certain level can be explained by a different amount of tiers, networks with OS = 0.11 have more tiers than networks with OS = 0.07. It seems that the amount of tiers has a (minor) impact on network complexity, its impact would be higher if the generator would respect the postulated amount of tiers. As a difference of 0.55 in C_j corresponds only with a maximum change in OS of 0.08, it is dubious that it spans the same range of complexity as OS. Fourth, both the amount of tiers and the network complexity measure have an impact on the network complexity. They have to be evaluated simultaneously which may complicate the resulting analyses. It is easier to have one measure that indicates the complexity. Fifth, specifying a number of tiers can reduce the probability of successful generation attempts. Let T_{max} be the maximum possible amount of tiers for a

[2]This objection is of a qualitative nature. To retain the focus of this dissertation, no profound quantitative analysis of the generated networks was executed to back up the objection. This is an opportunity for further research.

Table 4.1: Comparison between C_j and OS

C_j	Range of OS
0.14	[0.03, 0.06]
0.69	[0.07, 0.11]

given C_j. Browning and Yassine (2010a) conclude that when the amount of tiers $T \to T_{max}$, the probability of generating a network with a certain complexity C_j declines rapidly. As a consequence, generating these networks is computationally expensive and may take a long time. A last objection is of a more practical nature: the proposed generator is implemented in an Excel format for 3 projects with 20 activities. The configuration is not flexible in adapting these parameters, which reduces the applicability in other research efforts. The generator itself is coded in VBA in Excel, which is not computationally efficient.

To deal with the previously discussed issues, a new multi-project generator is proposed. The generated multi-project instances have the following parameters: $J = 5$, $N_j = 25$, $\forall j \in J$, $K = 4$. The general structure of the generated dataset follows Browning and Yassine (2010a). $NARLF \in [-3, -2, ..., 2, 3]$, $MAUF \in [0.6, 0.8, ..., 1.4, 1.6]$. Although values outside these boundaries are possible, most instances will not exceed these intervals or they become hard to generate (Browning and Yassine, 2010a). σ^2_{MAUF} can assume two levels: 0 or $[0.15, 0.20]$. Using the first level, $MAUF_k$ will be exactly the same for all resource types. For the second level, $MAUF_k$ of less-constrained resource types will be lower than $MAUF$. For low values of $MAUF$, it may be hard to find a combination that meets exactly $\sigma^2_{MAUF} = 0.2$. Therefore, it is allowed to be inside the interval of $[0.15, 0.20]$. The complexity of the instance is represented by a vector of complexity values per single-project. Three complexity levels are used: H (OS = 0.75), M (OS = 0.5), L (OS = 0.25). The vector HHHML means that 3 projects with High, 1 with Medium and 1 with Low complexity constitute the multi-project. Instances are generated over 7 complexity vectors: HHHHH, MMMMM, LLLLL, HHHML, HMMML, HMLLL, HHMLL. For every parameter combination, n = 25 instances are generated to reduce the impact of random factors. This results in a dataset size of $7 \cdot 6 \cdot 2 \cdot 7 \cdot 25 = 14700$ (NARLF $\cdot$ MAUF $\cdot$ σ^2_{MAUF} $\cdot$ OS $\cdot$ n) multi-project networks.

The adapted generation procedure consists of two steps: (1) generate single-project networks with specified levels of OS using RanGen 1, (2) combine these networks to RCMPSP-instances

and adapt the resource usage to meet specified levels for the resource related parameters. The resource tuning procedures are based on Browning and Yassine (2010a) but some improvements were added.

4.2.1 Single-project generation

RanGen 1 is used to generate single-project instances with the specified values for OS. The input values for resource measures are neglected, as they will be tuned in 4.2.2. For one instance with a specified complexity vector, multiple single-project instances are required (see Table 4.2). Per complexity vector 2100 ($7 \cdot 6 \cdot 2 \cdot 25$) instances need to be generated. Based on Table 4.2, this would require 25200, 23100 and 25200 single-project instances with OS = 0.75, 0.5 and 0.25 respectively. To reduce memory requirements, respectively 2940, 2694 and 2940 networks where generated. These where added to a pool from which the procedure in 4.2.2 randomly draws networks to combine. This means that on average, each single-project instance will be selected 8.57 times. With a total multi-project dataset size of 14700, the impact of a specific network should not have a significant bias on the resulting analysis.

Table 4.2: Amount of required single-projects

Complexity vector	Single-project count		
	H	M	L
HHHHH	5	0	0
MMMMM	0	5	0
LLLLL	0	0	5
HHHML	3	1	1
HMMML	1	3	1
HMLLL	1	1	3
HHMLL	2	1	2
Total	12	11	12

4.2.2 Combination and resource tuning

Let P_H, P_M, P_L be the complete sets with single-project networks of their respective complexity. S_H, S_M, S_L are the pools from which the networks will be drawn by the generator. $\mathcal{V}$ is

the vector containing the complexity values (e.g. HHHHH), $\mathcal{V}_j$ the complexity of project j in the multi-project. *spn* is a single-project network, m the multi-project instance, G the set of successfully generated instances. X is a temporary variable that can store a complexity value from {H,M,L}. The sample size is n (25 per parameter combination). Algorithm 4 describes the generation of all instances for a given complexity vector $\mathcal{V}$, NARLF, MAUF and σ^2_{MAUF}. In the current design, the algorithm will stop after successfully generating 25 instances and return the set G of all generated multi-projects.

Algorithm 4: generateMultiProjects(NARLF, MAUF, σ^2_{MAUF}, n)

```
1  S_H = P_H; S_M = P_M; S_L = P_L; G = ∅
2  initialize spn, m
3  successCount = 0
4  while successCount < n do
5  |  reset m
6  |  forall j in J do
7  |  |  X = V_j
8  |  |  spn = random network from S_X
9  |  |  add spn to m
10 |  |  S_X = S_X \ {spn}
11 |  |  if S_X = ∅ then
12 |  |  |  S_X = P_X
13 |  if tuneNARLF(NARLF, m) = success then
14 |  |  if tuneMAUF(MAUF, σ²_MAUF, m) = success then
15 |  |  |  successCount ++
16 |  |  |  G = G ∪ m
17 return G
```

Note that every time a network is selected from S_X, it is removed from that set (line 10). This imposes equal selection probabilities on each network. When the set is empty, all networks are added again (line 12). However, this is not a guarantee that they will be equally present in the resulting multi-project instances. Generation attempts can fail further down the algorithm, resulting in networks not being added to the set G. After the construction of m, the resource demand of its activities is tuned to meet the specified $NARLF$, $MAUF$ and σ^2_{MAUF} values (lines 13 and 14). These steps are elaborated in Algorithm 5 and Algorithm 6.

This paragraph discusses the procedure tuneNARLF (algorithm 5). Let $NARLF_m$ be the current $NARLF$-value of network m. α is a specified deviation parameter. When the deviation from the desired value $NARLF_{des}$ becomes smaller than α, the algorithm stops. This to avoid excess iterations for small improvements. For this design, $\alpha = 0.01$. If, after $maxIterations$ (=2500 in this case) iterations, $NARLF_m$ has not converged to its target and the tuning attempt is considered a failure (line 5). This results in the termination of the respective iteration in algorithm 4. In every iteration, r_{ijk} is adapted for a random i, j and k. If $NARLF_m < NARLF_{des}$, the resource load has to be shifted to the second half of the multi-project (and vice versa). This is done by incrementing r_{ijk} if a_{ij} falls mainly in the back half of the CP duration, and decrementing it otherwise. The condition in lines 10 and 18 avoid creating activities with $K_{ij} = 0$ or $r_{ijk} < 0$. If $NARLF_m$ reaches its target before the iteration limit is reached, tuneNARLF will return success.

If tuneNARLF was executed successfully, tuneMAUF (algorithm 6) will be started. First of all, generateMAUF (algorithm 7, cfr. infra) creates MAUF-values for all resource types to meet the specified $MAUF_{des}$ and $\sigma^2_{MAUF,des}$, which are stored in the set MV. MV_k refers to the desired MAUF-value for resource type k. Then the amount of available resources R_k is set to the value for which $MAUF_k$ approaches MV_k most closely (Line 3). Using the formula

$$R_k = \frac{W_k}{CP_{max} \cdot MV_k}, \tag{4.11}$$

with W_k being the total resource demand for type k by the whole portfolio, R_k can be calculated[3]. Using the new availability, the actual $MAUF_k$ is calculated (line 7). In the whole procedure, two errors can occur. First, the resource demand of an any activity could exceed this available amount (line 4). As there would never be enough of resource k available to meet its demand, it would never be eligible to be scheduled by for instance a SGS. Second, $MAUF_k$ could deviate too far from the postulated MV_k due to rounding in line 3. This deviation should not exceed β (=0.025). If one of these two errors occurs, the generation attempt is considered a failure and the respective iteration in algorithm 4 is terminated. Otherwise, the method returns success, which results in the multi-project m being added to G.

Now, generateMAUF will be discussed (algorithm 7). This procedure determines MV. If $\sigma^2_{MAUF,des} = 0$, the problem reduces to $MV_k = MAUF_{des}$, $\forall k$. When $\sigma^2_{MAUF,des} > 0$,

[3]Equation 4.11 is equivalent to equation 4.8 with one interval of length CP_{max}

Algorithm 5: tuneNARLF($NARLF_{des}$, m)

```
1  iteration = 0
2  Calculate CP_m
3  while |NARLF_m − NARLF_des| > α do
4      if iteration ≥ maxIterations then
5          return failure
6      else
7          i, j, k = select random non-dummy activity, project and resource type from m
8          if NARLF_m < NARLF_des then
9              if ES_ij + ⌈d_ij/2⌉ < ⌈CP_max/2⌉ then
10                 if (r_ijk > 1 or K_ij > 1) and r_ijk > 0 then
11                     r_ijk −−
12             else
13                 r_ijk ++
14         else
15             if ES_ij + ⌈d_ij/2⌉ < ⌈CP_max/2⌉ then
16                 r_ijk ++
17             else
18                 if (r_ijk > 1 or K_ij > 1) and r_ijk > 0 then
19                     r_ijk −−
20     iteration ++
21 return success
```

Browning and Yassine (2010a) propose setting $MV_1 = MAUF_{des}$ and $MV_k = MAUF_{des} - \sqrt{\sigma^2_{MAUF,des}}$, $\forall k > 1$. This results in resource type 1 being the most constrained and all others less but equally constrained. However, the validity of this composition can be questioned as the utilization of these resources is exactly the same. In realistic problem instances it is highly improbable that all but one resources have exactly the same utilization. Therefore, a new approach is proposed. Resource type $k = 1$ is again the most constrained but MV_k, $\forall k > 1$ can assume values within a range. First, the lower bound for MV_k is calculated (line 6). Take the extreme case where $MV_k = MAUF_{des}$ $\forall k \in K \setminus \{x\}$. Using equation 4.10, only MV_x contributes to σ^2_{MAUF} as $MAUF_{des} - MV_k = 0$ $\forall k \neq x$. Isolating MV_x results in:

$$MV_x = MAUF_{des} \pm \sqrt{\sigma^2_{MAUF,des} \cdot K}. \tag{4.12}$$

Algorithm 6: tuneMAUF($MAUF_{des}$, $\sigma^2_{MAUF,des}$,m)

```
1  MV = generateMAUF(MAUF_des, σ²_MAUF,des)
2  for k in K do
3      R_k = round(W_k / (CP_max · MV_k))
4      if R_k < max_{i,j}(r_ijk) then
5          return failure
6      else
7          MAUF_k = W_k / (CP_max · R_k)
8          if |MAUF_k − MV_k| > β then
9              return failure
10 return success
```

σ^2_{MAUF} is a variance from the maximum, so $MV_k \leq MAUF_{des}\ \forall k$. As a consequence, equation 4.12 reduces to

$$MV_x = MAUF_{des} - \sqrt{\sigma^2_{MAUF,des} \cdot K}. \tag{4.13}$$

This value is a lower bound (LB_σ). If MV_k would decrease further, σ^2_{MAUF} would exceed the $\sigma^2_{MAUF,des}$ resulting in an undesired outcome. If the extreme case is relaxed ($MV_k\ \forall k > 1$ can deviate from $MAUF_{des}$), each resource type can contribute to a further increase in σ^2_{MAUF}. We can conclude that if any $MV_k < LB_\sigma$, this will guarantee $\sigma^2_{MAUF} > \sigma^2_{MAUF,des}$. However, using equation 4.13 allows that $MV_x < 0$, resulting in a negative resource utilization. To avoid this, line 6 imposes $LB_\sigma \geq 0.1$. $MV_k,\ \forall k > 1$ are assigned random values in $[LB_\sigma, MAUF_{des}]$. Then, the current σ^2_{MAUF} is calculated and compared to its target. If the deviation is larger than γ (=0.01), a random MV_k with $k > 1$ is incremented or decremented to converge to the target. This is iterated until the deviation from $\sigma^2_{MAUF,des}$ is smaller than γ. The conditions in lines 15 and 18 ensure that $MV_k \in [LB_\sigma, MAUF_{des}]$.

Algorithm 7: generateMAUF($MAUF_{des}$, $\sigma^2_{MAUF,des}$)

$MV = \emptyset$, $LB_{var} = 0$
if $\sigma^2_{MAUF,des} = 0$ **then**
  for k *in* K **do**
    $MV_k = MAUF_{des}$
else
  $LB_\sigma = \max(MAUF_{des} - \sqrt{\sigma^2_{MAUF,des} \cdot K}, 0.1)$
  for k *in* K **do**
    if $k = 1$ **then**
      $MV_k = MAUF_{des}$
    else
      $MV_k = LB_\sigma + (MAUF_{des} - LB_\sigma) \cdot rand(0, 1)$
  while $|\sigma_{MAUF} - \sigma^2_{MAUF,des}| > \gamma$ **do**
    Select random MV_k from MV with $k > 1$
    if $Var(MV) > \sigma^2_{MAUF,des}$ **then**
      if $MV_k \leq MAUF_{des} - .05$ **then**
        $MV_k = MV_k + .05$
    else
      if $MV_k \geq LB_\sigma + .05$ **then**
        $MV_k = MV_k - .05$
return MV

4.3 Concluding remarks

The multi-project generator introduced here copes with some of the limitations of the generator from Browning and Yassine (2010a). First, the networks are generated with RanGen 1, which has a better claim on generating strongly random networks because it "does not incorporate superfluous parameters [e.g. the number of tiers] ... [and] because the use of these superfluous parameters dramatically restricts the domain from which they are generated" (Demeulemeester et al., 2003). For two rather distinct levels of C_j, the corresponding difference in OS was very small (cfr. supra). This is another indication that the incumbent multi-project generator is incapable of generating networks from the complete domain of networks. Second, the resource utilization levels used to calculate σ^2_{MAUF} are obtained in a more

realistic manner. Third, the new generator is also more flexible in the generated format of multi-project instances and is faster as it is implemented in C++.

However, some critical remarks remain. First, the NARLF-measure (4.6) is still biased. It evaluates the resource distribution relative to the *project's* duration instead of to the *portfolio's* duration. In certain cases, this can still lead to erroneous conclusions, as shown below. To resolve this issue, equation 4.2 is adapted as follows:

$$\bar{Z}_{ijt} = \begin{cases} -1 & t \leq CP_{max}/2, \\ 1 & t > CP_{max}/2 \end{cases} \tag{4.14}$$

When the NARLF is computed with $\bar{Z}_{ijt}$ instead of Z_{ijt}, no biases due to differences in CP durations will occur. Take a multi-project consisting of projects p_1 and p_2. $CP_1 = 6$, $CP_2 = 8$, as a consequence $CP_{max} = 8$. The resource profiles for both projects are shown in the left side of figure 4.3. The load is defined as the resource demand that falls in the first half of the project minus the resource demand that falls in the second half. When using Z_{ijt}, $\frac{CP_1}{2}$ is used as the middle of project 1 (the red line). The resulting load for p_1 is -6. For p_2, $\frac{CP_2}{2}$ is used, resulting in a load = 0. The resulting NARLF is equal to $\frac{Load_1+Load_2}{J \cdot CP_{max}} = -0.375$. When using $\bar{Z}_{ijt}$, for both projects $\frac{CP_{max}}{2}$ is considered the point separating the first and second half (i.e. the green line). For p_1 this changes the load to -8, nothing changes for p_2 as $CP_2 = CP_{max}$. The adapted NARLF is equal to -0.5. When one would aggregate the resource profiles into one, making abstraction of the underlying projects (the right half of figure 4.3, the load is also equal to -8, giving a NARLF of -0.5. Thus, $\bar{Z}_{ijt}$ does not bias the NARLF calculations and should be used instead of Z_{ijt}.

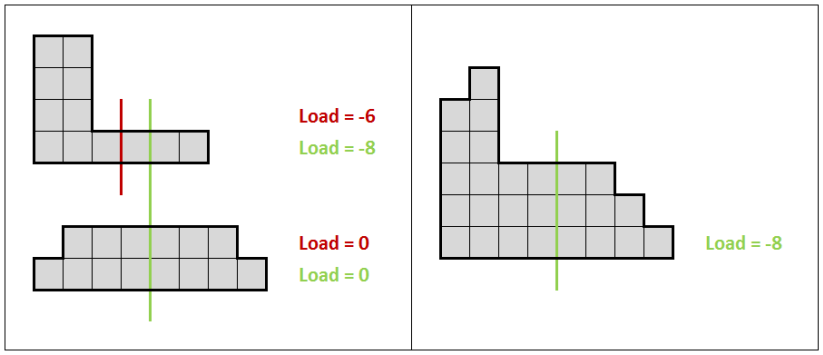


Figure 4.1: Differences in NARLF calculation

NARLF-values will tend to be more frequently negative as the resource load of shorter projects will fall mainly in the first half of the duration of the portfolio. If projects have different release dates, this could balance out as the resource demand of certain projects will be shifted to later periods. The data set used in this dissertation still uses the NARLF measure with Z_{ijt} instead of $\bar{Z}_{ijt}$. The insights regarding equation 4.14 were obtained when the dataset was already generated and multiple analyses were performed on it. It would be too time consuming to re-execute all analyses, so I continued with the imperfect data set. In future research, this flaw should be addressed.

Second, as every single-project network is incorporated multiple times in the multi-project data set, it is possible that this causes a bias. As shown in the figure below, 82.74% of the single-project networks is used between 7 and 10 times. However, some instances appear up to 14 times and others only 3 times. To improve the quality and the strong randomness, every single-project instance should be included only once in the data set.

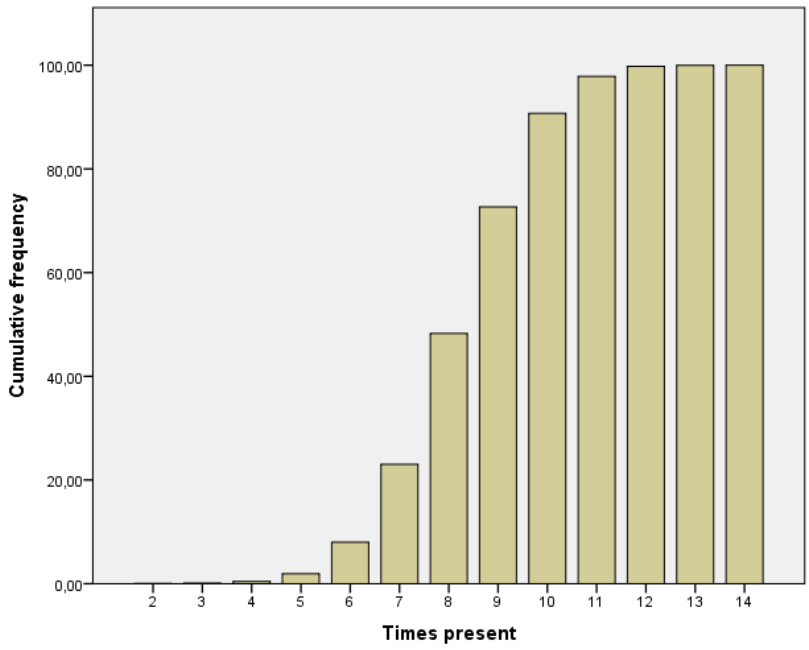


Figure 4.2: Number of appearances of single-project networks

Third, the resource tuning procedures are not flawless either, resulting in unequal (and unrealistic) resource distributions over activities. This results in a partly unrealistic data set (for both NARLF > 0 and low complexity networks). As a consequence, it is not possible to make unilateral conclusions regarding the effect of resource loading or complexity. I believe

that the cause for this is an erroneous understanding of NARLF. For more detail, I refer to Appendix A.

Last, Vanhoucke et al. (2008) create an updated version RanGen2 and show that it can generate networks in a wider range for multiple network topology indicators than its predecessor. In order to create a qualitative multi-project generator, the current generation procedure should be adapted such that it uses RanGen2 as network generator. However, as multiple additional network topology indicators are introduced, further research should first be addressed at devising extensions of these measures for the multi-project context.

Part III

Results and conclusion

Chapter 5

Benchmark methods

No standard benchmark data set for the RCMPSP exists. To evaluate the performance of the scheduling algorithms in this dissertation, different solution methods were implemented and applied on the data set proposed in chapter 4. This chapter will discuss two PR-based heuristics, a random benchmark and a genetic algorithm.

5.1 PR-based heuristics

A SSGS is implemented with two different priority rules: Shortest Activity of Shortest Project (SASP) and Maximum Total Work Content (MAXTWK). According to literature, these PR's provide good results (Browning and Yassine, 2010b). The former calculates the priority value of a_{ij} as follows:

$$\mathcal{P}_{ij} = d_{ij} + CP_j. \tag{5.1}$$

The activity with minimal $\mathcal{P}_{ij}$ is selected. Thus, the SSGS will give priority to short activities from short projects. The latter rule calculates the priority value using formula

$$\mathcal{P}_{ij} = TWK_j + \sum_{k=1}^{K} w_{ijk}. \tag{5.2}$$

For the definition of TWK_j, see equation 3.4. The sum of the resource demand (work content) of all activities of p_j that are already scheduled and the work content of a_{ij} is maximized. Resource intensive activities of resource intensive projects receive priority. Note that once some activities of p_j are scheduled, its remaining activities will have a higher probability of being selected, because the first term of 5.2 will be higher relative to other projects.

The PR-based heuristics works in two steps: (1) activity list creation and (2) scheduling. First, an activity list is constructed using one of the PR's mentioned above. Let $\mathcal{C}$ (Completed) be the set of activities that are already added to the list, $\mathcal{D}$ (Decision) the set that contains the activities of which all predecessors are in $\mathcal{C}$, and $\mathcal{I}$ (Ineligible) the set containing all activities for which the last condition does not hold. From $\mathcal{D}$, the activity with the best priority value $\mathcal{P}_{ij}$ is added to the list and transferred to $\mathcal{C}$. Now some activities from $\mathcal{I}$ may be transferred to $\mathcal{D}$ as the completed set is changed. This procedure is repeated until all activities are in $\mathcal{C}$. The resulting list contains all activities in a certain order. This order respects the precedence constraints. Second, this activity list is used by the SSGS, which starts with an empty partial schedule (i.e. no activities are added yet). Then, it selects the first activity in the list and searches the first time point in the partial schedule where no precedence or resource constraints of that activity are violated. It is added to the partial schedule at that time point and the resource availabilities are updated. Then, the SSGS selects the next activity on the list and repeats the process. This is iterated until all activities of the list are added to the partial schedule. Note that SASP and MAXTWK were implemented with a PSGS in Browning and Yassine (2010b). As an SSGS is used here, the results will not be exactly the same, but they should be similar to those in that paper.

5.2 Random benchmark

The random benchmark works similar to the method in 5.1. First an activity list is constructed, which is then scheduled by the SSGS. However, no priority rule is used to create the list. A list is randomly constructed, taking into account the precedence relations. The method creates and schedules 5000 random lists and returns the best performing list. It uses the strength of large numbers but incorporates no intelligence or heuristic knowledge.

5.3 Genetic algorithm

The last benchmark method is a genetic algorithm (GA), a meta heuristic that combines the strength of large numbers with a smart way of searching the solution space. The GA represents schedules by an activity list, which is also called a chromosome or an individual. It searches for well performing individuals and combines information from them to create better

ones. The general structure of the GA is elaborated in algorithm 8, each step is elaborated afterwards.

Algorithm 8: Structure genetic algorithm

```
1 Initialization
2 while generation < γ do
3     Evaluation
4     Elitism
5     Selection
6     Crossover
7     Mutation
```

The evolutionary strategy is inspired by Gonçalves et al. (2008), the concepts of the elitism and mutation step are used. However, as the authors address an extension of the RCMPSP with due dates and release dates, the solution representation has to be adapted. In this dissertation the activity list consists only of activities, no additional information is incorporated. Also, a different crossover function is implemented.

Initialization: An initial set (population) of 200 individuals is created. The activity lists are generated randomly, respecting precedence relations. After the initialization step, γ iterations (generations) of the algorithm are executed. The calculation of γ is explained at the end of this section.

Evaluation: In the evaluation step, every individual in the population is decoded into a schedule. This is done by executing the SSGS from section 5.1 with the activity list as input. The performance of the resulting schedules is evaluated with an objective function, which is APD in this case. The individuals in the population are sorted according to their performance.

Elistism: In the elitism step, the best $x\%$ chromosomes of a generation are copied to the population of the next generation. This strategy ensures that information about well performing individuals is preserved over generations.

Selection: In the crossover step, two parents will be combined to create offspring. The selection step selects the parents that will be used in the crossover step. The first parent is randomly selected from the $x\%$ best performing individuals. The second parent is randomly selected from the whole population. This approach ensures that information of strong individuals is present in the offspring, but at the same time it allows worse performing indi-

viduals to take part in the crossover process. This may postpone premature convergence (cfr. Mutation).

Crossover: In this step, individuals (children or offspring) for the next generation are created using individuals from the current generation (parents). Two parents are selected and the information residing in their respective activity lists is partly exchanged. This results in two new activity lists, i.e. the offspring. A two-point crossover is used as operator, its functionality is detailed in figure 5.1. The activity lists in step 1 represent the parents. In subsequent steps, two children are constructed. First, the operator randomly selects two cut off points in the parent activity lists (the bold lines in Step 1). In step 2, all activities before the first cut off point of one parent are placed in one child (grey fields). The information from parent 1 and 2 is copied to child 1 and 2 respectively. Then, all activities that are present in one but not both children are added to the one where it is not present yet (the red fields). In step 3, the activities between the cut off points of parent 1 not yet present in child 2 are added to that child (grey fields). The same is done for parent 2 and child 1. Once again, activities present in only one child are added to the back of the list of the other child (red fields). In step 4, the activities after the last cut off point of parent 1 not yet present in child 1 are added to it, the same happens for parent 2 and child 2. $y\%$ of the new population will be constructed by applying the crossover operator to parents.

Mutation: By selecting well performing individuals and combining these, one risks that after a few generations all offspring will resemble each other (i.e. premature convergence). In order to avoid this, mutation is introduced. This operator adds new information to a population, extending the search space in which the GA operates. In this dissertation, it is implemented as follows: the last $z\%$ $(= 100 - x - y)$ of a population is filled with newly generated random activity lists.

The percentages (x, y, z) are set to $(20, 50, 30)$. In the initialization step 200 schedules are generated. In every subsequent iteration, $(y + z) \cdot 200 = 160$ new schedules are generated. With a limit of 5000 schedules generated, this results in $\gamma = \frac{4800}{160} = 30$ generations to be executed.

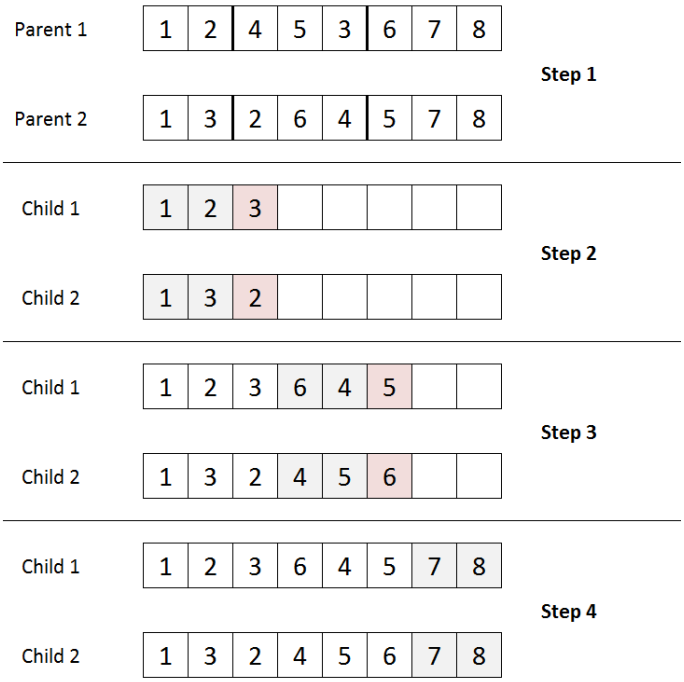


Figure 5.1: Operation of the two-point crossover

Chapter 6

Computational results

This chapter will discuss the computational results of both the MPSGS and the schedule game mechanic. The details on the data set on which the analyses are performed is discussed in section 4.2. The full set contains 14700 multi-project instances. Preliminary tests (e.g. parameter testing) are executed on a subset of 588 instances for two reasons. It reduces the computational intensity of the tests and avoids overfitting of the algorithms on the data set. The subset consists of one multi-project for every combination of the parameter levels OS, NARLF, MAUF and σ^2_{MAUF}. The full set contains 25 instances per parameter combination. First the MPSGS will be discussed. Second, the configuration of the schedule game is analyzed. Finally, the performance of the MPSGS and the schedule game are compared with the benchmark methods from chapter 5.

6.1 MPSGS

The activity priority rules (APR) and project priority rules (PPR) from section 3.1 are tested on the set of 588 instances. The graph below reports the mean APD of the decoupled priority rules over the instances of small set. The choice of a PPR has the biggest impact on the performance of the MPSGS. SP gives the best results, followed by MINCPL. In most cases, there is no interaction effect between the APR and PPR. For instance, indifferent from the choice of an APR, SP outperforms MINCPL. For any PPR, the same pattern occurs for the APR's. MINSLK performs best, followed by MAXWK and LA. Obviously, the other three are their opposites and perform worse. Although the pattern is similar, the magnitude of the

variations may differ (e.g. the performance spread of MINTWK is bigger than that of LP). The combination MAXWK-MAXCPL is an exception in the observations, it shows worse results than expected. According to the figure, BA and MAXTWK are equivalent. This makes sense as MAXTWK selects the agent with the highest work content of activities that are already scheduled and BA selects the agent with the most activities that are already scheduled. The only difference between them is the resource weight of the activities, but apparently this has no big impact.

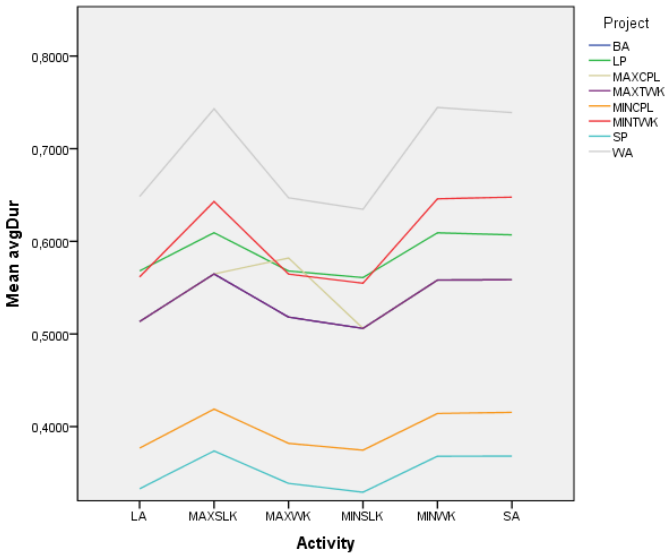


Figure 6.1: Impact of APR's and PPR's on performance MPSGS

LA-SP, MAXWK-SP and MINSLK-SP were studied in more detail. As the rules are tested on the same problems, the results are paired. The difference between any two rules is not normally distributed, which means that the paired samples t-test cannot be used. When the differences are approximately symmetrically distributed (MINSLK-SP versus MAXWK-SP and LA-SP versus MAXWK-SP), the Wilcoxon Signed Rank Test is used, otherwise (MINSLK-SP versus LA-SP) the Sign Test is used. As shown in the table below, MINSLK-SP shows the best performance and its difference from MAXWK-SP and LA-SP is significant on the 5% confidence level.

For every problem instance, the solutions of the three rules were ranked from best to worst performer. MINSLK-SP, LA-SP and MAXWK-SP ranked first for 63.1%, 45.9% and 37.9% of the problems respectively. These percentages do not add up to 100% as sometimes two or

Table 6.1: Comparison of performance

Selection rule	APD	Difference in APD		p-value (Wilcoxon)	p-value (Sign)
MINSLK-SP	0.3292	MINSLK-SP	LA-SP	<0.001	<**0.001**
LA-SP	0.3329	MINSLK-SP	MAXWK-SP	<**0.001**	<0.001
MAXWK-SP	0.3387	MAXWK-SP	MINSLK-SP	<**0.001**	<0.001

For every difference, the results of both the Wilcoxon and the Sign test are shown to indicate that they give the same outcome. The statistically correct p-value is reported in bold.

three rules are tied for the first place. Based on these results, MINSLK-SP will be used as priority rule in the schedule game.

To conclude this section, a comparison is made between the SSGS using SASP and MAXTWK and the MPSGS using SA-SP and MAXWK-MAXTWK. Decoupling the selection mechanism has a benificial impact in the case of SASP, but has an averse effect in the case of MAXTWK. This is mainly due to the fact that the PPR SP shows good results and MAXTWK does not (cfr. figure 6.1). The decoupled approach is not a guarantee for better performance, one has to choose the PPR and APR with care. Further research should be executed to discover additional good PPR's and APR's.

Table 6.2: Comparison between coupled and decoupled selection

Priority rule	APD
SASP	0.3850
SA-SP	0.3681
MAXTWK	0.4166
MAXWK-MAXTWK	0.5182

6.2 Schedule game

In this section, the impact of the resource allocation and random sampling will be evaluated. In the next section, the performance of the extensions to the basic game are discussed. Unless stated otherwise, the tests were executed on the set of 588 instances, .

6.2.1 Resource allocation

The three mechanisms were discussed in subsection 3.2.1, they will be referred to with their respective numbers (approach (1), (2) and (3)) as stated there. Figure 6.2 shows the results for the three mechanisms for different values of f. It is clear that the mechanism is particularly

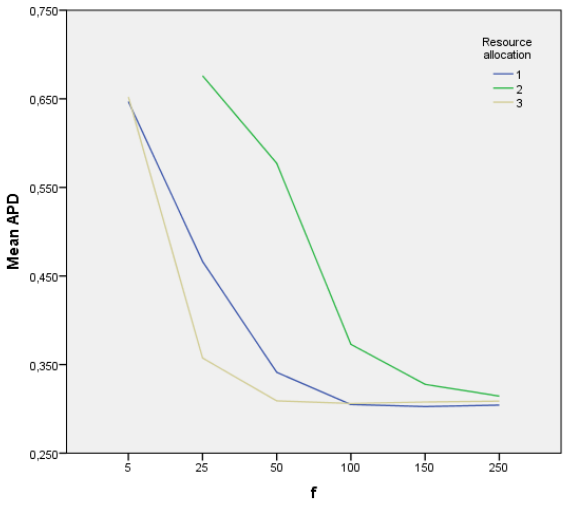


Figure 6.2: Comparison of resource allocation mechanisms

sensitive to a small f (i.e. allocating few resources per round). This can be explained as follows: the well performing project selection mechanism of the MPSGS will be disturbed too much when using small f. An agent has to wait until he has received enough resources before he can send a schedule order. When few resources are allocated per round, few schedule orders will be sent in parallel as agents will be waiting for resources during most of the rounds. As a consequence, the CA will not have to prioritize between projects, this prioritization is mostly done implicitly by the allocation of resources. When f is increased, the detrimental impact of resource allocation reduces. At higher levels of f, agents will be able send orders more frequently and will have fewer rounds during which they have to wait for resources. This

results in more schedule orders arriving at CA per round and the project selection rule being applied more frequently. At a certain level, the incremental improvement of allocating more resources approaches zero. One can assume that the disturbing impact of resource allocation has almost fully disappeared at that point.

Table 6.3: Optimal resource allocation parameters

Resource allocation	f^*	APD
(1)	150	0.3026
(2)	250	0.3144
(3)	100	0.3061

Each mechanism has a different optimal f^*, at the optimal level (1) performs the best, although the difference with (3) is not big (yet significant according to the Wilcoxon Signed Rank Test at the 5% confidence level). (2) lies further away, this can be explained as follows: the mechanism does not guarantee a minimal amount of resources received per round. The higher f, the higher the expected amount of received resources, but it is still probable that agents receive few or even 0 resources during certain rounds. Apparently this disturbs the prioritization effect more than (1) or (3). The differences between the allocation mechanisms become less pronounced at high f and their performance converges to approximately the same level. Together with the insights from the previous paragraph this indicates that the resource allocation step is not a main contributor to the performance of the mechanism. One has to ensure that agents receive enough resources per round to avoid disturbing the activity and project selection process. The specific manner in which they are assigned is less relevant.

6.2.2 Random sampling

Next, the impact of random sampling (see subsection 3.2.2) is evaluated. The parameter for selection pressure η is varied over the values [1, 2, 3, 4, 5]. The table below shows the mean APD over all problem instances.

The table shows that random sampling can slightly improve the performance of the schedule game. As RS with $\eta = 4$ shows the best results, this will be analyzed further. For the remainder of this subsection, when referring to RS, $\eta = 4$ is implicitly assumed. The difference

Table 6.4: Evaluation Random Sampling

	No RS	RS				
η	-	1	2	3	4	5
APD	0.3027	0.3031	0.3019	0.3012	0.3009	0.3016

between No RS and RS is analyzed with the Wilcoxon Signed Rank test (the differences are not normally distributed but are symmetrical). The resulting test statistic is -3.255 (p-value = 0.001), which leads to the conclusion that the difference between the two is significant on the 5% confidence level, albeit that the magnitude of the difference is hardly relevant. Next to this RS performs at least as good as No RS in 73.98% of the cases.

Table 6.5: RS: Percentage ranked best

Best performer	Count	Relative
No RS	153	26.02%
Tied	189	32.14%
RS	246	41.84%
Total	**588**	**100%**

Based on the previous two analyses, it seems favorable to use random sampling, even though the incremental improvement is not that large. As every game is executed multiple times, the best solution can be found in any of these iterations. The resulting pattern is shown in the figure below. The horizontal axis shows the iteration in which the best solution is found for a certain problem. The vertical axis represents the cumulative percentage of all problems that reached their best solution after that iteration.

When the game is executed without RS, for almost 50% of the problems the best solution is already found after the first iteration; with RS, this is only 27%. This makes sense as the latter enables agents to select activities that have a worse priority value, which may result in worse schedules. When enough iterations are executed the solution quality will improve. After 25 iterations, RS outperforms its deterministic variant as it has enough opportunities to explore its search space. However, if one would allow fewer iterations (e.g. 10), it is probable that RS would perform worse. For only 64% of all problems it would have found its current

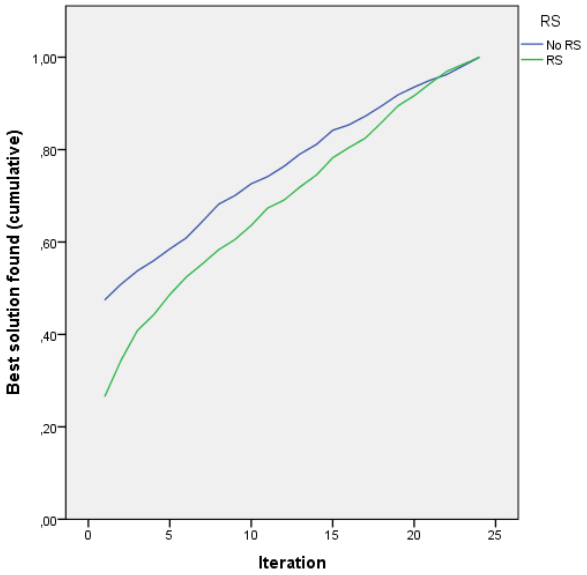


Figure 6.3: Convergence pattern No RS versus RS

best solution, where not using RS would have already found the best solution in 73% of the cases. This hypothesis was tested on the complete data set of 14700 instances: the results after 10 iterations without RS and with RS ($\eta = 4$) are reported in the table below. It appears to be not true, even with fewer iterations, random sampling is still better. The magnitude of the difference is approximately the same at 10 or 25 iterations.

Table 6.6: Impact RS after 10 iterations

Random sampling	APD
No	0.2901
Yes	0.2886

A last study was executed to evaluate the impact of additional iterations on the performance of the game mechanism. The results for 10 and 25 iterations are already available, two additional levels were tested: 100 schedules and 200 schedules (the latter gives the algorithm roughly the same computation time as the genetic algorithm, cfr. section 6.4). The results are reported in table 6.7, these tests were executed on the data set of 14700 instances. Granting the algorithm more computation time increases its performance. When the mechanism is given the same computation time as the genetic algorithm (200 schedules), it approaches the

GA closely: its average APD is 0.0055 higher. The main reason for this is that it is not a metaheuristic and does not apply any diversification. Increasing the computation time will increase the number of evaluated solutions in the same neighborhood. As a consequence it may find incremental improvements to its incumbent solution, but it will not explore parts of the search space that are further from the current solution.

Table 6.7: Impact of number of iterations

Iterations	10	25	100	200
APD	0.2886	0.2815	0.2732	0.2702

6.3 Extensions

All extensions were tested on the smaller data set of 588 instances. Each extension was activated separately first, in order to establish the main effects. The extensions have different parameters that can be changed. For every parameter, multiple levels were evaluated, but these analyses will not be discussed here. The reason for this is that none of the parameters has a substantial impact on the performance. The table below shows the range over which the parameters were tested and the level giving the best results.

Table 6.8: Parameter setting extensions

Extension	Parameters	Tested levels	Best level
Tightness	τ	[0.5, 0.7,..., 1.5]	1.1
	κ	[0.5, 1.0,..., 3.0]	3.0
Negotiation	ν	[5%, 10%,..., 25%]	10%
LBQU	S	[2, 4,..., 10]	8
PBQU	ω	[2, 4,..., 10]	2

The average APD over all problems of the extensions is compared with that of the base game in table 6.9.

Tightness and Load Based Quarry Updating (LBQU) outperform the base game although the differences are small. As the differences (Base game - Tightness and Base game - LBQU) are not normally distributed, nor symmetrical, only a Sign Test can be used to evaluate the

Table 6.9: APD of extensions

Extension	APD
Base game	0.3009
Tightness	0.3001
Negotiation	0.3018
LBQU	0.3002
PBQU	0.3017

significance of the differences. Neither difference proved to be significant on the 5% confidence level. A possible explanation for the small size of the effect is that the performance of the base game is already quite good in comparison with other benchmarks (cfr. infra). This makes it harder for the extensions to further improve the results. In previous tests with other priority rules (which are not relevant to discuss), some of the extensions had a larger improvement potential. However, in these cases the APD of the base game was almost twice as high as now, giving the extensions more room to enhance the solutions. When splitting the results over the parameters MAUF, NARLF and OS, there are small variations over the extensions, but the impact is never noteworthy. As the main effects are not significant, it is improbable that interaction effects would be, so I decided not to test all possible combinations of extensions. It would lead this dissertation too far from its main focus.

6.4 Comparison with benchmark methods

This section compares the MPSGS and game mechanic from the previous sections with the benchmark methods proposed in chapter 5. Before going further in detail about the results, it is important to classify the methods. Both SSGS's and the MPSGS are local search method as they evaluate a single point in the search space. The game mechanic is a multi-pass heuristic (multiple iterations are executed), but should still be categorized as a local search method. It evaluates multiple points in the search space, but its search is clearly focused in a certain direction. The method does not include a diversification part. The random benchmark is a passive heuristic as it evaluates many points in the search space but does not use any information to guide its search, it exists only of a diversification part. On the contrary, the

genetic algorithm is a metaheuristic that includes both diversification and intensification. It evaluates a broad spectrum of the search space and its search is guided by promising areas. All methods were executed on the full data set of 14700 instances. The resulting APD's and standard deviations are reported in the table below.

Table 6.10: Comparison with benchmark methods

Method	APD	σ_{APD}
Genetic algorithm	0.2647	0.3689
Game mechanic	0.2815	0.3521
MPSGS	0.3098	0.3657
Random benchmark	0.3414	0.4883
SSGS (SASP)	0.3625	0.4242
SSGS (MAXTWK)	0.3926	0.4215

The genetic algorithm performs best, followed by the game mechanic and the MPSGS. The difference between any two methods is significant on the 5% confidence level according to the Sign Test. The game mechanic has a smaller standard deviation, i.e. its solutions lie in a smaller interval around the mean than for other methods. Both the MPSGS and the game mechanic outperform SASP and MAXTWK, two local search methods that are known in literature to be strong performers (cfr. supra). Next to the performance, the computational efficiency of the methods is important too. Table 6.11 compares the computational efficiency of the three best algorithms. The tests were executed on a HP Pavilion g7, with an Intel(R) Core i7 processor (clock rate 2.10 GHz). It shows how much schedules every approach creates, how long it takes to find a solution for one problem instance and the incremental improvement in APD it reaches in comparison to the MPSGS. The genetic algorithm constructs 5000 times as much schedules to reach an incremental improvement of 0.0451 in APD compared to the MPSGS. The former is also 28 times as slow as the latter. The game mechanic finds an improvement of 0.0283 but only creates 25 times as much schedules and is only 3 times as slow. While the genetic algorithm is better than the game mechanic, it needs a lot more computational effort. Along the two dimensions of performance and computational efficiency, none of these methods is dominated by an other one. When the game receives the same computation time as the GA, they are almost equivalent in performance (cfr. supra). The

MPSGS is slower that the SSGS, which needs on average 0.0122 seconds per problem instance. However, this difference is mainly attributable to programming efficiency. When the code of the MPSGS would be cleaned up (removing redundancies etc.), its execution time should approach that of the SSGS.

Table 6.11: Computational efficiency

Method	Schedules	Seconds	Incremental improvement
Genetic algorithm	5000	0.5714	0.0451 (14.56%)
Game mechanic	25	0.0612	0.0283 (9.13%)
MPSGS	1	0.0204	-

The numbers between parentheses in the last column are the relative improvement.

All results reported above are the averages over 14700 problem instances. The data set was generated with varying values for the parameters OS, NARLF, MAUF and σ^2_{MAUF}. The performance of the best four methods is plotted over these parameters in figure 6.4. The SSGS based PR's are not reviewed as they are dominated by the MPSGS for all parameter values. The trends over the parameters are comparable to those found in Browning and Yassine (2010b), with one difference. The authors found that problems with a higher network complexity result in lower APD inflation, which is opposite to the results presented here (cfr. infra). The most probable cause for this is that their measure of complexity C_j is no good predictor of complexity (cfr. section 4.2).

When reviewing network complexity, it is clear that more projects with a high or medium OS result in a harder problem and a higher inflation of APD. It appears that the addition of even only one network with high complexity has a big impact on the complexity of the multi-project instance. The average OS ($\bar{OS}$) can predict the network complexity of a multi-project to a certain extent. However, it is not a sufficient measure, as for problems with $\bar{OS} = 0.5$ (i.e. HHMLL, HMMML, MMMMM), there are still big differences in performance it cannot explain. A vector summing up all levels of complexity is not a good measure either. It may be a better predictor for network complexity, but it lacks potential for generalization or aggregation and is hard to interpret, certainly for larger multi-project instances. Further research should be addressed at devising a better network complexity measure for multi-

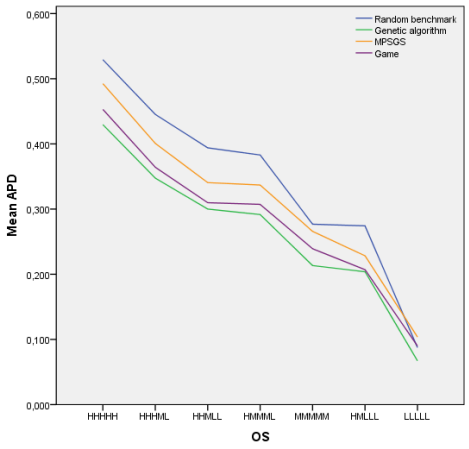
projects. It should allow aggregation while not making full abstraction of the underlying networks. The genetic algorithm performs the best for any complexity vector, but the game comes very close for HMLLL. For the least complex vector (LLLLL), the random benchmark reaches the same performance as the game.

The next parameter is NARLF. Multi-projects that are more front loaded show an increased APD. This trend was discovered in Browning and Yassine (2010b), the authors give the following explanation: "...the former [i.e. negative NARLF values] implies front-loading of the resource constraints, which has implications for all downstream activities in the network...". For front-loaded problems (NARLF < 0), the performance of the game and the genetic algorithm lie close together. When NARLF increases, the difference between the two methods becomes larger. When NARLF > 0, the random benchmark approaches the game mechanic.

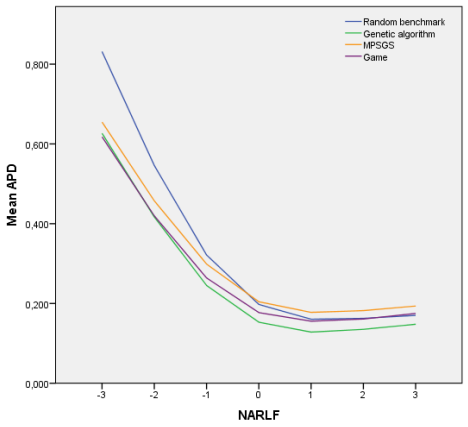
A higher MAUF results in a larger APD. This makes sense as resources become more constrained, causing more activities to be delayed beyond their earliest start time. The pattern observed in figures 6.4a and 6.4b reoccurs: for the hardest problems (high MAUF), the performance of the game approaches that of the genetic algorithm. For the easiest problems (low MAUF), the random benchmark lies close to the game.

The analysis of the last parameter, σ^2_{MAUF} does not deliver additional insights. A higher σ^2_{MAUF} results in less constrained resources and thus fewer delays. One could argue that the combination of MAUF and σ^2_{MAUF} is not an appropriate indicator for resource contention of a multi-project. As σ^2_{MAUF} is the variance from the maximum utilization, a higher value always results in a lower average utilization (ceteris paribus). Hence, σ^2_{MAUF} is actually a proxy for average utilization, which may confound conclusions based on it. An alternative could be to define MAUF as the *average* utilization and σ^2_{MAUF} as the variance from the mean. With this combination, a higher variance does not result in a lower average utilization. Another option is to evaluate both the average and the maximum utilization. However, as the variance includes information about the variation of utilization for all resources, it would receive my preference. Another objection about the current measures is that they look at the utilization per *resource type*, but not at the utilization per *project*. Taking this into account may also lead to better measures for resource utilization in a multi-project context. Further research could identify new measures for resource contention.

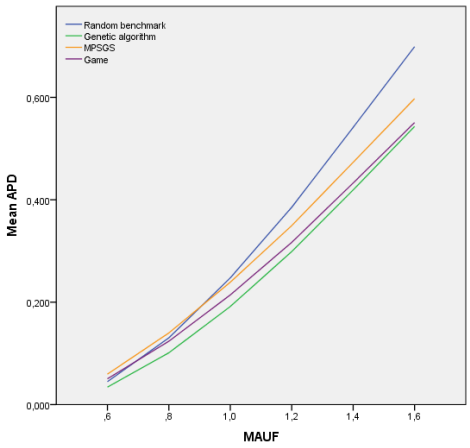
A critical remark from section 4.3 is repeated here. Regarding the impact of NARLF or the



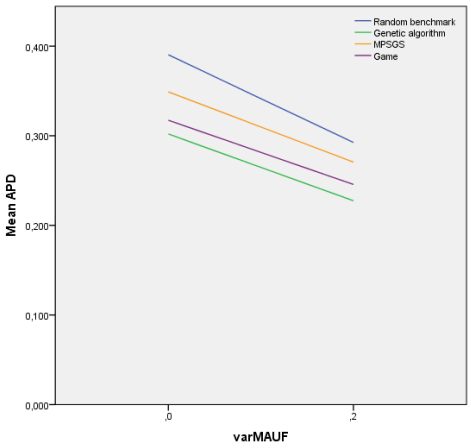
(a) Impact of OS



(b) Impact of NARLF



(c) Impact of MAUF



(d) Impact of σ^2_{MAUF}

Figure 6.4: The impact of problem parameters on the performance of scheduling algorithms

OS vector on the APD, the results in this section should be interpreted with care. Instances from this data set with high NARLF values or multiple low complexity networks contain lots of activities with a very low resource demand. This significantly reduces the hardness of the scheduling problem, as these activities will be delayed less frequently due to their low resource demand. However, this property is not causally related to a low complexity or a high NARLF. Instead, it is caused by the generation procedure that encounters difficulties to generate instances with a high NARLF because this parameter is not symmetrical. The effect of the parameters is confounded with the intricacies of the generation procedure. As the tuning procedures are based upon Browning and Yassine (2010a), I argue that the generated

instances in that paper and its conclusions are subject to the same effect. For more detail on this matter, I refer to the appendix A. This again indicates that a thorough evaluation of the existing multi-project parameters is necessary.

Chapter 7

Conclusion

Most previous research regarding PR-based heuristics for the RCMPSP uses the SSGS or PSGS for the RCPSP combined with PR's that sometimes include some project specific information. In this dissertation, an extension to the SSGS (called the MPSGS) was designed specifically for the RCMPSP. It involves two decision points: one for the project and one for the activity. This MPSGS combined with the decoupled priority rule MINSLK-SP outperformed the SSGS with some of the best performing PR's currently known, although its implementation or computational efficiency are not that different from the SSGS. As SGS's are often used as building blocks for more complex solution methods, the MPSGS is a valuable addition to the scheduling schemes for research in the multi-project context. It may improve the effectiveness of algorithms that use it as base heuristic. Further research could be addressed towards developing a multi-project variant of the PSGS and towards the discovery of new project and activity priority rules.

Agent based systems in multi-project scheduling literature often work in a deterministic way. The multi-agent system introduced here models game mechanics into the negotiation process, involving some degree of randomness. One could classify the resulting algorithm as a multi-pass extension of the MPSGS. The game mechanism approaches the performance of a genetic algorithm while being approximately 9 times as efficient in computation time. When both methods get the same computation time, the APD of their solutions is almost equivalent. Augmenting the decision making process of agents by adding extensions has not shown a significant improvement. However, the basic game on its own already has a strong performance. Further research could investigate the opportunity of applying game theory to

further improve the decision making of agents.

The absence of a standard set of problem instances for the RCMPSP is a flaw in the current multi-project literature. It makes it nearly impossible to compare the performance of different solution methods as the underlying test sets are different. One multi-project generator exists, but it has several shortcomings. To cover these weaknesses, a new multi-project generator was introduced in this dissertation, building on existing concepts and generators. It was used to generate a problem set with varying values over four problem parameters. The current generation procedure and data set are not perfect yet. The network generation shows room for improvement and the quality of the resource tuning algorithms should be investigated further. However, future research should first be addressed at defining meaningful, unbiased problem parameters for multi-projects. Next to NARLF and MAUF, it may be interesting to investigate other possible resource related parameters that take into account the relations between projects. Current network related parameters for multi-projects do not go beyond the aggregation of the underlying projects. The impact of interaction between their network complexities on the portfolio's complexity may provide valuable insights.

Part IV

Annexes

Appendix A

Quality of resource tuning procedures

The resource tuning procedures from Browning and Yassine (2010a) were implemented in the multi-project generator of this dissertation. The authors do not provide any analysis about the quality of these algorithms. This appendix will elaborate on the results of the tuneNARLF-procedure (algorithm 5 in chapter 4), as this has the highest impact on the resource demand of the activities in a problem instance. It can increase or decrease the r_{ijk} of any non-dummy activity a_{ijk} in order to meet its NARLF target. However, if an activity has only demand for one resource ($K_{ij} = 1$) and the demand for this resource is equal to 1, it cannot be decreased further. This avoids creating activities with no work content. These activities will be called Lower Bound Activities (LBA), as their resource demand has reached a lower bound. For every one of the 14700 multi-projects, the number of LBA's were counted. Figure A.1 shows the average count of LBA's per instance (called oneCount) over the different project parameters. The number of LBA's increases for higher NARLF-values and for multi-projects with more projects with a low complexity. The most extreme cases on average have up to 50 LBA's, which is 40% of the total activities of a multi-project (one instance consists of 125 non-dummy activities). These instances can hardly be called realistic and should not be part of a standard problem set.

The explanation for this pattern is as follows. NARLF is calculated based on CP_{max}, most of the projects in the multi-project will have a $CP_j < CP_{max}$. As all projects start at the same time, most of them will be finished when the time approaches CP_{max}. This results in

multi-projects being more front loaded, i.e. relatively more activities are active in the first half of CP_{max} than in the second half (according to their critical path). Due to this property, it is more probable that a multi-project (before any resource tuning) has a negative NARLF. If all projects would have the same CP duration, the NARLF value would be closer to 0. In general, the NARLF will also be more negative for projects with a lower complexity. As more activities can be executed in parallel, their earliest start times will be earlier and the resource unconstrained critical path schedule will be more front loaded than for projects with a high complexity. This leads to the conclusion that multi-project instances in general and those with low complexity in particular tend to have negative NARLF-values.

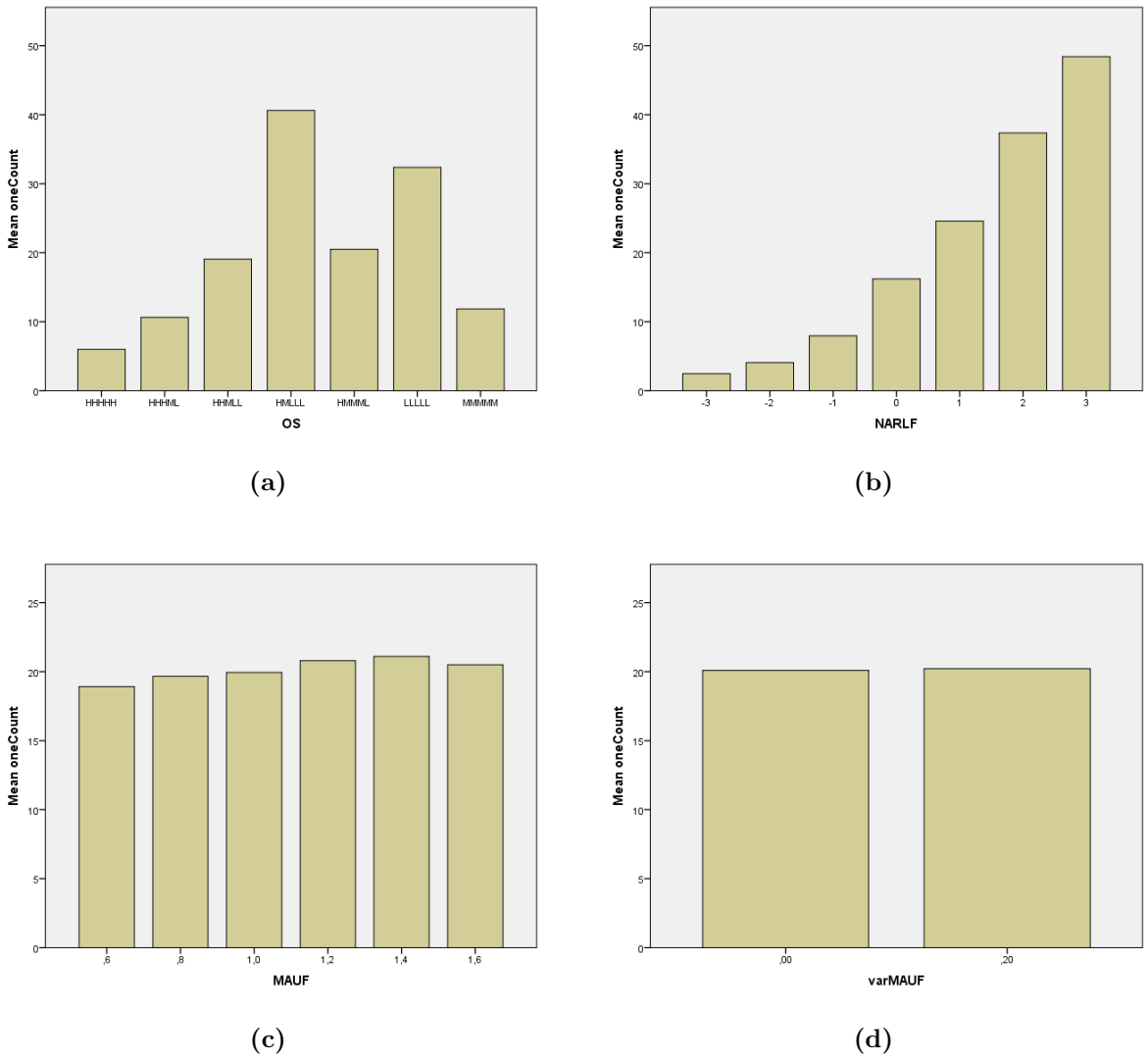
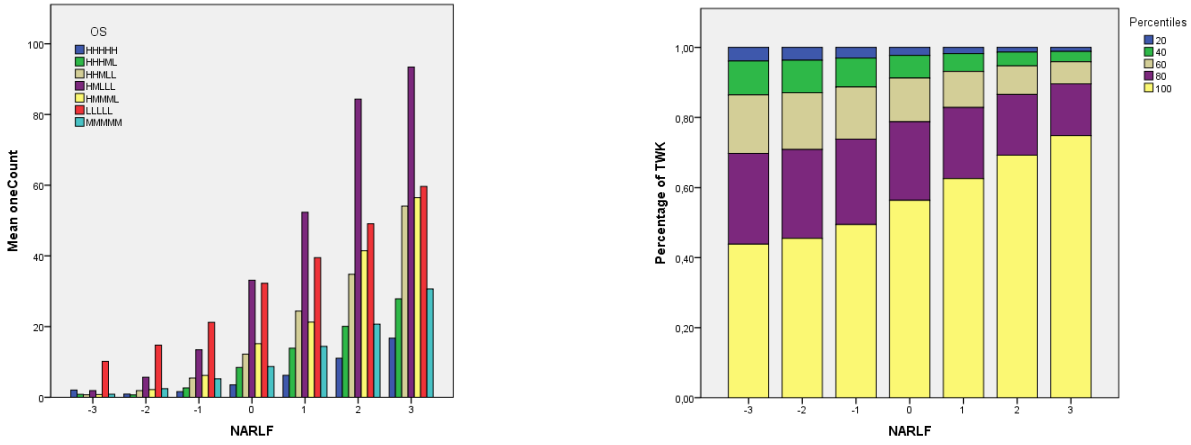


Figure A.1: The count of LBA's per project parameter

When a generator has to create instances with highly positive NARLF, the resource demand of a lot of activities in the front half will have to be reduced to their lower bound. This is exactly the pattern that occurs in figure A.1b. Figure A.1a shows that the count of LBA's is very high for multi-projects with lower complexities, indicating that it becomes harder to generate instances successfully. However, this second observation is mainly true when both OS is low and NARLF is high. This can be seen in figure A.2a. In general a multi-project of a lower complexity has more LBA's, but the effect becomes much stronger for NARLF > 0. There is no big variance in the count of LBA's over varying MAUF or σ^2_{MAUF}, indicating that the generator encounters fewer difficulties in creating instances over these values.



(a) **(b)**

Figure A.2: Detailed analysis generated problem set

Next to creating a lot of LBA's, the resource demand of activities in the second half will be inflated out of proportion. These combined effects result in unequally distributed demand profiles for activities. This is shown in figure A.2b: all activities per multi-project were ranked in ascending order of resource demand and grouped in percentiles of 20%. The vertical axis shows the percentage of the total work content of a multi-project that is occupied by a certain percentile. For instance, at NARLF $= -3$, the resource demand of the 20% most resource heavy activities (i.e. the 100-percentile) is around 40% of the TWK or the multi-project. As NARLF increases, it is clear that the TWK is more and more attributable to the resource intensive activities. At NARLF $= 3$, 60% of the activities only account for around 10% of the total work content.

It is clear that the generation procedure incurs major flaws for high NARLF values, resulting in unrealistic problem instances. The question remains whether the cause for this is the procedure itself or the underlying parameter. I would argue the problem lies with the parameter NARLF. In literature it is treated as a symmetrical parameter that can assume negative and positive values without difficulty. Both Browning and Yassine (2010b) and Kurtulus and Davis (1982) generate projects with (N)ARLF values in the range [-3,3]. However, I have argued that it is more probable that multi-project instances have a negative than a positive NARLF. The problems that tuneNARLF encounters for highly positive NARLF values are a symptom of this property. Conclusions based on this parameter are thus also biased. Section 6.4 states that problem instances with a higher NARLF or low complexity have a lower increase in APD. It is now clear that this is due to the fact that a high amount of activities with a very low resource demand are present in these problem instances, making the scheduling process easier. The real effects of a back-loaded or a low complexity multi-project are confounded by this effect and no conclusions can be made about it. Two options exist: (1) NARLF remains used in research, but the effect of its asymmetrical distribution is taken into account; or (2) a new resource loading parameter is designed that is more symmetrical.

Bibliography

Agnetis, A., Briand, C., Billaut, J.-C., and Ščha, P. (2015). Nash equilibria for the multi-agent project scheduling problem with controllable processing times. *Journal of Scheduling*, 18(1):15–27.

Bock, D. B. and Patterson, J. H. (1990). A comparison of due date setting, resource assignment, and job preemption heuristics for the multiproject scheduling problem. *Decision Sciences*, 21(2):387–402.

Browning, T. R. and Yassine, A. A. (2010a). A random generator of resource-constrained multi-project network problems. *Journal of scheduling*, 13(2):143–161.

Browning, T. R. and Yassine, A. A. (2010b). Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 126(2):212–228.

Confessore, G., Giordani, S., and Rismondo, S. (2007). A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135.

Demeulemeester, E., Dodin, B., and Herroelen, W. (1993). A random activity network generator. *Operations Research*, 41(5):972–980.

Demeulemeester, E., Vanhoucke, M., and Herroelen, W. (2003). Rangen: A random network generator for activity-on-the-node networks. *Journal of scheduling*, 6(1):17–38.

Dumond, J. (1992). In a multi-resource environment, how much is enough? *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 30(2):395–410.

Dumond, J. and Mabert, V. A. (1988). Evaluating project scheduling and due date assignment procedures: an experimental analysis. *Management Science*, 34(1):101–118.

Elmaghraby, S. E. (1977). *Activity networks: Project planning and control by network models*. Wiley New York.

Gonçalves, J. F., Mendes, J. J., and Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190.

Homberger, J. (2007). A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. *International Transactions in Operational Research*, 14(6):565–589.

Homberger, J. and Fink, A. (2017). Generic negotiation mechanisms with side payments–design, analysis and application for decentralized resource-constrained multi-project scheduling problems. *European Journal of Operational Research*.

Knotts, G., Dror, M., and Hartman, B. C. (2000). Agent-based project scheduling. *Iie Transactions*, 32(5):387–401.

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14(3):179–192.

Kolisch, R. (1996b). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333.

Kolisch, R. and Meyer, K. (2006). Selection and scheduling of pharmaceutical research projects. In *Perspectives in Modern Project Scheduling*, pages 321–344. Springer.

Kolisch, R., Schwindt, C., and Sprecher, A. (1998). Benchmark instances for project scheduling problems, 197–212.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703.

Kumanan, S., Jose, G. J., and Raja, K. (2006). Multi-project scheduling using an heuristic and a genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 31(3-4):360–366.

Kurtulus, I. (1985). Multiproject scheduling: Analysis of scheduling strategies under unequal delay penalties. *Journal of Operations Management*, 5(3):291–307.

Kurtulus, I. and Davis, E. (1982). Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172.

Lawrence, S. R. and Morton, T. E. (1993). Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. *European Journal of Operational Research*, 64(2):168–187.

Lee, Y.-H., Kumara, S. R., and Chatterjee, K. (2003). Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism. *Journal of Intelligent Manufacturing*, 14(5):471–484.

Lova, A., Maroto, C., and Tormos, P. (2000). A multicriteria heuristic method to improve resource allocation in multiproject scheduling. *European Journal of Operational Research*, 127(2):408–424.

Lova, A. and Tormos, P. (2001). Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. *Annals of Operations Research*, 102(1-4):263–286.

Mastor, A. A. (1970). An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science*, 16(11):728–746.

Shtub, A., LeBlanc, L. J., and Cai, Z. (1996). Scheduling programs with repetitive projects: a comparison of a simulated annealing, a genetic and a pair-wise swap algorithm. *European Journal of Operational Research*, 88(1):124–138.

Tsubakitani, S. and Deckro, R. F. (1990). A heuristic for multi-project scheduling with limited resources in the housing industry. *European Journal of Operational Research*, 49(1):80–91.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., and Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524.