Spatial Resource Constrained Project Scheduling

The allocation and sequencing of activity groups to spatial resources

Master Thesis Applied Mathematics

A.L. Kok

June 21, 2006

Faculty of EEMCS
Chair of DMMP

University of Twente
The Netherlands

Spatial Resource Constrained Project Scheduling

The allocation and sequencing of activity groups to spatial resources

Master Thesis Applied Mathematics

A.L. Kok
s0000949
June 21, 2006

Supervisors:
Ir. J.J. Paulus
Dr. J.L. Hurink

Committee:
Dr. J.L. Hurink
Ir. J.J. Paulus
Dr. Ir. J.M.J. Schutten

University of Twente
The Netherlands

Preface

This thesis is the result of the research I have been doing in the last seven months. This research and thesis form the final phase of my study Applied Mathematics at the University of Twente. I really enjoyed the work that I have been doing in these last seven months, and this is one of the reasons that I decided to continue working at the university of Twente as a PHD-student. I would have never accomplished this thesis without the support of several people. I am very grateful for them and I want to thank some of them in particular.

First of all, I would like to thank my supervisors Johann Hurink and Jacob-Jan Paulus. Their knowledge and experience have brought and kept me on track during my research and their critical notes considering my thesis have improved the quality of it. The discussions with them about the progress of my research have been of great value.

Furthermore, I would like to thank the people working at the chair of DMMP. Thanks to them, I always felt a nice and comfortable atmosphere during my work which had a positive influence on the quality of it. Outside the university, my friends and family have always been supporting me with my research, for which I am very grateful.

I would like to say a special word of thanks to my fiancee Simona. She always supported me and trusted in me, both in easy times and in hard times. Without her, I would have never accomplished this thesis.

Abstract

In this paper, we present a new topic in connection with project scheduling. Since a limited work space can form a restriction to project scheduling problems, an efficient usage of these work spaces (or so-called spatial resources) is required to produce good project schedules. The requirement for the resource units of these spatial resources is in an adjacent manner (we cannot place a product under construction in separate parts of the work space). A spatial resource unit is not required by a single activity, but by a so-called activity group.

The requirement for adjacent resource units not only appears in project scheduling, but also in several other applications. For example, the problem of scheduling check-in desks at an airport. The reserved check-in desks for each single flight have to be adjacent. Another example appears in the berthing problem, where quay space has to be assigned to ships of different length, which berth during specified time periods. Within warehouse management and the reservation of hotel rooms, resource adjacency is not required but desirable.

The aim of this paper is to derive a solution method which provides an efficient spatial resource usage within project scheduling. This solution method allocates the activity groups to the spatial resources and schedules the activity groups, such that good solution possibilities for the overall project scheduling problem are provided.

In this paper, we present an ILP formulation for the group scheduling problem, and we show how the input for the group scheduling problem can be extracted from the overall project scheduling problem. Furthermore, we propose a way of translating back the output of the group scheduling problem to the overall project scheduling problem. Afterwards, we show that the group scheduling problem is NP-hard. Since the size of realistic problem instances for the group scheduling problem is not very large, we test the ILP model of the group scheduling problem to see upto what input sizes the problem can be solved within reasonable time. Before we perform these tests, we present a modification of the ILP formulation in order to decrease the size of the ILP.

Contents

Preface i

Abstract ii

Contents iii

1 Introduction 1

2 Problemdescription 3
- 2.1 Introduction . . . 3
- 2.2 Existing solution methods . . . 4
  - 2.2.1 Solution methods for the RCPSP and TCPSP . . . 4
  - 2.2.2 The Two-dimensional Strip Packing Problem . . . 6
  - 2.2.3 The Two-Dimensional Bin Packing Problem . . . 7
  - 2.2.4 The Simple Assembly Line Balancing Problem (SALBP) 8
- 2.3 Objectives of this paper . . . 8

3 Model 10
- 3.1 Introduction . . . 10
- 3.2 The TCPS problem with spatial resources . . . 10
- 3.3 The group scheduling problem . . . 12
- 3.4 Modeling of two-dimensional spatial resources . . . 17

4 Relation between TCPSP and GSP 19
- 4.1 Introduction . . . 19
- 4.2 Extracting the input from the TCPSP . . . 19
  - 4.2.1 Parameters TCPSP . . . 19
  - 4.2.2 Earliest and latest start and completion times . . . 20
  - 4.2.3 Minimum group duration . . . 22
  - 4.2.4 Precedence relations between different groups . . . 26
- 4.3 The solution of the group scheduling problem translated back to the TCPSP . . . 27
- 4.4 Objective functions for the group scheduling model . . . 28
  - 4.4.1 Maximizing the idle times . . . 29
  - 4.4.2 Weighed duration . . . 33
  - 4.4.3 Conclusions . . . 35
- 4.5 Two-phase solution method . . . 35
  - 4.5.1 Introduction . . . 35
  - 4.5.2 Phase one: group allocation . . . 36
  - 4.5.3 Phase two: group sequencing . . . 39

5 Solution method **42**
5.1 Complexity 42
5.2 Input size 44
5.3 Better modeling to decrease input size 44

6 Computational results **48**
6.1 Introduction 48
6.2 Different time horizons 50
6.3 Different scalings of resource capacity and request 51
6.4 Different scalings of release and due dates 53
6.5 Subproblems of GSP 55
6.6 Conclusions 56

7 Conclusions and Recommendations **58**

References **60**

Appendices **64**
Appendix 1: ILP-formulation of GSP 64
Appendix 2: Improved ILP-formulation of GSP 65

1 Introduction

For many companies, project scheduling problems are a big issue. Since companies have a limited amount of resources (machines, manhours, etc.), it is very important to use these resources efficiently. The higher this efficiency, the more projects the companies can handle and the more profit they can make. A high efficiency of the resource usage can only be reached by developing good schedules for the activities that have to be executed.

A special type of resources which companies have to deal with, are spatial resources. Spatial resources are the work spaces, where the activities have to be executed. For example, a dry dock is a spatial resource for a company that builds ships. A special property of these spatial resources, is that the requirement for its resource units, is in an adjacent manner. If we, for example, want to place a ship in a dock, we have to reserve a number of adjacent resource units for it. The adjacency constraint ensures us, that we do not have to break the ship into several pieces to be able to dock it.

The requirement for adjacent resource units not only appears in project scheduling, but also in several other applications. An example is the scheduling of check-in desks at the airport. Depending on the number of passengers and the duration of the check-in period, a number of check-in desks is required for each flight, during a certain time period before departure of the flight. The reserved check-in desks for a certain flight have to be adjacent.

Also in the berthing problem, resource adjacency appears. The berthing problem is the problem of finding an assignment of quay space, when ships of different length berth during specified time periods.

There are also applications in which resource adjacency is not required, but desirable. This happens, for example, within warehouse management and hotel reservations. When a customer wants to hire a number of positions in the warehouse during a certain period, then for convenience and cost-efficiency reasons, it is desirable that these positions are adjacent. Within the reservation of hotel rooms, if a number of guests belonging to a group wants to stay in the hotel, then they might desire adjacent hotel rooms.

In this paper, we present a solution method, that tries to deal with the spatial resource constraints to the project scheduling problem in an efficient way. This solution method not only determines an allocation for the activities to the required spatial resources, where resource adjacency is satisfied, but also determines a sequencing of the activities, that get allocated to the same spatial resource units. The solution method tries to find an allocation and sequencing on the spatial resources that provides good solution possibilities for the remainder of the project scheduling problem.

In the next section, we give a more detailed description of the problem

we want to solve, and we discuss a number of existing solution methods for (special cases of) this problem. In Section 3 we derive an ILP formulation for the problem. In Section 4 we explain how the spatial resource problem is related to the overall project scheduling problem, and in Section 5 we present a solution method. In Section 6 we discuss the performance of the solution method and in Section 7 we give some recommendations for further research.

2 Problemdescription

2.1 Introduction

In many practical instances of the project scheduling problem, a limited work space forms a restriction on the problem. This restriction can have a large influence on the feasibility of project schedules. Therefore, it is very important to use the available space in an efficient way. Before we discuss the problem of finding an efficient spatial resource usage in more detail, we present two project scheduling problems, known from literature.

A well-known project scheduling problem is the RCPSP (see De Boer [14] and Brucker *et al.* [8]), the 'Resource Constraint Project Scheduling Problem'. The RCPSP is called a *resource driven* project scheduling problem, which states that the resource capacities cannot be exceeded and the objective is a minimization function of the completion times of the activities. The *time driven* counterpart of the RCPSP is the TCPSP (see Guldemond [20]), the 'Time Constraint Project Scheduling Problem'. In the TCPSP, deadlines for the activities are given, and in order to meet these deadlines, it might be necessary to hire extra capacity for the resources. The objective of the TCPSP is to minimize the total cost of the hired capacity.

In the standard RCPSP and TCPSP, the work that has to be done is modelled as so-called *activities*. These activities have a certain request for the available resources. Since space can form a restriction to the problem, *spatial resources* were introduced by De Boer [14]. Examples are dry docks in ship yards, shop floor space, pallets, etc. The spatial resources differ from the regular resources in two essential ways.

The first difference is that a resource unit from a spatial resource is not required by a single activity, but by a group of activities, called a *spatial resource activity group* (or just *activity group*). The spatial resource unit is occupied from the first moment an activity from such a group starts until the last activity in the group finishes. Since the start time and completion time of an activity group can be determined by different activities, the processing time of the activity group is variable.

The second difference is that an activity group requires adjacent spatial resource units (see Duin and van Sluis [17]). The assignment of spatial resource units to activity groups, represents the allocation of a product under construction to a certain work space. Since a product under construction cannot be broken into several smaller pieces, the spatial resource units, where a single activity group gets allocated to, have to be adjacent.

In this paper, we derive a solution method, that tries to deal with the additional spatial restrictions on the RCPSP and TCPSP in an efficient way.

The problem that the solution method tries to solve, can be formulated as follows: "Where should we allocate the activity groups, and in which order should the activity groups, that get allocated to the same spatial resource units, be performed, in order to provide good solution possibilities for the overall project scheduling problem (RCPSP or TCPSP)?" We call this problem, the 'Group Scheduling Problem' (GSP). In the remaining of this section, we discuss a number of existing solution methods for (special cases of) the GSP, and we state the objectives of this paper.

2.2 Existing solution methods

The GSP is a new topic in connection with the project scheduling problem and no solution methods for the GSP can be found in the literature. For project scheduling problems, like the RCPSP, many solution methods exist. However, the solution strategies that are used for the RCPSP and TCPSP, are not applicable to the GSP. In this section, we give a brief overview of the solution methods for the RCPSP and TCPSP, and we explain, why these solution methods are not applicable to the GSP.

Although no solution methods for the GSP can be found in literature, there are some special cases of the GSP, which are equivalent to well-known mathematical problems. For these special cases, many solution methods can be found in literature. In this section, we also discuss these special cases of the GSP and we give a brief overview of existing literature on these problems. The first of these problems that we observe is the: 'The Two-Dimensional Strip Packing Problem'. This problem is a special case of the GSP, where only one spatial resource is considered. The second special case of the GSP, is the: 'The Two-Dimensional Bin Packing Problem'. This problem is a special case of the GSP, where more then one spatial resource can be considered. The last special case of the GSP is a generalization of the Bin Packing problem: 'The Simple Assembly Line Balancing Problem'.

2.2.1 Solution methods for the RCPSP and TCPSP

The solution methods for the RCPSP and TCPSP can be divided into optimal procedures and heuristics. The most successful optimal procedures for the RCPSP at this moment, are based on branch and bound methods. We mention the algorithms proposed by Demeulemeester and Herroelen [15] and by Stinson *et al.* [37].

Neumann *et al* [30] showed that the RCPSP is NP-hard. Therefore, unfortunately, even the best optimal procedures known at this moment require too much computation time to solve instances of a reasonable size. As a

consequency, many heuristics have been developed to find good, but not necessarily optimal, solutions within reasonable time. For an extensive overview of these heuristics and the optimal procedures, we refer to Davis [13] and Herroelen and Demeulemeester [21]. Guldemond [20] showed how adaptive search can be used to derive good solutions for the TCPSP.

The reason that existing solution methods for the RCPSP and TCPSP are not applicable to the GSP, is that most of these solution methods, are (partially) based on two observations, that do not hold for the GSP.

The first observation concerns the resource-usage of a set of activities. If, for a set of activities, it holds for every resource, that the total requirement of the activities in the set for that resource does not exceed the capacity of that resource, then all the activities in this set can be in progress at the same time period. A similar observation for the GSP holds for a single time unit, but not when we are scheduling over a certain time period. Namely, if we have, in a similar way, a set of activity groups, such that it holds for every spatial resource, that the total requirement of the activity groups in the set for that spatial resource does not exceed the capacity of that spatial resource, then this set of activity groups can get scheduled at the same time unit. However, when we are scheduling over a certain time period, then the adjacency restriction to the the spatial resource units to which an activity group gets allocated to, makes it impossible to say whether this set of activity groups can get scheduled at the same time period or not. If we, for example, have to allocate two groups, which both require 5 spatial resource units, to a spatial resource with capacity 10, then we can only schedule them at the same time period, if the group which gets started first, gets allocated to one end of the spatial resource. If it gets allocated somewhere in the middle of the spatial resource, then there are not enough adjacent spatial resource units left to allocate the other group at the same time period.

The second observation concerns precedence relations between activities. In both the RCPSP and TCPSP, precedence relations between activities may appear. A precedence relation between two activities states that we cannot start with an activity, before its predecessor is completed. In many of the solution methods for the RCPSP and the TCPSP, these precedence relations are used to decrease the number of activities, that are ready to be scheduled at a certain time, when developing partial schedules. As we show in Section 3.3, these precedence relations between activities can be translated into precedences relations between activity groups. However, the precedence relations between groups is of a different type than the precedence relations between activities. Unfortunately, the precedence relation between groups does not tell if a group cannot start before its preceeding group is completed. Therefore, we cannot decrease the number of activity groups, that are ready

to be scheduled at a certain time, when developing partial schedules.

The existing solution methods for the RCPSP and TCPSP, rely on these two observations. Therefore, the solution methods for the RCPSP and TCPSP cannot be used for the GSP and we have to develop new solution methods.

However, there are some special cases of the GSP, which are equivalent to well-known mathematical problems, for which many solution methods have been developed. In the following subsections, we discuss these cases one by one.

2.2.2 The Two-dimensional Strip Packing Problem

The two-dimensional strip packing problem, first proposed by Baker *et al.* [2], is defined as follows: Pack a set of rectangles into a strip of width 1, such that the height that is used is minimized. The width of the rectangles is assumed to be smaller or equal to 1, and they cannot be rotated. This problem is equivalent to the following special case of the GSP.

Recall that we consider just one spatial resource. Furthermore, assume that there are no precedence relations between the groups, that each group has a fixed processing time, and that without loss of generality the spatial resource has spatial length 1. Then, if we minimize the latest completion time of the groups, the GSP is equivalent with the strip packing problem. If we namely picture the groups as rectangles where the width represents its spatial length and the height represents its duration, then finding a solution of the GSP is equivalent with placing these rectangles into a strip with width 1 (the length of the spatial resource). Minimizing the height of this strip is equivalent with minimizing the latest completion time of the groups (see also Figure 1).

The strip-packing problem is NP-hard (see Baker *et al.* [2]). Therefore, the search for good solution methods for the strip-packing problem has concentrated on heuristics and approximation algorithms. In Baker *et al.* [2], some efficient approximation algorithms are developed and their worst-case bounds are determined. In Martello *et al.* [27], a relaxation of the problem is proposed that produces good lower bounds for the problem. The results are used in a branch-and-bound algorithm, which is able to solve test instances from the literature involving up to 200 items. In Remila [34], an approximation algorithm is developed, that finds a packing of n rectangles, whose total height is within a factor of $(1 + \epsilon)$ of optimal and has running time polynomial both in n and in $\frac{1}{\epsilon}$. In Steinberg [36], an approximation algorithm is proposed with absolute performance bound 2. A more extensive overview of existing solution methods, can be found in Lodi *et al.* [25], where mathe-

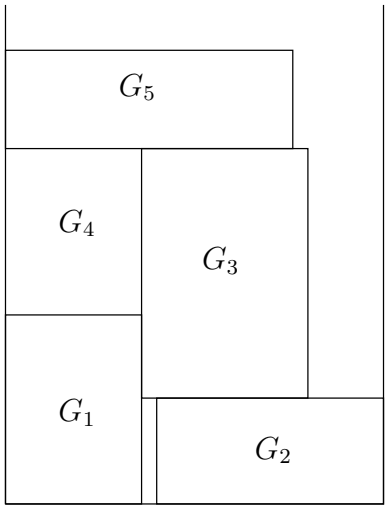

Figure 1: Strip Packing

matical models and lower bounds, classical approximation algorithms, recent heuristic and metaheuristic methods, and exact enumerative approaches are discussed.

2.2.3 The Two-Dimensional Bin Packing Problem

If the GSP has to deal with more then one spatial resource, but the resources are identical, then another well-known mathematical problem appears, namely: 'The Two-Dimensional Bin Packing Problem', which is defined as follows (see Martello *et al.* [26]): Pack a set of rectangular items in identical rectangular bins, such that the number of bins that is used is minimized. The items cannot be rotated. The two-dimensional bin packing problem is equivalent to the following special case of the GSP.

All spatial resources have equal spatial length, the groups have a fixed duration, there are no precedence relations between the groups, and the groups can be allocated to any of the spatial resources. Given a fixed time horizon, the problem is to schedule all the groups within this time horizon, minimizing the number of used spatial resources.

We can set the width of the bins equal to the spatial length of the resources and the height of the bins equal to the time horizon. Then the rectangles represent the groups, where the width of the rectangle equals the spatial length of the group and the height of the rectangle equals the duration of the group. Minimizing the number of bins that we use is equivalent with minimizing the number of spatial resources (see also Figure 2).

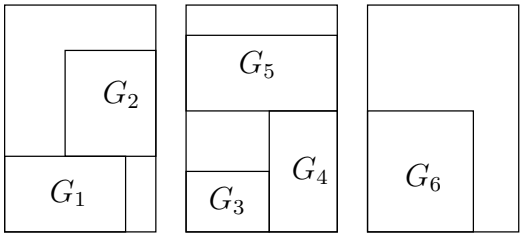

Figure 2: Bin Packing

The two dimensional bin-packing problem is NP-hard, since already the one dimensional bin-packing problem is NP-hard (see Garey and Johnson [19]). Therefore, several heuristics and approximation algorithms for the bin-packing problem have been developed. Berkey and Wang [5] developed several greedy algorithms for the bin-packing problem. Falkenauer and Delchambre [18], proposed a genetic algorithm. Martello *et al.* [26] give a survey of recent advances on exact algorithms and effective heuristics and metaheuristics for the two dimensional bin-packing problem.

2.2.4 The Simple Assembly Line Balancing Problem (SALBP)

If we extend the above given special case of the GSP with precedence relations between the groups, then the following well-known mathematical problem appears: 'The Simple Assembly Line Balancing Problem'. This problem was first proposed by Salveson [35] and is a generalization of the bin packing problem. The SALBP is equivalent with the bin-packing problem where precedence constraints are allowed (see Baybars [4]). Therefore, if we add the possibility of having precedence constraints between the groups to the special case of the GSP, as described in the previous section, then we have a special case of the GSP, which is equivalent with the SALBP.

Since the SALBP is a generalization of the bin packing problem, it is also NP-hard. For an overview of exact algorithms for the SALBP, see Baybars [4]. Bautista *et al.* [3], proposed a local search heuristic and a genetic algorithm for the SALBP. McMullen and Frazier [28] demonstrated how Simulated Annealing can be used to obtain line balancing solutions when one or more objectives are important.

2.3 Objectives of this paper

As described, efficient use of the work space can have a large influence on the solution possibilities for the project scheduling problem. In this paper, we

develop a solution method, that tries to deal with these spatial restrictions in an efficient way. In order to develop a solution method for the GSP, we first need to derive a model for the GSP. We develop an ILP model, which may form the hose for efficient algorithms. To be able to solve the GSP, we need to know, how we can extract the input for the GSP from the overall project scheduling problem. Furthermore, after solving the GSP, we have to translate the output of the GSP back to the overall project scheduling problem. Therefore, we can formally state the following objectives for this paper:

- Develop an ILP model for the Group Scheduling Problem
- Determine how the input for the GSP can be extracted from the overall project scheduling problem, and how the output of the GSP can be translated back to the overall project scheduling problem.
- Derive a model for the GSP, that gives solutions for the GSP, which lead to good solution possibilities for the overall project scheduling problem.
- Find an algorithm, that solves the GSP within reasonable time for realistic input sizes.

3 Model

3.1 Introduction

In this section we derive an ILP model for the GSP. Since the GSP results as a subproblem of the project scheduling problem (RCPSP or TCPSP) with spatial resources, we first review the model for the general project scheduling problem. Therefore, we review the TCPS-model with spatial resources proposed by Guldemond [20] in the next section. Afterwards, we develop the group scheduling model and an ILP formulation for the GSP. In the last section, we discuss in more detail the modeling of the spatial resources.

3.2 The TCPS problem with spatial resources

For the TCPS problem, activities have to be scheduled before a deadline. These activities can be seen as work that has to be done for some project. If we take, for example, the building of a ship, then all the work that has to be done can be divided into activities, like designing the ship, welding several components of the ship, painting the cabin. We assume that we have N activities $A_1, ..., A_N$. Furthermore, we have a time horizon consisting of T time units $t = 1, ..., T$. We assume that every activity A_i has a given processing time p_i, which states that every activity A_i has to be scheduled for exactly p_i time units, and that preemption is not allowed, which states that if we start on an activity, then we have to keep on working on it without interruption, until it is completed.

Each activity A_i also has a release date r_i and a due date d_i. These dates bound the time window in which activity A_i has to be scheduled. The release date of an activity depends on, for example, the delivery date of materials that are necessary for performing this activity.

Between some activities, precedence relations exist. A precedence relation between two activities states that one activity cannot start before the other is finished. For example, a ship cannot get painted, before it is constructed. If an activity A_j cannot start before activity A_i is finished, then A_i is called a predecessor of A_j, and A_j is called a successor of A_i. Sometimes we have to wait a certain amount of time between the completion of an activity and the start of a successor. This happens, for example, if we have to wait for the paint to dry, before we can do other work on the ship. Therefore we introduce on each precedence relation a nonnegative time-lag l_{ij}. This time-lag l_{ij} equals the minimum time difference required, between the completion time of activity A_i and the start time of activity A_j. With P_i we denote the set of predecessors of A_i and with S_i we denote the set of successors of

activity A_i.

To build up a schedule for each activity A_i, we have to determine a start time SA_i, which describes the earliest time bucket at which A_i gets scheduled. Since preemption is not allowed and we have to process A_i for exactly p_i time units, A_i is completed in time bucket $CA_i = SA_i + p_i - 1$. Furthermore, activity A_i cannot get scheduled before its release date or after its due date, which implies $SA_i \geq r_i$ and $CA_i \leq d_i$. The timelag l_{ij} between activity A_j and its predecessor A_i implies $SA_j \geq CA_i + l_{ij} + 1$.

Due to precedence relations, it might be impossible to start an activity at its release date (for example, if activity A_i has $r_i = 2$ and its predecessor A_j has $r_j = 1$, then A_j cannot start at its release date). Therefore we define the earliest start time ESA_i of an activity A_i as the earliest moment in time when an activity can start, taking into account the timing restrictions of all activities. The latest completion time LCA_i of an activity A_i is the latest moment in time at which an activity has to be completed, taking into account the timing restrictions of all activities. Obviously we have $r_i \leq ESA_i$ and $LCA_i \leq d_i$. The earliest start time and latest completion time of an activity define the time window, in which activity A_i can be scheduled.

The earliest completion time ECA_i of an activity equals the sum of its earliest start time and its processing time ($ECA_i = LSA_i + p_i - 1$). The latest start time LSA_i of an activity equals the difference between its latest completion time and its processing time ($LSA_i = LCA_i - p_i + 1$).

For the performance of the activities, a number of resources is required. These resources are, for example, machines and man hours. We assume that there are K resources $R_1, ..., R_K$. Resource R_k has a certain capacity Q_{kt} at time $t \in \{1, ..., T\}$ (the capacity of the resources may vary in time). Each activity A_j requires q_{jk} units of resource R_k during its complete processing. If, at time t, the total request of all scheduled activities for resource R_k exceeds its capacity Q_{kt}, then extra capacity can be hired (against a certain cost). The total cost for hiring this extra capacity is part of the objective of the TCPS problem.

Next to the regular resources, spatial resources appear. These resources are denoted by $SR_1, ..., SR_\Lambda$. The spatial resources are, for example, the work spaces where the activities have to be performed. A spatial resource can be, for example, a dock where the ship has to lay, when a number of activities have to be performed, like building the cabin. An important difference between spatial resources and regular resources is that a resource unit from a spatial resource is not required by a single activity, but by a group of activities, called a *spatial resource activity group* (or just *activity group*), introduced by De Boer [14]. A spatial resource unit is occupied from the first moment an activity from such a group starts until the last activity in

the group finishes. We denote the activity groups by $G_1, ..., G_I$. The set of activities A_i that belong to group G_g is denoted by $S(g)$.

In this paper, we focus on one-dimensional spatial resources, thus, the capacity of the spatial resources is given as a certain length L_λ. This length can be, for example, the length of a dock. Every activity group G_g has a certain length requirement l_g, which is the request for the spatial resource. This length can be, for example, the length of a ship that has to lay in a dock. We consider the case that every group G_g has to be allocated to one specified spatial resource denoted by $SR_{\lambda(g)}$. Since a ship cannot be broken into several pieces, the groups require adjacent spatial resource units (see Duin and Van Sluis [17]).

3.3 The group scheduling problem

The group scheduling problem is the relaxation of the TCPS problem, where the capacities of all regular resources (the regular resources are the non-spatial resources) are set to infinity. Using the group scheduling model, we try to find a scheduling of the activity groups, that provides good solution possibilities for the TCPSP.

For the scheduling of the activity groups, two decisions have to be made. First, we have to decide where the activity groups have to be allocated and second, we have to decide for which period the activity groups get scheduled. In order to provide good solution possibilities for the TCPSP, we at least have to ensure that the output of the group scheduling model leaves the room for feasible, or even better, high quality solutions for the TCPSP.

The period that a group occupies a spatial resource is called the duration of the group D_g. This period lasts from the first moment an activity from such a group starts until the last activity in the group finishes. We denote the start time of group G_g by SG_g and the completion time of group G_g by CG_g. Observe that the duration of a group is variable (if for a certain schedule the start and completion time of a group G_g are determined by different activities and we decrease the start time of the activity that determines the start time of G_g, then the start time of G_g decreases, while the completion time of G_g remains the same, i.e., the duration of G_g increases). However, as we show in Section 4.2.3, we can derive a minimum for the group duration, in order to ensure feasibility for the TCPSP.

To restrict the possible values for the start and the completion of the groups, we derive earliest start, latest start, earliest completion and latest completion times for the groups. The earliest start time ESG_g of a group G_g is defined as the earliest moment in time that an activity belonging to G_g can start, i.e., ESG_g equals the minimum over all the earliest start times

of the activities in G_g. The latest start time LSG_g of a group G_g equals the latest moment in time that G_g can start, such that we can still meet the time constraints of all the activities. Since the start time of a group equals the earliest start time over the activities in the group, and every activity A_i in the group has to start before or at its latest start time LCA_i to meet the time constraints, it follows that LSG_g equals the minimum over all the latest start times of the activities in the group. The earliest completion time ECG_g of a group G_g equals the earliest moment in time when all the activities in the group can be completed, which is equal to the maximum over all the earliest completion times of the activities in the group. The latest completion time LCG_g equals the maximum over all the latest completion times of the activities in the group.

The above considerations only focus on individual groups. However, in general there are also timing relations between the possible processing times of different groups, caused by precedence relations between activities belonging to different groups. To ensure that we can fulfill these restrictions after we derived a schedule for the groups, we have to translate these precedence relations somehow into a precedence relation between different groups. There is only one major difference between the precedence relations between different activities and the precedence relations between different groups. A precedence relation between two activities implies that the two activities have to be scheduled after each other. For groups however, it might happen that they have to be scheduled at (partially) the same period of time, due to the precedence relations between the activities. Assume, for example, that we have two groups G_1 and G_2 and three activities $A_1, A_2 \in S(1)$ and $A_3 \in S(2)$. Furthermore, A_1 is a predecessor of A_3 and A_3 is a predecessor of A_2. To maintain this precedence relations in the TCPSP, we have to schedule the groups G_1 and G_2 (partially) at the same period of time (at least for the period that A_3 gets scheduled).

The precedence relation in the TCPSP between two activities implied that the succeeding activity could not start before a nonnegative time-lag was passed after the preceding activity was completed. This kind of precedence relation is not suitable for groups. Therefore we introduce a precedence relation between groups, that states that if group G_1 is a predecessor of group G_2, then group G_2 cannot complete before a nonnegative time-lag TL_{12} is passed, after group G_1 has started, i.e. $SG_1 + TL_{12} \leq CG_2$. If we get back to our example, then we can see that the fact that A_1 needs to be scheduled before A_3, implies $SG_1 + TL_{12} \leq CG_2$ and the fact that A_2 needs to be scheduled after A_3 implies $SG_2 + TL_{21} \leq CG_1$. Figure 3 clarifies that these two constraints imply that G_1 and G_2 will be scheduled at (partially) the same time period.

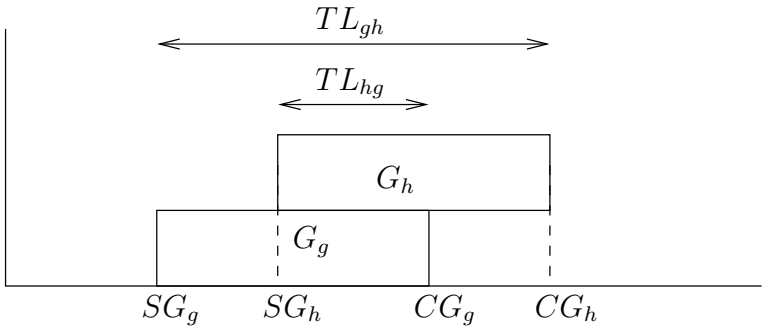

Figure 3: Precedence relations between groups

Deriving from all precedence relations between activities of different groups corresponding timelags between the involved groups leads to a set of precedence relations between the groups. We denote with P_h the resulting set of predecessors of group G_h.

To model the GSP, we derive an ILP model. The necessary decisions for the GSP are the start and finish times for the groups and the spatial resource units which are reserved for each group. For the derivation of the start and completion times of the groups, we use the binary variables gs_{gt} and z_{gt}. The binary variable gs_{gt} takes value 1 if group G_g starts at time unit t and takes value 0 elsewhere. The binary variable z_{gt} takes value 1 if group G_g is busy at time unit t and 0 elsewhere. We can use the following equations to derive the start and completion times of the groups:

$$SG_g = \sum_t gs_{gt} * t \qquad \forall g = 1, ..., I \qquad (3.1)$$

$$CG_g = \sum_t (z_{gt} - z_{gt+1} + gs_{gt+1}) * t \qquad \forall g = 1, ..., I \qquad (3.2)$$

Observe for the first equation that we only add t to the sum when $gs_{gt} = 1$ and that is exactly when group G_g starts. For the second equation, we have 3 situations:

1. $z_{gt} - z_{gt+1} + gs_{gt+1}$ is 0 when group G_g is not busy at time t ($z_{gt} = z_{gt+1} = gs_{gt+1} = 0$ or $z_{gt} = 0$ and $z_{gt+1} = gs_{gt+1} = 1$)

2. $z_{gt} - z_{gt+1} + gs_{gt+1}$ is also 0 when group G_g is busy at time t, but also at time $t + 1$ ($z_{gt} = z_{gt+1} = 1$ and $gs_{gt+1} = 0$)

3. $z_{gt} - z_{gt+1} + gs_{gt+1}$ is 1 when group G_g is busy at time t, but not at time $t + 1$ ($z_{gt} = 1$ and $z_{gt+1} = gs_{gt+1} = 0$).

We see that t only gets added to the sum when t is the last time unit at which G_g is busy. We know that a group G_g cannot start before its earliest start time and not after its latest start time. The same holds for the completion time which implies the constraints:

$$\begin{aligned} z_{gt} = gs_{gt} = 0 \qquad & \forall g = 1, ..., I, t < ESG_g \text{ or } t > LCG_g \\ CG_g \geq ECG_g \qquad & \forall g = 1, ..., I \\ CG_g \leq LCG_g \qquad & \forall g = 1, ..., I \end{aligned}$$

To ensure that every group starts exactly once, we have to add the following constraint:

$$\sum_t gs_{gt} = 1 \qquad \forall g = 1, ..., I \tag{3.3}$$

Furthermore, a group can only be busy at time t if it starts at time t or if it was already busy at time $t-1$, which results in the constraints:

$$z_{gt} \leq z_{gt-1} + gs_{gt} \qquad \forall g = 1, ..., I, t > 1 \tag{3.4}$$

$$z_{g1} = gs_{g1} \qquad \forall g = 1, ..., I \tag{3.5}$$

For the allocation of the groups to the spatial resource units, we use the binary variables b_{gl} and y_{gl}. The variable b_{gl} is set to 1 if l is the first unit of the spatial resource $SR_{\lambda(g)}$ to which group G_g is allocated and 0 elsewhere. Since a group G_g may occupy only one 'first' unit, we have to add the constraint:

$$\sum_{l=1}^{L_{\lambda(g)}} b_{gl} = 1 \qquad \forall g = 1, ..., I \tag{3.6}$$

The variable y_{gl} indicates whether group G_g gets allocated to spatial resource unit l of $SR_{\lambda(g)}$. It takes value 1, when group G_g gets allocated to spatial resource unit l, otherwise it takes value 0. Since the spatial resource units for every group G_g have to be adjacent, group G_g can only get allocated to spatial resource unit l if it is also allocated to spatial resource unit $l-1$ or if spatial resource unit l is the lowest index for which group G_g gets allocated to its spatial resource. This results in the constraints:

$$y_{gl} \leq y_{gl-1} + b_{gl} \qquad \forall g = 1, ..., I, l > 1 \qquad (3.7)$$
$$y_{g1} = b_{g1} \qquad \forall g = 1, ..., I \qquad (3.8)$$

We also have to reserve enough space on the spatial resource for every group G_g. We know that group G_g has to be allocated to l_g spatial resource units of its spatial resource. This is satisfied with the following constraint:

$$\sum_{l=1}^{L_{\lambda(g)}} y_{gl} = l_g \qquad \forall g = 1, ..., I \qquad (3.9)$$

If two groups have to be allocated to the same spatial resource, then we have to make sure that either they do not get allocated to the same spatial resource unit or their scheduling periods do not overlap. To satisfy this, we introduce the binary variable w_{gh}, which takes value 1 if groups G_g and G_h have at least one common unit in their spatial resource allocation and takes value 0 when the scheduling periods of groups G_g and G_h are not disjunct. Since w_{gh} cannot take both values 0 and 1, the variable w_{gh} ensures that at no point in time it cannot happen that two groups are allocated to the same spatial resource unit. The following two constraints guarantee that the variable w_{gh} behaves as described above:

$$y_{gl} + y_{hl} \leq 1 + w_{gh} \qquad \forall g \neq h, \lambda(g) = \lambda(h), l = 1, ..., L_{\lambda(g)} \qquad (3.10)$$
$$z_{gt} + z_{ht} \leq 1 + (1 - w_{gh}) \qquad \forall g \neq h, \lambda(g) = \lambda(h), t = 1, ..., T \qquad (3.11)$$

The last variable is D_g, the duration of group G_g. The duration follows simply from the equation:

$$D_g = CG_g - SG_g \qquad \forall g = 1, ..., I \qquad (3.12)$$

In Section 4.2.3 we derive a minimum for the duration of the group ($D\min_g$), which implies the constraint $D_g \geq D\min_g$. In Section 4.2.4 we derive the time-lag TL_{gh} for the already mentioned constraint $SG_g + TL_{gh} \leq CG_h$. These two constraints ensure that we can meet the time constraints of all the activities if we incorporate the solution of the GSP into the TCPSP.

The objective function of the model is discussed in Section 4.4. The complete constraints of the ILP model for the GSP is summarized in Appendix 1.

3.4 Modeling of two-dimensional spatial resources

In the group scheduling model, the capacity of the spatial resources is modelled as a certain length. This modeling works well for spatial resources like docks, in which ships can only lay in line with each other, but not next to each other. However, there are many applications for which the spatial resource capacity is not restricted to just one dimension. Examples are, shop floor space, rooms, or pallets. If we consider such spatial resources, then the products can not only be placed in line with each other, but also next to each other (and maybe even on top of each other). In other words, such spatial resources have a capacity in two (or even three) dimensions. In this section, we show how we have to adjust the group scheduling model, such that we can also deal with this these higher dimensional spatial resources.

For two dimensions, every spatial resource SR_λ has, next to the length L_λ, a certain width W_λ. Furthermore, for every group, next to the length requirement of the group l_g, a certain width requirement of the group b_g is given. This modeling is a generalization of the modeling in the previous section, because if we take $W_\lambda = 1$ for all spatial resources and $b_g = 1$ for all groups, then we get spatial resources, where the groups can only be placed in line with each other and not next to each other.

For the allocation of the groups to the spatial resource units, we now use the binary variables bl_{gl}, yl_{gl}, bb_{gb} and yb_{gb}. The variable bl_{gl} (bb_{gb}) is set to 1 if l (b) is the first length (width) unit of $SR_{\lambda(g)}$, to which group G_g gets allocated, and 0 in all other cases. To ensure that group G_g has one 'first' length (width) unit of $SR_{\lambda(g)}$, we have to replace (3.6) by:

$$\sum_{l=1}^{L_{\lambda(g)}} bl_{gl} = 1 \qquad \forall g = 1, ..., I$$

$$\sum_{b=1}^{L_{\lambda(g)}} bb_{gb} = 1 \qquad \forall g = 1, ..., I$$

Furthermore, variable yl_{gl} (yb_{gb}) indicates whether group G_g uses the l^{th} $\left(b^{th}\right)$ spatial length (width) unit, or not. To ensure that the spatial length (width) units are adjacent for every group allocation, we have to replace (3.7) and (3.8) by:

$$
\begin{aligned}
yl_{gl} &\leq yl_{gl-1} + bl_{gl} && \forall g = 1, ..., I, l = 2, ..., L_{\lambda(g)} \\
yl_{g1} &= bl_{g1} && \forall g = 1, ..., I \\
yb_{gb} &\leq yb_{gb-1} + bb_{gb} && \forall g = 1, ..., I, b = 2, ..., B_{\lambda(g)} \\
yb_{gb} &= bb_{g1} && \forall g = 1, ..., I
\end{aligned}
$$

To ensure that every group gets enough space on their spatial resource, we have to replace (3.9) by

$$
\begin{aligned}
\sum_{l=1}^{L_{\lambda(g)}} yl_{gl} &= l_g && \forall g = 1, ..., I \\
\sum_{b=1}^{B_{\lambda(g)}} yb_{gb} &= b_g && \forall g = 1, ..., I
\end{aligned}
$$

The last two constraints that we have to adjust are (3.10) and (3.11). Two groups have spatial overlap on a spatial resource if they both have a spatial length unit as a spatial width unit in common in their allocation. Therefore we replace the binary variable w_{gh} by wl_{gh} and wb_{gh}, such that wl_{gh} (wb_{gh}) takes value 1 when groups G_g and G_h get allocated to the same spatial length (width) unit. To avoid spatial resource conflicts, we have to ensure that when group G_g and G_h get scheduled at (partially) the same time period, then at least one of the two binary variables wl_{gh} and wb_{gh} has to take value zero. In this case it is clear that it should not happen that group G_g and G_h have spatial overlap and get scheduled at partially the same time period. This leads to a replacements of (3.10) and (3.11) by:

$$
\begin{aligned}
yl_{gl} + yl_{hl} &\leq 1 + wl_{gh} && \forall g = 1, ..., I, h = 1, ..., I, g \neq h, \\
& && \lambda(g) = \lambda(h), l = 1, ..., L_{\lambda(g)} \\
yb_{gb} + yb_{hb} &\leq 1 + wb_{gh} && \forall g = 1, ..., I, h = 1, ..., I, g \neq h, \\
& && \lambda(g) = \lambda(h), b = 1, ..., B_{\lambda(g)} \\
z_{gt} + z_{ht} &\leq 1 + (2 - wl_{gh} - wb_{gh}) && \forall g = 1, ..., I, h = 1, ..., I, g \neq h, \\
& && t = 1, ..., T
\end{aligned}
$$

With these adjustments, we can use the group scheduling model also for applications where the spatial resources have two dimensions.

4 Relation between TCPSP and GSP

4.1 Introduction

The GSP is a relaxation of the TCPSP. Therefore, most of the input for the GSP can be taken directly from the TCPSP. However, to ensure that the TCPSP is still feasible after solving the GSP, we have to add certain restrictions to the GSP, which cannot be taken directly from the TCPSP. In the previous section, we already mentioned that, in order to ensure feasibility of the TCPSP, a minimum for the group duration has to be derived (Section 4.2.3) and in certain cases also precedence relations between different groups (Section 4.2.4).

After we solve the GSP, we have to translate the output of the GSP back to the TCPSP. This translation can be done in different ways. We discuss in Section 4.3 the advantages and disadvantages of the different ways and show how the translation can be done.

The aim of solving the GSP, is to provide good possibilities for getting solutions of the TCPSP. Whether these solution possibilities are provided or not, depends strongly on the objective function used for solving the group scheduling model. Therefore, we discuss in Section 4.4 possible objective functions and make a choice between these possibilities. In Section 4.5, we propose a two-phase solution method for the GSP.

4.2 Extracting the input from the TCPSP

In this section we derive the input for the group scheduling model, that cannot be taken directly from the TCPS-model. The input which we have to derive are the earliest and latest start times of the groups, the earliest and latest completion times of the groups, the minimum duration of the groups, and the precedence relations between the groups with the associated time-lags. Before we derive this input, we summarize the parameters introduced to the TCPS problem in Section 3.2, and we add some new parameters, which are useful for the derivation.

4.2.1 Parameters TCPSP

We recall the following parameters of the TCPS-model.

A_i : Activity i.

$S(g)$: Set of activities A_i that belong to group G_g.

r_i : Release date of activity A_i.

d_i : Due date of activity A_i.

p_i : Duration of activity A_i.

P_i : Set of direct predecessors of A_i.

S_i : Set of direct successors of A_i ($S_i = \{A_j | A_i \in P_j\}$).

l_{ij} : Minimum time-lag between the completion time of activity A_i and the start time of activity A_j, for $A_j \in S_i$.

$\overline{P_i}$: Set of all predecessors of A_i, i.e., $\overline{P_i} = P_i \cup \left\{\overline{P_j} | A_j \in P_i\right\}$.

$\overline{S_i}$: Set of all successors of A_i $\left(\overline{S_i} = \left\{A_j | A_i \in \overline{P_j}\right\}\right)$.

ESA_i : Earliest start time of activity A_i.

LSA_i : Latest start time of activity A_i.

ECA_i : Earliest completion time of activity A_i.

LCA_i : Latest completion time of activity A_i.

A_{sg} : Artificial start activity of group G_g (explanation in Section 4.2.3).

A_{tg} : Artificial end activity of group G_g (explanation in Section 4.2.3).

4.2.2 Earliest and latest start and completion times

For the derivation of the earliest and latest start times, and the earliest and latest completion times of the groups, we need to make use of the earliest start times and latest completion times of the activities. Therefore, we first show how to derive these parameters.

As mentioned before, some activities cannot start at their release date or finish at their due date, due to precedence relations and timelags. Therefore, we define the earliest start time of an activity, as the earliest moment in time at which an activity can start, such that the time constraints of all activities can be met, and the resource requirements are relaxed. The latest completion time of an activity is defined as the latest moment in time an activity can finish, such that the time constraints of all activities can be met, and the resource requirements are relaxed. We can derive the earliest start times with a forward recursion through the precedence network.

In each step, we have a set C of activities, for which we already determined their earliest start time, and a set D of activities, which are ready to have their earliest start time derived (i.e., for all its predecessors we already

determined their earliest start time). For an activity $A_i \in D$, its earliest start time is bounded by its release date r_i and by the earliest completion time plus the time lag for all its direct predecessors A_j (i.e., $ESA_j + p_j + l_{ij}$). Algorithm 1 summarises this process.

Algorithm 1

Initialize : $C, D := \varnothing$.

Step 1 : $D := \{A_i | P_i \subseteq C, A_i \notin C\}$.

Step 2 : *For every* $A_i \in D$ *do* $ESA_i := \max\left(r_i, \max_{A_j \in P_i}(ESA_j + p_j + l_{ji})\right)$.

Step 3 : $C := C \cup D$. *If* $C = \cup_i A_i$ *then stop. Else go to step 1.*

We can derive the latest completion times in a similar way by a backward recursion through the precedence network. The sets C and D are defined in a similar way as in Algorithm 1.

Algorithm 2

Initialize : $C, D := \varnothing$.

Step 1 : $D := \{A_i | S_i \subseteq C, A_i \notin C\}$.

Step 2 : *For every* $A_i \in D$ *do* $LCA_i := \min\left(d_i, \min_{A_j \in S_i}(LCA_j - p_j - l_{ij})\right)$.

Step 3 : $C := C \cup D$. *If* $C = \cup_i A_i$ *then stop. Else go to step 1.*

The earliest and latest start and completion times of the groups follow from the following equations.

- $ESG_g = \min_{A_i \in S(g)}(ESA_i)$.
- $LSG_g = \min_{A_i \in S(g)}(LCA_i - p_i)$.
- $ECG_g = \max_{A_i \in S(g)}(ESA_i + p_i)$.
- $LCG_g = \max_{A_i \in S(g)}(LCA_i)$.

4.2.3 Minimum group duration

In this section, we derive a minimum on the duration of the groups. This minimum on the group durations is necessary to meet the time constraints of all activities, under the assumption that the regular resources have infinite capacity. As mentioned in Section 3.3, the duration of a group is defined as the period from the first moment an activity within that group starts until the last activity within the group finishes. Since there exist time restrictions on the activities, we can derive a minimum duration for the groups, which is necessary to meet these time restrictions. Before we give a general procedure to derive the minimum group duration, we give a small group example, in which we derive the minimum duration for this group. Afterwards, we show how to derive the minimum group duration in general.

To get a clear view on the problem, we present the example instance in an Activity-On-Node network (AON-network). In this network, the nodes represent the activities, and the arcs the precedence relations. The nodes are labelled with a tuple (ESA_i, LSA_i), where LSA_i is defined as the latest start time of activity A_i (the latest start time of an activity is simply its latest completion time LCA_i minus its processing time p_i). If there is an arc going from activity A_i to A_j, then A_i is a direct predecessor of A_j. We label the arcs with the sum of p_i (the processing time of A_i) and l_{ij} (the time-lag). If an activity A_i has no direct successor, then we write an outgoing arc at its node, labelled with its processing time p_i. Our example is presented in Figure 4.

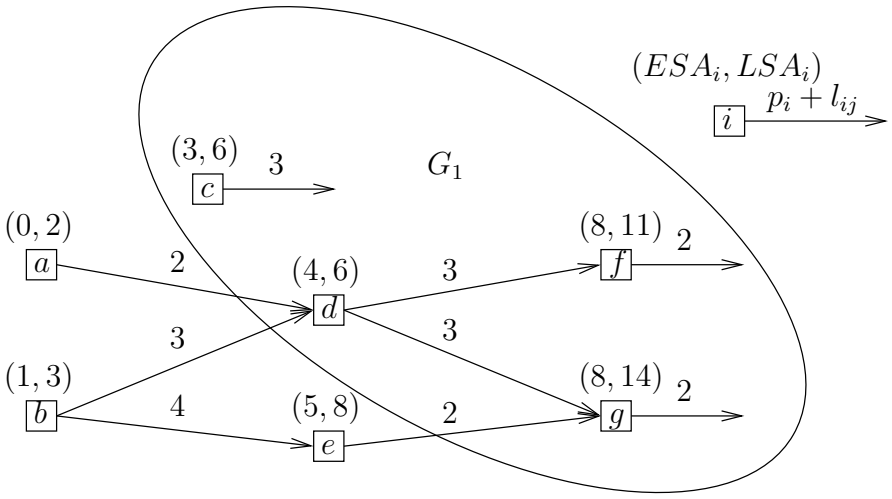

Figure 4: Example instance represented in AON-Network

In the example, group G_1 is given by $S(1) = \{A_c, A_d, A_f, A_g\}$. This group is presented in Figure 5. The other activities belong to no group.

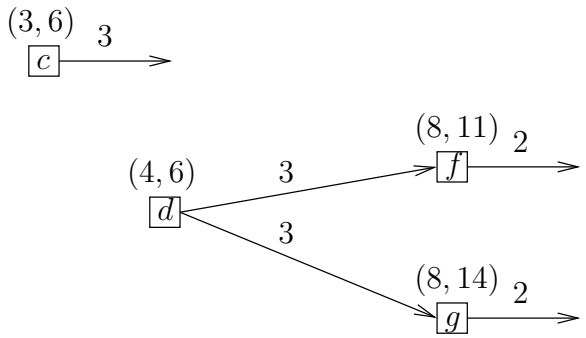

Figure 5: Activities in group G_1, extracted from figure 4

It is easy to see that in this example the minimum duration of group G_1 equals 5. This duration can be attained if we start activity A_c and A_d at time 6, and activity A_f and A_g at time 9. On the other hand, the duration of group G_1 cannot be smaller then 5, since the sum of the processing times of activities A_d and A_f, and the time-lag between these two activities equals 5. Now we show a method to derive the minimum duration in general.

To derive the minimum duration of group G_g in general, we first add two dummy-activities (A_{sg} and A_{tg}) to group G_g, where A_{sg} represents the start of group G_g and A_{tg} represents the completion of group G_g. We use these dummy activities, to give a generic algorithm for the derivation of the minimum group duration. We define the processing times of A_{sg} and A_{tg} to be 0, and we define $SA_{sg} := SG_g$ and $SA_{tg} := CG_g$. Furthermore, we define A_{sg} to be a direct predecessor of all the activities in G_g, that do not have any predecessors within G_g (i.e., $A_{sg} \in P_i$ if and only if $\overline{P_i} \cap G_g = \varnothing$), and we define A_{tg} to be a direct successor of all the activities in G_g, that do not have any successors in G_g (i.e., $A_{tg} \in S_i$ if and only if $\overline{S_i} \cap G_g = \varnothing$). All the time-lags for precedence relations in which A_{sg} or A_{tg} are involved are put to zero. In Figure 6, we see the example group G_{g_1}, where the activities A_{sg} and A_{tg} are added to the group.

For group G_1 we can derive:

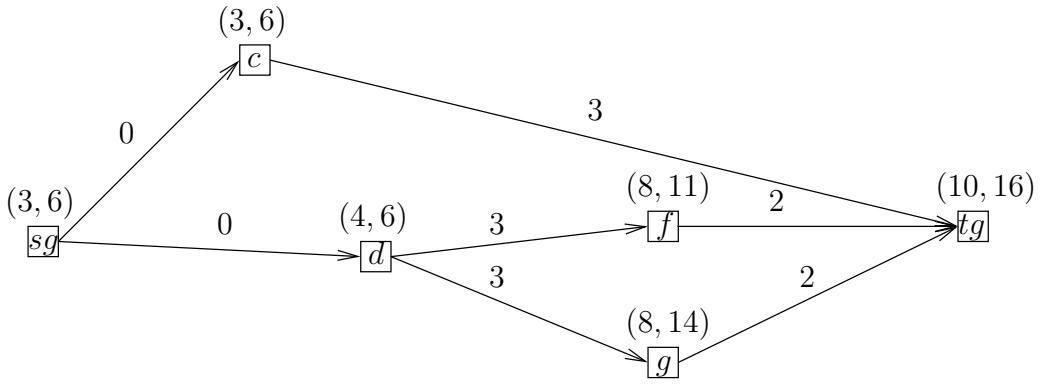

Figure 6: Group G_{g_1} with dummy-activities A_{sg} and A_{tg}

$$ESA_{sg} = ESG_1 = \min_{A_i \in S(1)} (ESA_i) = 3.$$
$$LSA_{sg} = LSG_1 = \min_{A_i \in S(1)} (LSA_i) = 6.$$
$$ESA_{tg} = ECG_1 = \max_{A_i \in S(1)} (ESA_i + p_i) = 10.$$
$$LSA_{tg} = LCG_1 = \max_{A_i \in S(1)} (LSA_i + p_i) = 16.$$

The group duration equals the difference in start times between A_{sg} and A_{tg} ($SA_{sg} = SG_g$ and $SA_{tg} = CG_g$ and $D_g = CG_g - SG_g$). Finding a minimum for the group duration, that is necessary to provide feasibility of the TCPSP, is now equivalent with finding the smallest possible difference between SA_{sg} and SA_{tg}, such that the time restrictions of all activities within G_g, can still be met. In the following, we show how to find this smallest difference.

We first let every activity A_i start at its earliest start time: $SA_i = ESA_i$. This gives a feasible schedule (an activity can always start at its earliest start time if all its predecessors start at their earliest start time). Next, we fix the start time of activity A_{tg} at its earliest start time, and we increase the start times of its predecessors, by a backward recursion through the AON-network, until we reach A_{sg}. In every recursion step, we increase the start times of the activities as much as possible, taking into account the already changed start time of the successors of this activity. Thus, by construction, we increase the start time of A_{sg} as much as possible, without having to increase the start time of A_{tg}. We claim that the minimum duration of group G_g equals the difference between the derived start times of A_{sg} and A_{tg}.

To see that this claim is true, observe that the following two arguments imply that we cannot decrease the duration anymore. First, we cannot decrease the start time of A_{tg}, since it was already set to its earliest start time. Second, we cannot increase the start time of A_{sg}, without having to increase the start time of A_{tg} by the same amount, or hitting a deadline.

If we execute the process described above for group G_1 from our example, we start with setting the start time of activity A_{tg} to its earliest start time, i.e., $SA_{tg} = 10$. Now we can increase the start times of its predecessors, in the following recursive way: $SA_c = 6$, $SA_f = 8$, $SA_g = 8$, and $SA_d = 5$, and $SA_{sg} = 5$, yielding a minimum duration of 5 for the group.

In the example, we only had to consider the activities within group G_1. However, it might be possible that there exist directed paths in the AON-network between two activities within G_1, but visiting activities outside G_1. Therefore, in order to derive a minimum duration for group G_g (which is necessary to meet the timing constraints of all activities, and not just the activities in group G_g itself), we also need to consider the activities outside G_g. The activities that lay on a directed path from A_{sg} to A_{tg} (this holds by definition for every activity within G_g) can have an influence on the minimum duration of G_g. Therefore, we can derive the minimum group duration, step by step, as follows.

In the first step, we set activity A_{tg} to its earliest start time, and we set every activity A_i in G_g, which is not a predecessor of A_{tg} (and therefore it does not lay on a directed path from A_{sg} to A_{tg}) to its latest start time LSA_i. In each following step, we have a set C of activities for which we already determined the latest possible start time, taking into account the start times of its successors. Furthermore, we have a set D of activities, which are ready to have their latest possible start time determined. For an activity $A_i \in D$, its latest possible start time is bounded by its latest start time LSA_i, and by the latest possible start time minus the timelag and the duration of all its direct successors A_j (i.e. $LSA_j - p_j - l_{ij}$). Algorithm 3 summarizes this process which ha to be applied to each group in order to derive the minimum durations.

Algorithm 3

Initialize: $\widetilde{SA_{tg}} := ESG_g$.
$\widetilde{SA_i} := LSA_i, \forall A_i \notin \overline{P_{tg}}$.
$C := \{A_{tg}\} \cup \left\{A_i | A_i \notin \overline{P_{tg}}\right\}$.

Step 1: $D := \{A_i | S_i \subseteq C, A_i \notin C\}$.

Step 2: *For every* $A_i \in D$ *do* $\widetilde{SA_i} := \min\left(LSA_i, \min_{A_j \in S_i} \widetilde{SA_j} - l_{ij} - p_i\right)$.

Step 3: $C := C \cup D$.
If $A_{sg} \in C$, *then return:* $D\min_g = \widetilde{SA_{tg}} - \widetilde{SA_{sg}}$.
Else go to step 1.

4.2.4 Precedence relations between different groups

In Section 3.3, we already showed that, in order to provide feasibility of the TCPSP, we have to translate the precedence relations between activities belonging to different groups into precedence relations between the corresponding groups. Furthermore, we showed that multiple precedence relations between activities belonging to two (or more) different groups, might imply that we have to schedule these groups at (partially) the same time period. Therefore, we defined precedence relation on groups, which state that a group G_g is a predecessor of group G_h, if there exist a nonnegative timelag TL_{gh} between the start time of G_g and the completion time of G_h, i.e., $SG_g + TL_{gh} \leq CG_h$ with $TL_{gh} \geq 0$. Note, that using this definition, it is possible that G_g is a predecessor of G_h and vice versa.

Observe now, that for every two groups G_g and G_h, for which there exist a directed path from an activity $A_i \in S(g)$ to $A_j \in S(h)$ in the corresponding AON-network, it holds that G_g is a predecessor of G_h. Namely, if such a directed path exist, then, by construction of A_{sg} and A_{tg}, there exist a directed path from A_{sg} to A_i, a directed path from A_i to A_j, and a directed path from A_j to A_{th}, which implies that there exist a directed path from A_{sg} to A_{th}. Therefore, the timelag TL_{gh}, which is the minimum difference between the start time of G_g and the completion time of G_h to maintain feasibility of the TCPSP, equals the minimum time difference between SA_{sg} and SA_{th}, such that the time-restrictions of all activities can be met.

In the previous subsection, we showed how to derive the minimum time difference, between the start time and completion time of a single group, in order to meet the time restrictions of all activities. To derive the minimum difference between the completion time of a group, and the start time of its

predecessor, we can follow a similar strategy. The minimum time difference, between SA_{sg} and SA_{th}, is influenced by all directed paths between SA_{sg} and SA_{th} in the AON-network. Algorithm 3 calculates the minimum time difference between the activities SA_{sg} and SA_{tg}, taking into account all directed paths between SA_{sg} and SA_{tg}. Therefore, we can determine, for every pair of groups G_g and G_h, whether G_g is a predecessor of G_h, and if so, the timelag TL_{gh}, by substituting A_{th} for A_{tg} in Algorithm 3.

4.3 The solution of the group scheduling problem translated back to the TCPSP

The group scheduling model solves a part of the overall problem, the TCPSP. It provides an allocation of the activity groups to the spatial resources, and a scheduling of the activity groups, which is part of the solution for the TCPSP. Therefore, the group scheduling model simplifies the TCPSP.

We can relate the output of the group scheduling model in different ways to the TCPSP. One way is to fix the allocations and the schedules of the groups, as it is in the output of the group scheduling model. This implies a strong simplification of the TCPSP. Another possibility, is to use the output of the group scheduling model, just as an indication of the allocation and the scheduling of the activity groups. This implies a smaller simplification of the TCPSP. An example of such an indication is to fix the allocation of the groups to the spatial resources, but not the group schedules, i.e. the start and completion times of the groups. However, in order to provide feasible schedules for the TCPSP, we have to ensure that groups, that get allocated to the same spatial resource unit, will not get scheduled at the same time period. One way to ensure this, is to fix the sequences in which such groups get scheduled in the output of the group scheduling model. Fixing the sequence of groups can be done by adding precedence constraints between the activities within these groups. To motivate the chosen strategy to relate the output to the TCPSP, observe the following.

We can use the objective function in the group scheduling model to look forward to the TCPSP. If we make a good choice for the objective function, then we may get a group schedule that provides good solution possibilities for the TCPSP. However, this only works well if we do not restrict the TCPSP too much. If we fix the scheduling of the groups completely, then it will be very hard (probably even impossible through resource constraints) to get a good solution for the TCPSP. The problem is, that the GSP is a relaxation of the TCPSP. The GSP 'ignores' the regular resources. Observing only activity groups and spatial resources, we cannot predict in detail, at what moments in

time we can expect resource conflicts at the regular resources when we try to schedule the activities, and to which activity groups these activities belong. In the GSP, we view the project scheduling problem at group level, which is a more global view, than viewing the problem at activity level. Therefore, to prevent resource conflicts, it may be better to keep the group schedules flexible. For this reason, we opt for the second option that we mentioned in the previous paragraph, fix the allocation of the groups to the spatial resources and fix the sequences of the groups groups, which get allocated to the same spatial resource unit. We can maintain the group sequences as follows.

Assume that group G_g has to be scheduled before group G_h. This implies that all the activities in group G_g have to be completed before we can start with any of the activities in G_h. We can ensure this by adding precedence relations between all pairs of activities $(A_i, A_j) \in (S(g), S(h))$, which makes activity A_i a direct predecessor of A_j with time-lag zero. These precedence relations restrict the TCPSP, which makes it more easy to solve. In the following section, we discuss a number of objective functions, that might provide good solution possibilities for the TCPSP.

4.4 Objective functions for the group scheduling model

Till now, we mentioned only the constraints of the GSP and how we want to incorporate the solution of the GSP into the TCPSP. We still have to choose an objective function in the GSP. This objective function has a large influence on the group schedules in the output of the model. But since the output of the group scheduling model is translated back to the TCPSP, the objective function of the group scheduling model also has a large influence on the solution possibilities for the TCPSP. The choice of the objective function should be motivated by this last observation and is certainly not trivial. Therefore, we discuss in this section a number of objective functions, that might provide good solution possibilities for the TCPSP. Before we get to the first objective function, we discuss what kind of allocations and sequences of activity groups may provide good solution possibilities for the TCPSP.

In the group scheduling model, we have relaxed the capacities of all regular resources R_k in the TCPSP. We have set their capacities to infinity. In the TCPSP however, these capacities can have a large influence on the group schedules that provide good solution possibilities. For example, a simple observation tells us that we have to schedule every group at least long enough, such that every regular resource can deliver enough capacity, to meet the total request for that resource from the activities within the group. If the request is too high in a certain period for a certain resource, then we need

to hire extra capacity. To minimize this extra capacity, we have to keep the group schedules as flexible as possible.

The easiest way to keep a group schedule flexible, is to allocate a group to spatial resource units, to which no other group is allocated. If this holds for a certain group, then the start time and completion time of the group are not restricted by other groups because of having (partially) the same allocation. As a consequence, no extra time restrictions are added to the activities within that group. Therefore, we try to find objective functions which search for an equal distribution of the groups over the spatial resource units.

4.4.1 Maximizing the idle times

The first objective we discuss, maximizes the idle times of the spatial resource units. We call a spatial resource unit l idle at time t if there is no group allocated to it at time t. If we maximize the times that the spatial resource units are idle, then the objective function keeps the group durations short. When the group durations are short, then there is much space for the groups to increase their completion time or to decrease their start time. In other words, the group schedules are flexible with respect to the TCPSP.

We use the binary variable $i_{\lambda lt}$ to indicate whether spatial resource unit l of spatial resource SR_λ is idle at time t or not. This variable takes value 1 if the corresponding spatial resource unit is idle at time t and it takes value 0 when it is occupied at time t. We can do this with the following constraint:

$$i_{\lambda lt} \leq 2 - y_{gl} - z_{gt} \qquad \forall \lambda = 1, ..., \Lambda, g = 1, ..., I, \lambda(g) = \lambda, \\ l = 1, ..., L_\lambda, t = 1, ..., T$$

$$i_{\lambda lt} \in \{0, 1\}$$

Recall that y_{gl} takes value 1 when group G_g gets allocated to spatial resource unit l and z_{gt} takes value 1 when group G_g is scheduled at time t. We can see that $i_{\lambda lt}$ only can take value 1, if every group G_g, which gets scheduled on spatial resource SR_λ, is not allocated to spatial resource unit l ($y_{gl} = 0$) and/or it is not scheduled at time t ($z_{gt} = 0$). Otherwise it is pushed to 0.

This objective function, simply maximizing the total idle time, can easily cause some problems. If we use this objective function, then a situation as represented in Figure 7 can be the output of the model. If in this example all the groups are scheduled for their minimum duration, then we cannot create any extra idle times. The amount of idle time is therefore maximal, so we have an optimal solution.

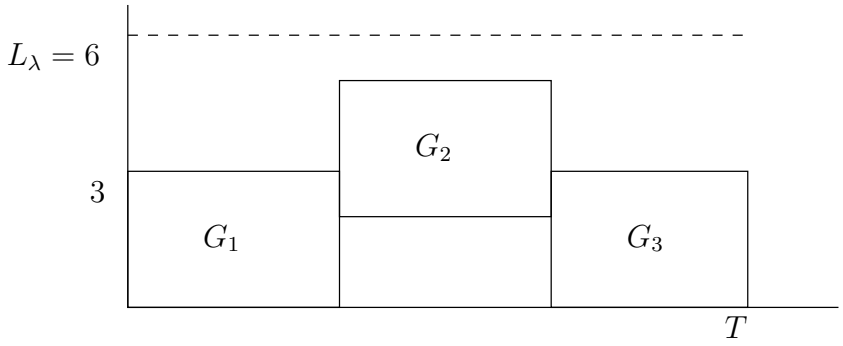

Figure 7: Bad solution when maximizing the total idle time

However, in this example the flexibility of the group schedules is minimal. None of the completion times can increase and none of the start times can decrease. A better distribution of the groups over the spatial resource units, and therefore more flexibility for the group schedules, can be created by shifting group G_2 to the highest spatial resource units. This results in the solution represented in Figure 8. The solution in Figure 7 makes the flexibility of all group schedules minimal, while the solution in Figure 8, gives maximal flexibility to group G_2, and it results in a sequencing of the groups G_1 and G_3.

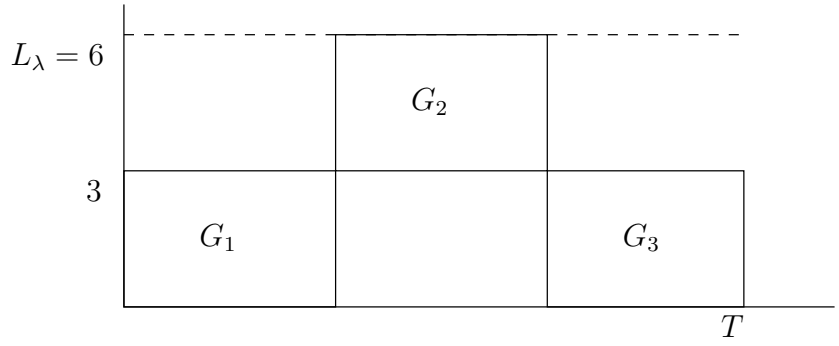

Figure 8: Better solution than in Figure 7

Since the objective 'maximizing the total idle time' does not make any distinction between the solutions represented in Figures 7 and 8, we need a slightly modified objective function, that does make this distinction. A suitable objective is to maximize the minimum total idle time over the spatial resource units, per spatial resource. If we do this for our example, then the

objective value of the solution in Figure 7 becomes 0 (spatial unit 3 has no idle times), but for the solution in Figure 8 it becomes equal to the minimum duration of group G_2 (this minimum is taken at spatial resource units 1, 2 and 3, which are idle during the period that group G_2 is scheduled). Since we want to maximize the minimum total idle time over the spatial resource units, per spatial resource, we derive the minimum total idle time over the spatial resource units for every spatial resource, and maximize the total sum of these minima, i.e.:

$$\max \sum_{\lambda} \left(\min_{l} \sum_{t} i_{\lambda l t} \right)$$

It can happen that for some spatial resources it is more difficult to find a good distribution of the groups, then for other spatial resources. This happens, for example, if the total request for the spatial resource units from the groups on a spatial resource, is high with respect to the capacity of the spatial resource. Therefore, it might be better to use a weighed sum that we want to maximize, where 'difficult' spatial resources get a higher weight. If we call this weight w_λ for spatial resource SR_λ, then our objective becomes:

$$\max \sum_{\lambda} \left(w_\lambda * \min_{l} \sum_{t} i_{\lambda l t} \right) \tag{4.1}$$

Another problem that can occur, is that there is one group on a spatial resource that makes the minimum idle time for that spatial resource very bad. A clear example is a group that needs to be scheduled for the whole period T. Minimizing the maximal idle time for that spatial resource will always result in an objective value of 0, and the objective function does not search for a good distribution of the remaining groups on the remaining spatial resource units. However, if this situation occurs in practice, then we can easily deal with it by doing some pre-processing. We can allocate, for example, the group to the first spatial resource units of the spatial resource and we decrease the length of the spatial resource by the length of the group. Afterwards we can remove this activity group from the TCPS-model.

Another problem that can occur using this objective function, cannot be handled as easily in practice. This problem is illustrated in Figure 9, where the groups are scheduled on one spatial resource for their minimum durations. All groups have the same minimum duration and the total number of spatial resource units that they need is higher than the length of the spatial

resource. It is easy to see that the solution in Figure 9 is optimal with respect to objective (4.1) (the minimum of the total idle times is twice the duration of a group).

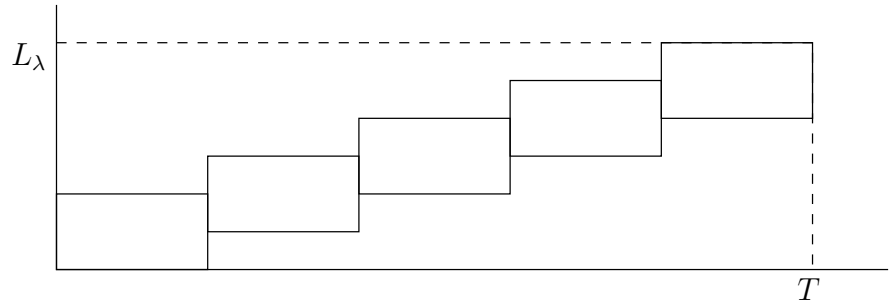

Figure 9: Bad solution for maximizing minimum idle time

However, in this example the flexibility of the group schedules is minimal, while we can increase this flexibility, by moving some of the groups:

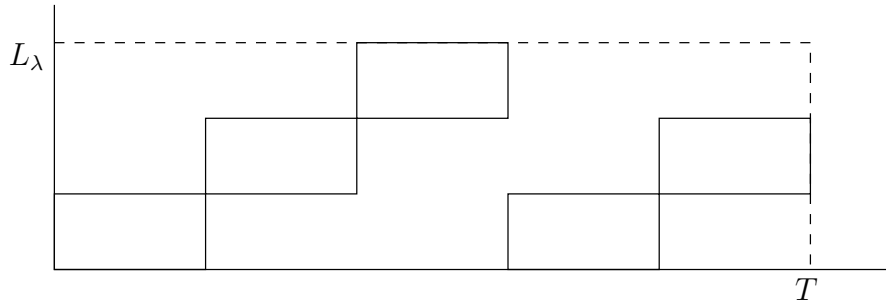

Figure 10: Better solution than in Figure 9

To overcome this problem, we have to take a more local view of the problem. With the objective 'maximize the sum of the minimum total idle times of the spatial resources', we took a global view of the problem. The idea was to find an objective, that keeps as much space as possible for the groups, to increase their flexibility. However, it might happen, as we showed in the last example, that this objective does not work well. In the example, every group has a few spatial resource units, for which it is very flexible, but not for all spatial resource units where it gets allocated to, which made the flexibility for the group schedules minimal. Therefore, we discuss in the remainder of this section a number of objective functions that take a more local view of the problem.

4.4.2 Weighed duration

In this subsection, we try to keep the group schedules as flexible as possible, by maximizing the durations of the groups. Observing the group durations of the groups directly, we take a more local view of the problem than in the previous section, which must help us to overcome problems as in Figure 9.

In general, the activities within different groups, also have a different total request for the regular resources. Therefore, it might be more important for one group to get a long duration, then for another group. This especially holds for activity groups, where the activities within that group have a very high total request for one specific regular resource. Increasing the group duration might make it possible to schedule the activities within this group in series, which can prevent resource conflicts in the TCPSP. Therefore, we give a weight to every group, which indicates the importance for this group to get a long duration. The way we derive these weights will be discussed later in this subsection. We represent these weights by W_g. This leads to the objective function that maximizes the weighed sum of the durations:

$$\max \sum_{g} (W_g * D_g)$$

A disadvantage of this objective function is that it might happen that just a few groups with a high weight get their duration maximized, and some other groups, with a slightly lower weight, get scheduled for their minimum duration. A possibility to overcome this problem is maximizing the minimum of the weighed durations over the groups. We now need to use the inverse weight $\frac{1}{W_g}$, to increase the durations of the groups with a high weight more then the groups with a smaller weight. Since the possibility of enlarging the durations of groups strongly depends on the space that is left on their spatial resource, we maximize the minimum of the weighed durations per spatial resource. We get:

$$\max \sum_{\lambda} \min_{g|\lambda(g)=\lambda} \left(\frac{1}{W_g} * D_g \right)$$

In the remainder of this subsection, we discuss how to derive the weights W_g, and the different factors that have influence on the weights.

The first factor is the total request for the regular resources from the activities within the group. Groups need to get scheduled long enough, such that every regular resource can accommodate the total request for that resource of the activities within the group. Especially, groups with a high

request for one specific regular resource need to be scheduled for a long duration and therefore need to get a higher weight. We can formulate this factor as follows.

For every resource R_k, we first determine the total request of the activities within group G_g. The total request for a resource R_k of a single activity A_i equals $q_{ik} * p_i$, so the total request (q_{gk}) of the activities within group G_g for resource R_k equals $q_{gk} = \sum_{A_i \in S(g)} q_{ik} * p_i$. To fulfill these requests, we need to schedule the groups long enough. How long we need to schedule the groups, depends on the capacity of resource R_k per time unit. Therefore, we divide the total requests for the resources by their capacities, where we denote the resource capacity by $Q_k = \max_t Q_{kt}$. We get for every resource R_k the value $\frac{q_{gk}}{Q_k}$, which states how many time units we need to schedule G_g at least, in order to fulfill the total request of the activities within G_g for R_k. We define the weight W_g to be the maximum of these values over the resource R_k, since it is most likely to encounter capacity problems for this maximum. We get:

$$
\begin{aligned}
W_g &= \max_k \frac{q_{gk}}{Q_k} && && (4.2)\\
q_{gk} &= \sum_{A_i \in S(g)} q_{ik} * p_i && \forall g = 1, ..., I, k = 1, ..., K\\
Q_k &= \max_t Q_{kt} && \forall k = 1, ..., K
\end{aligned}
$$

The second factor has to do with groups, that surely have to be scheduled (partially) at the same time period. This happens, for example, when there are two groups G_g and G_h, such that the intervals $[LSG_g, ECG_g]$ and $[LSG_h, ECG_h]$ intersect (every group surely needs to get scheduled from its latest start time to its earliest completion time). For these two groups we know for sure, that they both have to get scheduled during this intersection period. During this period, the activities within both groups, require from the regular resources. Therefore, if both groups have a large total request for a particular resource, then there is a big chance that we get resource conflicts during this period. In that case, it is very important to increase the flexibility of the group schedules of these groups. Therefore, we might want to add this factor somehow to the weight. This can be done in several ways. One way is to take for each group, the time that it is scheduled together with the other group as a percentage of its minimum duration. Then we can multiply the weight with this percentage.

A third factor we can use for the derivation of W_g, is the amount of precedence relations between the activities within a group. This amount can be seen as the flexibility of the schedules for the activities within the group.

If there are many precedence relations between the activities within a group, then this flexibility is small. For such a group, increasing the duration of the group has just little effect on this flexibility. The flexibility seems to increase at first sight, but if we fix just a few schedules of the activities within the group, the flexibility for the other activity schedules might be small again. On the other hand, if there are few precedence relations, then increasing the group duration has a stronger effect on the flexibility of the schedules of the activities within the group. Therefore, we might give a higher weight to groups, where many precedence relations exist between the activities within these groups.

4.4.3 Conclusions

The aim of the GSP, is to find a group allocation and sequencing, which provides good solution possibilities for the TCPSP. In this section, we derived two objective functions, which were supposed to find a good group allocation and sequencing. However, test results showed that both the objective 'maximizing the idle times' and the objective 'maximizing the weighed duration' did produce, to a certain extent, good allocations for the groups, but in certain cases they did not make any distinction between the possible group sequences. Therefore, we choose to derive two new objective functions, where the first objective function is concentrated on searching for good group allocations, and the second objective function is concentrated on searching for good group sequences. These two objective functions form a two-phase solution method for the GSP, where in the first phase a good group allocation is being determined and in the second phase a good group sequencing.

4.5 Two-phase solution method

4.5.1 Introduction

The group scheduling model solves two parts of the TCPSP. The first part is the allocation of the groups to the spatial resources, and the second part is the sequencing of groups which got allocated to the same spatial resource units. In the previous section, we observed a number of objective functions that tried to solve these two parts. However, test results showed that the derived objective functions only searched for good group allocations. For the determination of good group sequences, the objective functions were most of the time useless. This happened, for example, when two groups with (almost) the same schedule period (the schedule period for a group G_g is defined as the time period $[ESG_g, LCG_g]$), got allocated to the same spatial resource

unit. Since they got allocated to the same spatial resource unit, a sequence between these two groups had to be determined. If they also did not have any precedence relations with any other group, both group sequences were possible. However, for the objective functions from the previous section, both sequences lead to the same objective value, and therefore the sequence is arbitrary.

Since the derived objective functions only worked reasonable for the allocation of the groups, we decided to derive two new objective functions, which solve the GSP in two phases. The first objective function concentrates on the allocation of the groups and forms the first-phase of the solution method. The second objective function concentrates on the sequencing of the groups and forms the second phase of the solution method.

In some cases, it happens that the GSP forms only a sequencing problem and not an allocation problem. This happens, for example, when all spatial resources have spatial length 1. In that case, we only need to run phase two of our two-phase solution method. On the other hand, if we find an allocation in the first phase, such that no group sequences have to be determined anymore (for example, when there are no different groups allocated to the same spatial resource units), then we do not need to run phase two anymore. In the following sections, we discuss the two phases in more detail.

4.5.2 Phase one: group allocation

In the first phase, we search for a good allocation of the groups to the spatial resources. As mentioned before, a good allocation is an allocation where the group schedules are as flexible as possible. This flexibility decreases, when two groups, with overlapping schedule periods, get allocated to the same spatial resource unit. Two groups G_g and G_h have an overlapping schedule period, when both $ESG_g \leq LCG_h$ and $ESG_h \leq LCG_g$ hold, as shown in Figure 11.

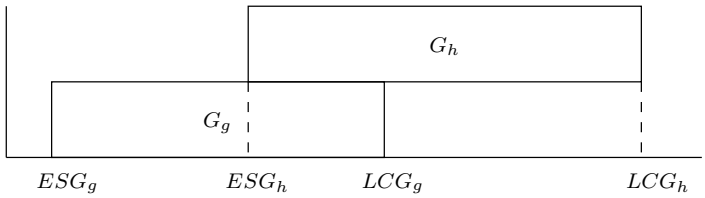

Figure 11: Overlapping schedule periods

It is easy to see that if these two groups get allocated to the same spatial resource unit, the flexibility of the group schedule for at least one of the two

groups gets reduced. Therefore we, will penalize if groups with overlapping schedule periods get allocated to the same spatial resource unit. The objective of the first phase is to minimize this penalty. Before we formally state the objective function, we discuss in more detail the penalty we want to use and how it can be derived.

Penalties need to be derived, when groups with overlapping schedule periods get allocated to the same spatial resource units. Therefore, we define on every pair of groups G_g and G_h, which have overlapping schedule periods (i.e., $[ESG_g, LCG_g] \cap [ESG_h, LCG_h] \neq \varnothing$) and which have to be allocated to the same spatial resource (i.e., $\lambda(g) = \lambda(h)$), a nonnegative penalty p_{gh}. When these two groups do not get allocated to the same spatial resource unit, this penalty takes value 0. When these two groups do get allocated to the same spatial resource unit, then p_{gh} will be derived as follows.

Reducing the flexibility of group schedules can cause resource conflicts, when solving the remaining TCPSP. This is especially the case, when groups with a high weight W_g (as in 4.2), get a large flexibility reduction of their group schedule. Therefore, we use the following two factors as input for the penalty p_{gh}:

- The bigger the request for the regular resources from the activities within the group, the bigger the penalty.
- The bigger the reduction of the flexibility of the group schedule, the bigger the penalty.

As already mentioned, we can use the weight W_g for the first factor. For the second factor, we use the reduction of the flexibility of the group schedule relative to the maximum flexibility of the group schedule, i.e., $\frac{LCG_g - ESG_g - D_g}{LCG_g - ESG_g}$. Since flexibility reduction can always be spread over two groups (with each flexibility reduction, two different groups are involved), we define the penalty p_{gh} as the maximum of the weighed flexibility reductions of the group schedules over the groups G_g and G_h:

$$p_{gh} \geq \frac{LCG_g - ESG_g - D_g}{LCG_g - ESG_g} * W_g \tag{4.3}$$

$$p_{gh} \geq \frac{LCG_h - ESG_h - D_h}{LCG_h - ESG_h} * W_h \tag{4.4}$$

These constraints only hold if the groups G_g and G_h have overlapping schedule periods and if they get allocated to the same spatial resource unit. We can satisfy this by the following modification of (4.3) and (4.4). Recall

that the binary variable w_{gh} is set to 1 when the groups G_g and G_h get allocated to the same spatial resource unit.

$$p_{gh} \geq 0 \tag{4.5}$$

$$p_{gh} \geq \frac{LCG_g - ESG_g - D_g - T(1 - w_{gh})}{LCG_g - ESG_g} * W_g \tag{4.6}$$

$$p_{gh} \geq \frac{LCG_h - ESG_h - D_h - T(1 - w_{gh})}{LCG_h - ESG_h} * W_h \tag{4.7}$$

$$\forall \lambda\left(g\right) = \lambda\left(h\right), g \neq h, ESG_g \leq LCG_h, ESG_h \leq LCG_g$$

If G_g and G_h do not get allocated to the same spatial resource unit, then w_{gh} can be put to zero and the constraints (4.6) and (4.7) get redundant (when $w_{gh} = 0$, the right hand sides of (4.6) and (4.7) are smaller then zero), and the penalty p_{gh} can be pushed to zero. Furthermore, we define the penalties p_{gh} only for those pairs of groups G_g and G_h, which have to be performed on the same spatial resource ($\lambda\left(g\right) = \lambda\left(h\right)$), and which have an overlapping schedule period ($ESG_g \leq LCG_h, ESG_h \leq LCG_g$).

Our objective is to minimize the total penalty. We can do that in several ways (minimize the sum of the penalties, minimize the maximal penalty, etc.). Since the allocation problem can be harder for one spatial resource then another, we can get dominating penalties on that resource. Therefore we minimize the penalty per spatial resource. We do that by taking the maximal penalty for every spatial resource and minimize the sum of these maxima. We define p_λ to be the maximum penalty for spatial resource SR_λ:

$$p_\lambda \geq p_{gh} \qquad \forall \lambda = 1, ..., \Lambda, g \neq h, \lambda(g) = \lambda(h) = \lambda, \; ESG_g \leq LCG_h, ESG_h \leq LCG_g \tag{4.8}$$

Using these values, we get the following objective:

$$\min \sum_{\lambda} p_\lambda \tag{4.9}$$

This is the solution approach for the first phase. Of course there are other ways to define penalties. The best way to define penalties depends on the cases you work on in practice. It might happen, for example, that for certain regular resources, hiring extra capacity is very expensive or even impossible. If activities in a group have a (large) request for these resources, then it is

very important for these groups that the flexibility of their schedule period is large.

The output of this phase provides a good allocation of the groups to the spatial resources. However, it does not necessarily provide good sequences for the groups which get allocated to the same spatial resource unit. In the next section we derive an objective function that searches for good group sequences, which forms phase two of our solution method.

4.5.3 Phase two: group sequencing

Before we derive the objective function that has to search for good group sequences, we need to discuss in which situations we actually have to determine group sequences. We mentioned earlier, that if two groups get allocated to the same spatial resource unit, then we have to decide in which sequence these two groups have to get scheduled. However, in many cases there is just one sequence possible. Between two groups G_g and G_h, there are only two sequences possible if both $ECG_g \leq LSG_h$ and $LSG_g \geq ECG_h$ holds. This is clarified in Figure 12:

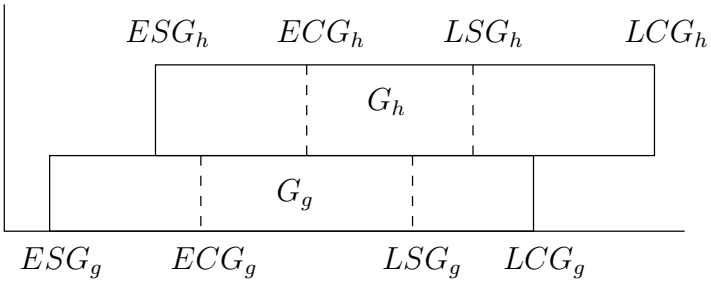

Figure 12: Both sequences possible for group G_g and G_h

If in this case, where both sequences are possible, also the penalty p_λ is the same for both sequences, then the sequence in which groups G_g and G_h get scheduled in the first phase, will be randomly chosen. The two possible sequences do not have a different influence on the flexibility of the group schedules of the groups G_g and G_h. However, we introduce in this section some new activity groups, for which different sequences do have different influences on the flexibility of the group schedules.

Assume that group G_g and G_h are two groups, which have to be sequenced. If group G_g gets scheduled before G_h, then it will lose flexibility to the right side of its schedule period, which forces certain activities within G_g to finish before their latest completion time. But then, as a consequence,

it might happen that the direct predecessors of these activities, also have to finish before their latest completion time. Therefore, scheduling group G_g first, might result in a reduction of the flexibility of the activity schedules of the direct predecessors of the activities within G_g. In a similar way we can deduce, that scheduling G_g after G_h might result in a reduction of the flexibility of the activity schedules of the direct successors of the activities within G_g. These flexibility reductions might cause resource conflicts when solving the TCPSP and therefore we penalize the reduction of the flexibility of these schedule periods. Since we observe the project scheduling problem on a group level, we introduce the following groups in this phase of the solution method.

The group $G_{pre(g)}$ is defined as the group of activities that are direct predecessor of one or more activities in $S(g)$. The group $G_{suc(g)}$ is defined as the group of activities that are direct successor of one or more activities in $S(g)$. For example, in the group example represented in Figures 4 and 5, we can see that $S(pre(g_1)) = \{A_a, A_b\}$ and $S(suc(g_1)) = \emptyset$. Since all the activities in group $G_{pre(g)}$ have a direct successor in G_g, its completion time will always be strictly smaller then the completion time of G_g. In other words, there exist a positive time-lag $TL_{g,pre(g)}$ between the completion time $CG_{pre(g)}$ of group $G_{pre(g)}$ and the completion time of group G_g. On the other hand, since all the activities in group $G_{suc(g)}$ have a direct predecessor in G_g, its start time will always be strictly larger then the start time of G_g. So there also exist a positive timelag $TL_{g,suc(g)}$ between the start time of group G_g and the start time $SG_{suc(g)}$ of group $G_{suc(g)}$.

Consider now again the two groups G_g and G_h for which we have to determine the best sequence. Suppose that we schedule G_g first. Then the flexibility of the schedule period of G_g gets reduced from the right, which implies by the time-lag $TL_{g,pre(g)}$ that also the flexibility of the group schedule of $G_{pre(g)}$ gets reduced. We penalize this flexibility reduction in a similar way as how we did in the first phase. We can deduce similarly that also the flexibility of the group schedule of $G_{suc(h)}$ gets reduced.

In other words, scheduling G_g before G_h can imply a flexibility reduction of the group schedules of $G_{pre(g)}$ and $G_{suc(h)}$, and scheduling G_h before G_g can imply a flexibility reduction of the group schedules of $G_{pre(h)}$ and $G_{suc(g)}$. We penalize these flexibility reductions and we try to minimize these penalties to find the best possible sequences. These penalties are similar to the penalties in the previous section and therefore we need to derive the weight $w_{pre(g)}$ for group $G_{pre(g)}$ and the weight $w_{suc(g)}$ for group $G_{suc(g)}$. We also have to add a binary variable v_{gh} to the model, to indicate whether group G_g gets scheduled first or group G_h. The variable v_{gh} takes value 0 when G_g gets scheduled first and it takes value 1 when G_h gets scheduled first. The second

phase becomes as follows.

First we fix the allocation of the groups as in the output of the first phase. This implies that we can leave the constraints (3.6) to (3.11) out of the model and the variables b_{gl}, y_{gl} and w_{gh} become given parameters with the values from the output of the first phase. For every pair of groups G_g and G_h, which satisfy $w_{gh} = 1$, $ECG_g \leq LSG_h$ and $LSG_g \geq ECG_h$, we introduce the groups $G_{pre(g)}$, $G_{suc(g)}$, $G_{pre(h)}$ and $G_{suc(h)}$, and the binary variable v_{gh}. As in the first phase, we try to minimize the maximum penalty per spatial resource, but now we also consider the groups $G_{pre(g)}$ and $G_{suc(g)}$ and therefore we have to add the following constraints:

$$p_{gh} \geq \frac{LCG_{pre(g)} - ESG_{pre(g)} - D_{pre(g)} - Tv_{gh}}{LCG_{pre(g)} - ESG_{pre(g)}} * w_{pre(g)} \tag{4.10}$$

$$p_{gh} \geq \frac{LCG_{suc(h)} - ESG_{suc(h)} - D_{suc(h)} - Tv_{gh}}{LCG_{suc(h)} - ESG_{suc(h)}} * w_{suc(h)} \tag{4.11}$$

$$p_{gh} \geq \frac{LCG_{suc(g)} - ESG_{suc(g)} - D_{suc(g)} - T(1 - v_{gh})}{LCG_{suc(g)} - ESG_{suc(g)}} * w_{suc(g)} \tag{4.12}$$

$$p_{gh} \geq \frac{LCG_{pre(h)} - ESG_{pre(h)} - D_{pre(h)} - T(1 - v_{gh})}{LCG_{pre(h)} - ESG_{pre(h)}} * w_{pre(h)} \tag{4.13}$$

$$\forall g \neq h, w_{gh} = 1, ECG_g \leq LSG_h, LSG_g \geq ECG_h$$

Observe that constraints (4.12) and (4.13) are redundant when $v_{gh} = 0$ and (4.10) and (4.11) when $v_{gh} = 1$ (the schedule period of a group is always smaller then the complete time horizon T). So when $v_{gh} = 0$, we just consider the penalties for $G_{pre(g)}$ (4.10) and $G_{suc(h)}$ (4.11), which is what we want, since $v_{gh} = 0$ indicates that G_g starts before G_h. On the other hand, with $v_{gh} = 1$ we indicate that G_h starts before G_g, which is satisfied, since $v_{gh} = 1$ implicates that we just consider the penalties for $G_{suc(g)}$ (4.12) and $G_{pre(h)}$ (4.13).

In the second phase we keep the constraints (4.6) and (4.7). We do this, because the sequencing of G_g and G_h can have influence on the schedule periods of G_g and G_h and therefore on the flexibility of the group schedules of these groups. The objective function in the second phase is the same as in the first phase (4.9), where we derive p_λ as in (4.8).

5 Solution method

Till now, we derived an ILP formulation for the GSP, we discussed a number of objective functions for the GSP, and we mentioned how we want to incorporate the solution of the GSP into the TCPSP. In this section, we discuss how we are going to solve the GSP.

Applying the GSP in practice makes only sense if it can be solved within reasonable time for realistic input sizes. The time to solve the GSP, depends on the running time of the algorithm we use. The running time of an algorithm is determined by the complexity of the algorithm, and the input size of the instance. The complexity of the algorithm tells how fast the running time grows in terms of the input size. An algorithm is called efficient, when its running time is bounded by a polynomial in the input size. If an algorithm is not efficient, then the running time of the algorithm explodes when we increase the input size.

If a problem belongs to the complexity class P, then there exist an efficient algorithm for that problem. If a problem is NP-hard, then it is highly unlikely that there exist an efficient algorithm. The latter is the case for the GSP, as we show in Section 5.1. Therefore, solving it to optimality is only possible for small input sizes of the GSP.

Since it is unlikely to find an algorithm, that solves the GSP to optimality within reasonable time, we resort to heuristic approaches. Heuristics are solution methods that try to find good, but not necessarily optimal, solutions within reasonable time.

In the next two subsections, we discuss the complexity and the size of the GSP. We start with the complexity of the problem, where we also give a short introduction to the notion of P and NP, which are complexity classes for problems. Furthermore we explain what we mean with NP-complete problems and NP-hard problems. Afterwards, in Section 5.2 we discuss the size of the ILP model of the GSP for realistic input sizes of the GSP. We finish this section by giving a modification of the ILP formulation, which reduces the number of variables significantly. A formulation with a smaller number of variables makes ILP solvers work, in general, faster.

5.1 Complexity

In this section, we discuss the complexity of the GSP. The complexity of a problem tells something about the possibilities of the running time of algorithms for it. The running time of an algorithm can be expressed in the number of elementary calculations it has to execute to solve the problem. This number depends on the input size of the problem. The input size of a

problem is the number of characters that represents the input.

An algorithm is called polynomial, when its running time is polynomial in the input size, i.e., the number of elementary calculations is of $O(p(n))$ where $p(n)$ is a polynomial function in n. If there exist a polynomial time algorithm for a certain problem, then this problem belongs to the class P, the class of problems which can be solved within polynomial time. The class P is a subclass of the class NP. A problem belongs to the class NP, if there exist an algorithm that can verify a given solution to the problem within polynomial time. If we can solve a problem within polynomial time, then we can also verify a given solution to the problem within polynomial time. Therefore, we have $P \subseteq NP$. Whether also $NP \subseteq P$ holds or not, is still an unsolved question in mathematics.

A special class of problems in NP is the class of NP-complete problems. A problem is called NP-complete, if we can reduce any other problem in NP to this problem within polynomial time. Therefore, if we can find a polynomial algorithm for an NP-complete problem, then we can solve every problem in NP within polynomial time and we get $NP \subseteq P$. The NP-complete problems are the hardest problems in NP to solve.

The problems in P and NP are decision problems. For every optimization problem, like the GSP, we have an associated decision problem. When the objective of an optimization problem is to minimize something, then the associated decision problem is to determine if there exist a solution that does not exceed a certain specified value. Optimization problems for which their associated decision problem is NP-complete are called NP-hard.

One of the problems that belongs to the class of NP-hard problems is the Partion Problem: "Given a set of n integers $\{a_1, ..., a_n\}$. Does there exists a partition of the index set $I = 1, ..., n$ into two subsets S_1 and S_2, such that $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j$."

Consider now the following instance of the GSP. We have 1 spatial resource with 1 spatial resource unit. Assume that we have $T = \sum_{j=1}^{n} a_j + 1$ time units and that we have $n+1$ groups such that the minimum duration of groups G_j equals a_j, for $j = 1, ..., n$. Furthermore, we have that the minimum duration of group G_{n+1} is 1 and that $ESG_{n+1} = LSG_{n+1} = \frac{1}{2}\sum_{j=1}^{n} a_j$ and $ECG_{n+1} = LCG_{n+1} = \frac{1}{2}\sum_{j=1}^{n} a_j + 1$. In other words, there is only one possibility to schedule G_{n+1}, namely from $t = \frac{1}{2}\sum_{j=1}^{n} a_j$ until $t = \frac{1}{2}\sum_{j=1}^{n} a_j + 1$. The other groups have no restrictions to their start and completion times and neither are there precedence relations between the groups. Independent of the objective function that is used, we only get a feasible schedule if we partition the groups G_1 to G_n into two sets of groups, such that the sum of the minimum durations in both these sets equal $\frac{1}{2}\sum_{j=1}^{n} a_j$, as shown in

Figure 13.

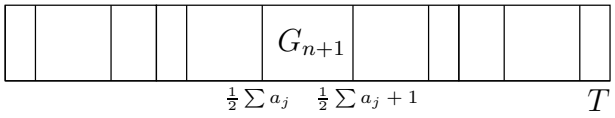

Figure 13: NP-hardness

Finding a feasible schedule for this small instance of the GSP is therefore equivalent with the Partition Problem. Therefore, the GSP is NP-hard.

Observe that this small instance (all groups and spatial resources have spatial length 1) shows that even when the allocation of all groups is known, the GSP is NP-hard. On the other hand, we can show that if the timing is known for all groups, the remaining part of the GSP, i.e. the allocation of the groups, is also NP-hard. Namely, if we know the timing of all groups, then we know for every group their start and completion time, and our problem is equivalent with the ARS-R problem, as described in Duin and van Sluis ([17]). Duin and van Sluis ([17]) showed that the ARS-R is strongly NP-hard, and therefore also the GSP, where the timing of all groups is known, is strongly NP-hard.

5.2 Input size

In the previous section, we showed that the GSP is NP-hard. As a consequence, it is highly unlikely that we can find an efficient algorithm for the GSP, and therefore solving the GSP to optimality in reasonable time is only possible for inputs of small size. In this section, we discuss the size of the ILP, which we get for realistic input sizes of the GSP. If the size of the ILP becomes too large, then we need to use other solution methods, like heuristics to solve the GSP.

The following input values represent already quite large instances of problems in practice. Assume that we have an instance with 20 groups ($I = 20$), 200 weeks ($T = 200$), and 5 spatial resources ($\Lambda = 5$) with each 20 spatial resource units ($L_\lambda = 20$). This leads to an ILP model with the following variables: gs_{gt},z_{gt},b_{gl},y_{gl},w_{gh},SG_g,CG_g,D_g. For inputs of the mentioned size, we get a total of 9260 integer variables and 88520 constraints.

5.3 Better modeling to decrease input size

Although the GSP is NP-hard, we try to solve the problem to optimality, since realistic input sizes for the GSP are not too large. In order to decrease

the running time of an ILP solver, we propose some modifications of the group scheduling model. These modifications decrease the number of variables in the ILP model, without changing the group allocations and sequencings in the output of the ILP. Therefore, these modifications may decrease the running time to solve the GSP, but do not affect the quality of the solutions of the GSP.

The variables that have the biggest contribution to the total number of variables are the variables with two indices: $gs_{gt}, z_{gt}, b_{gl}, y_{gl}, w_{gh}$. Since we have to make sure that groups do not have both overlap in time and in allocation, we have to keep the variable w_{gh}. Therefore, we try to reformulate the group scheduling model, without using the variables gs_{gt}, z_{gt}, b_{gl} and y_{gl}.

If we throw the mentioned variables away, then we also throw away the constraints (3.1) until (3.11). These constraints determined the allocation of the groups, the schedule of the groups, and they ensured that groups do not have both overlap in time and allocation. We replace these constraints as follows.

For every group G_g we define the variables SL_g ($0 \leq SL_g \leq L_{\lambda(g)}$) and CL_g ($0 \leq CL_g \leq L_{\lambda(g)}$), where SL_g represents the lowest value for which G_g gets allocated to its spatial resource and CL_g represents the highest value for which G_g gets allocated to its spatial resource. Since we have to reserve enough spatial length for every group on its spatial resource, we have to add the following constraint:

$$CL_g - SL_g = l_g \qquad \forall g = 1, ..., I \tag{5.1}$$

When two groups G_g and G_h have overlap in allocation, w_{gh} has to take value 1. To satisfy this, observe the following. If two groups G_g and G_h do not overlap in allocation, then either $SL_g \geq CL_h$ or $SL_h \geq CL_g$ holds. When they do overlap in allocation, then neither $SL_g \geq CL_h$ nor $SL_h \geq CL_g$ holds. Therefore, the following two constraints ensure that w_{gh} takes value 1, when G_g and G_h have overlap in allocation, and w_{gh} can take value 0, when they do not overlap in allocation.

$$w_{gh} \geq \frac{\left(CL_h - SL_g - L_{\lambda(g)} * y_{gh}\right)}{L_{\lambda(g)}} \qquad \forall g = 1, ..., I, h = 1, ..., I, \; g \neq h, \lambda(g) = \lambda(h) \qquad (5.2)$$

$$w_{gh} \geq \frac{\left(CL_g - SL_h - L_{\lambda(g)} * (1 - y_{gh})\right)}{L_{\lambda(g)}} \qquad \forall g = 1, ..., I, h = 1, ..., I, \; g \neq h, \lambda(g) = \lambda(h) \qquad (5.3)$$

$$y_{gh} \in \{0, 1\} \qquad (5.4)$$

If the groups G_g and G_h do not overlap in allocation, then exactly one of the values $CL_h - SL_g$ and $CL_g - SL_h$ is strictly positive and one of these values is nonpositive. Furthermore, these values are always smaller then or equal to $L_{\lambda(g)}$. Therefore, making the right choice for y_{gh} (for example, if $CL_h - SL_g > 0$, then we choose $y_{gh} = 1$), we can make both right hand sides of (5.2) and (5.3) nonpositive and w_{gh} can take value 0. If G_g and G_h do have overlap in allocation, then we cannot do this, because then both the values $CL_h - SL_g$ and $CL_g - SL_h$ are strictly positive, and we can only make one of the right hand sides of (5.2) and (5.3) nonpositive with our choice for y_{gh}. So in case of allocation overlap, one of the right hand sides of (5.2) and (5.3) is strictly positive, and since we divide both right hand sides by $L_{\lambda(g)}$, this value is always smaller or equal to 1. We do not get infeasibility and w_{gh} will be pushed to 1.

If the groups G_g and G_h do overlap in allocation, then we have to ensure that these groups do not overlap in time. We can do that with the following two constraints.

$$SG_g + T * (1 + v_{gh} - w_{gh}) \geq CG_h \qquad \forall g = 1, ..., I, h = 1, ..., I, \; g \neq h, \lambda(g) = \lambda(h) \qquad (5.5)$$

$$SG_h + T * (2 - v_{gh} - w_{gh}) \geq CG_g \qquad \forall g = 1, ..., I, h = 1, ..., I, \; g \neq h, \lambda(g) = \lambda(h) \qquad (5.6)$$

If the groups G_g and G_h do overlap in allocation, then $w_{gh} = 1$. In this case, if we take $v_{gh} = 0$, then (5.5) implies that group G_g cannot start before G_h is completed. If we take $v_{gh} = 1$, then (5.6) implies that group G_h cannot start before G_g is completed. Since $T \geq CG_g - SG_h$ holds for every G_g and G_h, (5.6) is redundant when $v_{gh} = 0$ and (5.5) is redundant when $v_{gh} = 1$. If the groups G_g and G_h do not overlap in allocation ($w_{gh} = 0$), then (5.5) and (5.6) are both redundant, no matter what choice we make for v_{gh}.

The new model can be found in Appendix 2. The variables in the new group scheduling model are: w_{gh},y_{gh},v_{gh},SG_g,CG_g,D_g,SL_g,CL_g. If we take the same input sizes as mentioned in the previous section, then we only have 730 variables, instead of the 9260 variables, and 1940 constraints instead of 88520, before the modification of the ILP model. Therefore, an ILP solver using Branch and Bound techniques may be much faster on this new formulation.

6 Computational results

6.1 Introduction

As we showed in the previous section, the GSP is NP-hard. Therefore, it is highly unlikely that there exist an efficient algorithm to solve the GSP to optimality. However, as we also mentioned, realistic input sizes for the GSP are not very large. In order to see upto what input sizes the GSP can be solved to optimality within reasonable time, we test several problem instances for the group scheduling model. The problem instances we use, are benchmark instances generated by the parameterdriven project generator ProGen, which was developed by Kolisch *et al.* [24]. The project generator ProGen has been widely used as a tool for the evaluation of algorithms proposed for resource-constrained project scheduling.

The benchmark problems generated by ProGen, can be obtained from the library PSPLIB [33]. For our tests, we modified the benchmark problems generated for the standard RCPSP. These problem instances consider 30, 60, 90 or 120 jobs (activity groups). Within the GSP, every group needs to get allocated to exactly 1 spatial resource. Therefore, we only observe those problem instances where every job has a requirement for exactly 1 resource.

The benchmark instances are generated for the standard RCPSP. Since there are some differences between the RCPSP and the GSP, we made some adjustments to the problem instances. First, we assume that the requirement for the resources is in an adjacent manner. Second, we assume that the given job durations are minimum durations for the activity groups. Within the benchmark instances, precedence relations on the jobs exist, but no time-lags. These precedence relations state that a job cannot start before all its predecessors are completed. However, as we discussed in Section 3.3, a precedence relation between two activity groups is defined as a nonnegative timelag between the start time of the preceeding group and the completion time of its successor. Therefore, we add for every precedence relation a timelag equal to the sum of the minimum group durations of the two groups involved in the precedence relation. Figure 14 clarifies that this timelag implies that group G_h can get scheduled after group G_g (and if fixed durations are concerned instead of minimum durations, then G_h has to get scheduled after its predecessor G_g).

In Section 4.4, we discussed a number of objective functions for the group scheduling model. However, a few preliminary test results showed that these objectives only work reasonable for the allocation problem of the GSP. Therefore, we developed the two-phase solution method in Section 4.5, which we assume to be the most promising method to solve the GSP. Therefore, we

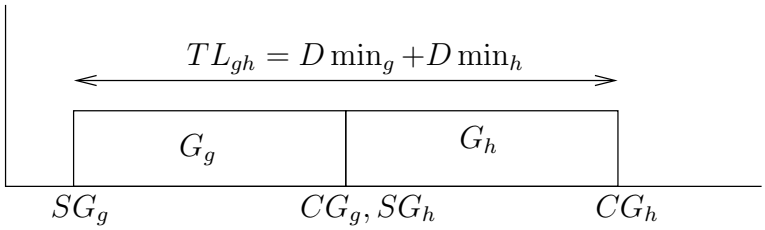

Figure 14: Adding timelags to the benchmark problems

only test the GSP for the two-phase solution method. As we will show in the end of this section, phase 1 of the two-phase solution method forms the bottleneck concerning the running time. Therefore, we only test the GSP for the first phase of the two-phase solution method.

Running just a few tests already shows that problem instances where 30 groups (or more) are considered are too large to be solved to optimality within reasonable time. Therefore, we only consider the first n groups of the benchmark instances for our tests, where we try to find the maximum value of n for which the GSP can be solved within reasonable time. We also want to test the GSP for different numbers of spatial resources. However, all the benchmark instances consider 4 spatial resources. In order to create instances considering 1 or 2 spatial resources, we propose the following adjustments of the instances.

We create instances considering 1 spatial resource as follows. For every problem instance, the capacity of the spatial resource equals the sum of the capacities of the 4 resources in the original instance. The requirement from a group for the spatial resource equals the sum of the requirement over all 4 resources in the original instance. We create instances considering 2 spatial resources in a similar way. The resource capacity of spatial resource 1 (2) equals the sum of the capacities of resources 1 (2) and 3 (4) in the original instance, and the resource requirement from a single group for spatial resource 1 (2) equals the sum of the resource requirements of the group for resources 1 (2) and 3 (4) in the original instance.

The tests for the GSP are organized as follows. First, we test the problem instances with different deadlines as no deadlines are provided for the jobs within the benchmark instances. Afterwards, we test the problem instances with different adjustments to the resource capacities and to the requirements of the activity groups for the resources. Next, we test the influence of release and due dates on the solver time. Since the benchmark problems do not provide release and due dates for the activity groups (the earliest start times and latest completion times of the groups only depend on the precedence

relations), we introduce release and due dates for the groups by adjusting the earliest start times and latest completion times of the groups. Finally, we test the two subproblems of the GSP, where all group allocations are known, or where all group schedules are known.

We programmed the ILP model of the GSP as presented in Section 5.3 in AIMMS 3.6 [1] and XA 14 [38] is used as an MIP solver. At the end of this section, we derive some conclusions about the input sizes for which the GSP can be solved to optimality within reasonable time.

6.2 Different time horizons

The benchmark problems do not provide a deadline T. Therefore, we introduce deadlines. We let the deadlines vary from T_{min} to T_{max}, where T_{min} equals the maximum over the earliest completion times of the groups and T_{max} equals the sum of the minimum durations of the groups. In other words, T_{min} is the earliest moment in time at which all groups can be completed when we relax the resource constraints, and T_{max} is earliest deadline for which all groups can get scheduled one after the other. Therefore, the deadline T_{max} does not form a restriction to the allocation of the groups, and the allocations will not change by increasing T_{max}.

We test the problem instances for different numbers of groups, different numbers of spatial resources, and different deadlines. We test instances with 1, 2 and 4 spatial resources. Furthermore, we test the problem instances for the deadlines T_{min}, T_{max} and a number of deadlines $T_{min} + a * \Delta$, where $0 < a < 1$ and $\Delta = T_{max} - T_{min}$. For every combination of number of groups, number of spatial resources and deadline T, we solve 80 problem instances. Preliminary tests showed that most problem instances which could not be solved within 100 seconds, also could not be solved within 1 hour. Therefore, we decide to interrupt the solver after 100 seconds (if necessary) and we present the results as the percentage of the 80 problem instances which were solved within 100 seconds.

When we test problem instances with 6 groups, all problem instances for all different configurations of spatial resources and deadlines are solved within 100 seconds. The results of the problem instances with 8 and 10 groups are presented in Table 1.

As we can see, increasing the deadline makes it harder to solve the problem instances. This can be explained by the fact that increasing the deadline might imply more possible group sequences, and therefore more feasible solutions. The problem instances considering 8 groups did not cause many problems. More problems occur, when we try to solve the problem instances where 10 groups and 1 or 2 spatial resources are considered. It seems a

Table 1: Results for 8 and 10 groups with different deadlines

# Groups	# SR	T_{min}	$T_{min}+\frac{1}{9}\Delta$	$T_{min}+\frac{2}{9}\Delta$	$T_{min}+\frac{3}{9}\Delta$	$T_{min}+\frac{5}{9}\Delta$	T_{max}
$I = 8$	$\Lambda = 1$	100%	98.8%	97.5%	95%	93.8%	93.8%
	$\Lambda = 2$	100%	97.5%	96.3%	95%	63.8%	93.8%
	$\Lambda = 4$	100%	100%	97.5%	96.2%	96.2%	97.5%
$I = 10$	$\Lambda = 1$	92.5%	80%	71.2%	70%	66.2%	65%
	$\Lambda = 2$	77.5%	66.2%	57.5%	52.5%	47.5%	46.2%
	$\Lambda = 4$	98.8%	98.8%	97.5%	98.8%	97.5%	97.5%

little strange that most problems occur when 2 spatial resources are considered. One might expect that when fewer spatial resources are considered, the problem becomes harder due to the increase of the number of possible group sequences.

However, there might be another reason for the fact that fewer problems occur when we consider 1 spatial resource instead of 2 spatial resources. As mentioned, the benchmark instances only consider 4 spatial resources. In order to create instances considering 1 and 2 spatial resources, we added capacities of different spatial resources and we added requirements for different spatial resources. But by adding these capacities and requirements, we relaxed the resource constraints in the problem instances a bit, which might make it more easy to find an optimal schedule. This relaxation is the strongest when we consider only 1 spatial resource, and that can be the reason for having more problems solving instances considering 2 spatial resources instead of 1 spatial resource. To get an idea of the influence of the resource capacities and requirements on the solver time, we developed some tests which are presented in the next subsection.

6.3 Different scalings of resource capacity and request

As mentioned in the previous section, adjusting the problem instances to instances considering 1 or 2 spatial resources has relaxed the resource capacity. On the other hand, in the benchmark problems, the request from the jobs for the resources vary from 1 to 10 resource units. However, in many practical instances, the spatial length of different groups does not vary that much. Therefore, we scale the capacity of the resources and the spatial group lengths. We do this, by dividing the resource capacities and the spatial group lengths by a factor $a > 1$, and by rounding up the new resource capacities and spatial group lengths. This implies a smaller number of different pos-

sible group lengths, and since we round up the resource requirements and resource capacities, the occupation of the spatial resources becomes tighter. The extra tightness reduces the resource relaxation that appeared when we created the problem instances considering 1 or 2 spatial resources.

Again, the test results show that all instances considering 6 groups could be solved within 100 seconds. The results of the problem instances considering 8 groups are presented in Table 2.

Table 2: Results for 8 groups with different resource scalings

# SR	Scaling	T_{min}	$T_{min}+$ $0.1 * \Delta$	$T_{min}+$ $0.25 * \Delta$	$T_{min}+$ $0.5 * \Delta$	T_{max}
$\Lambda = 1$	$a = 1$	100%	98.8%	97.5%	93.8%	93.8%
	$a = 2$	100%	96.3%	92.5%	88.8%	86.3%
	$a = 4$	96.3%	85%	77.5%	72.5%	67.5%
	$a = 10$	91.3%	65%	48.8%	42.5%	35%
	$a = 1000$	100%	100%	100%	100%	100%
$\Lambda = 2$	$a = 1$	100%	97.5%	96.3%	93.8%	92.5%
	$a = 2$	100%	98.8%	100%	96.3%	95%
	$a = 4$	98.8%	98.8%	96.3%	92.5%	91.3%
	$a = 10$	98.8%	95%	92.5%	91.3%	88.8%
	$a = 1000$	100%	100%	100%	100%	100%
$\Lambda = 4$	$a = 1$	100%	100%	97.5%	97.5%	97.5%
	$a = 2$	100%	98.8%	97.5%	97.5%	97.5%
	$a = 4$	100%	100%	98.8%	96.3%	96.3%
	$a = 10$	100%	100%	100%	97.5%	96.3%
	$a = 1000$	100%	100%	100%	100%	100%

If we observe the results for 4 spatial resources, then we see that scaling the resource capacities and requirements does not have much influence on the solver time. The last resourcescaling ($a = 1000$), shows what happens if all resource capacities and requirements are 1, i.e. when all allocations are known. If this is the case, then all instances can be solved within 100 seconds.

If we observe the results for 1 and 2 spatial resources, then we can see that the problem is more hard to solve, when the occupation of the resources becomes tighter. Now we can also see that the problem instances, where we consider just 1 spatial resource, are harder to solve than problem instances where we consider 2 spatial resources. We expected this to happen, since

more groups on 1 spatial resource might imply more possible group sequences, and therefore more feasible schedules. It seems that problem instances considering 8 groups can only be solved within reasonable time, if more then 1 spatial resource is considered. If we need to allocate 8 or more groups to 1 spatial resource, then the GSP probably cannot be solved anymore within reasonable time using the presented ILP model (unless the spatial resource capacity is 1).

The test results for 10 groups are presented in Table 3. Since the allocation of 8 groups to 1 spatial resource causes severe problems, we only present the results where 2 or 4 spatial resources are considered.

Table 3: Results for 10 groups with different resource scalings

# SR	Scaling	T_{min}	$T_{min}+$ $0.1 * \Delta$	$T_{min}+$ $0.25 * \Delta$	$T_{min}+$ $0.5 * \Delta$	T_{max}
$\Lambda = 2$	$a = 1$	92.5%	86.3%	76.3%	71.3%	68.8%
	$a = 2$	95%	86.3%	71.3%	65%	63.8%
	$a = 4$	88.8%	73.8%	57.5%	53.8%	50%
	$a = 10$	82.5%	53.8%	41.3%	35%	30%
	$a = 1000$	100%	100%	100%	100%	100%
$\Lambda = 4$	$a = 1$	98.8%	98.8%	97.5%	97.5%	96.3%
	$a = 2$	98.8%	98.8%	97.5%	96.3%	96.3%
	$a = 4$	97.5%	97.5%	98.8%	96.3%	95%
	$a = 10$	98.8%	95%	95%	90%	88.8%
	$a = 1000$	100%	100%	100%	100%	100%

We can make a similar observation as for instances considering 8 groups. Resourcescaling has not much effect (in this case even a slightly negative effect) on the instances considering 4 spatial resources. For the instances considering 2 spatial resources, we can see that the resource relaxation by the creation of these instances had a large effect on the solver time, and we can conclude that problem instances considering 2 spatial resources cannot be solved within reasonable time using the presented ILP, when 10 groups or more are considered.

6.4 Different scalings of release and due dates

The benchmark instances do not provide release dates and due dates for the activity groups. Therefore, the earliest start times and latest completion times of the groups, only depend on the precedence relations between the

groups. In order to test the influence of release and due dates on the solvability of the problem instances, we made the following adjustment to the problem instances.

For every group, we have a certain time window $[ESG_g, LCG_g]$, which defines the schedule period of the group. Since we schedule over a given time horizon, the scheduling problem is symmetric, and we can measure the influence of release and due dates by only adjusting the earliest start times. In order to keep feasibility of the problem instances, we may increase the earliest start times of the groups to at most $LCG_g - D\min_g$. More precisely, we increase the earliest start time of every group by $a*(LCG_g - ESG_g - D\min_g)$, where $a \in [0, 1]$. The results for the problem instances considering 8 groups are presented in Table 4.

Table 4: Results for 8 groups with different earliest start time adjustments

# SR	Scaling	T_{min}	$T_{min}+$ $0.1 * \Delta$	$T_{min}+$ $0.25 * \Delta$	$T_{min}+$ $0.5 * \Delta$	T_{max}
$\Lambda = 1$	$a = 0$	100%	98.8%	97.5%	93.8%	93.8%
	$a = 0.2$	100%	98.8%	98.8%	93.8%	93.8%
	$a = 0.4$	100%	98.8%	98.5%	96.3%	93.8%
	$a = 0.6$	100%	100%	98.8%	98.8%	93.8%
	$a = 0.8$	100%	100%	100%	98.8%	98.8%
	$a = 1$	100%	100%	100%	100%	100%
$\Lambda = 2$	$a = 0$	100%	97.5%	96.3%	93.8%	92.5%
	$a = 0.2$	100%	100%	97.5%	96.3%	92.5%
	$a = 0.4$	100%	100%	98.8%	93.8%	93.8%
	$a = 0.6$	100%	100%	98.8%	96.3%	95%
	$a = 0.8$	100%	100%	100%	98.8%	97.5%
	$a = 1$	100%	100%	100%	100%	100%
$\Lambda = 4$	$a = 0$	100%	98.8%	97.5%	97.5%	95%
	$a = 0.2$	100%	97.5%	96.3%	95%	95%
	$a = 0.4$	100%	100%	97.5%	97.5%	95%
	$a = 0.6$	100%	100%	100%	98.8%	95%
	$a = 0.8$	100%	100%	100%	98.8%	97.5%
	$a = 1$	100%	100%	100%	100%	100%

As we can see, enlarging the release dates makes the problem easier to solve. The schedule periods of the groups get reduced, and therefore the number of possible group sequences gets reduced. If we set $a = 1$, then the

group schedules are completely determined, i.e. $ESG_g = LSG_g$, and the GSP becomes only an allocation problem. In that case, the GSP is equivalent with the ARS-R problem as proposed by Duin and van Sluis [17]. Since adjusting the release dates of the groups has a positive influence on the solvability of the GSP, we present in Table 5 the results for instances considering 10 groups. Here we can see even more clearly the influence of release dates on the solvability of the GSP.

Table 5: Results for 10 groups with different earliest start time adjustments

# SR	Scaling	T_{min}	$T_{min}+$ $0.1 * \Delta$	$T_{min}+$ $0.25 * \Delta$	$T_{min}+$ $0.5 * \Delta$	T_{max}
$\Lambda = 1$	$a = 0$	95%	83.8%	72.5%	66.3%	65%
	$a = 0.2$	96.3%	87.5%	76.3%	68.8%	65%
	$a = 0.4$	97.5%	91.3%	81.3%	70%	66.3%
	$a = 0.6$	97.5%	95%	92.5%	80%	68.8%
	$a = 0.8$	97.5%	97.5%	93.8%	93.8%	83.8%
	$a = 1$	98.8%	98.8%	98.8%	98.8%	98.8%
$\Lambda = 2$	$a = 0$	90%	83.8%	75%	67.5%	65%
	$a = 0.2$	90%	82.5%	76.3%	70%	65%
	$a = 0.4$	91.3%	86.3%	76.3%	71.3%	65%
	$a = 0.6$	93.8%	91.3%	85%	76.3%	68.8%
	$a = 0.8$	98.8%	98.8%	95%	90%	76.3%
	$a = 1$	100%	100%	100%	100%	100%
$\Lambda = 4$	$a = 0$	98.8%	98.8%	98.8%	97.5%	96.3%
	$a = 0.2$	98.8%	98.8%	97.5%	96.3%	97.5%
	$a = 0.4$	98.8%	98.8%	98.8%	96.3%	96.3%
	$a = 0.6$	100%	100%	100%	98.8%	96.3%
	$a = 0.8$	100%	100%	100%	100%	98.8%
	$a = 1$	100%	100%	100%	100%	100%

6.5 Subproblems of GSP

If we fix the complete group schedule, then the problem gets reduced to an allocation problem. If we fix the whole group allocation, then we only need to schedule the groups. These are two subproblems of the GSP, which are both NP-hard as shown in Section 5.1. We also test these subproblems, in order to see upto what input sizes these problems can be solved within reasonable time. Therefore, we test the benchmark instances for resourcescaling

1000, and for the maximal release date adjustment, and we searched for the maximum number of groups, such that at least 95 per cent of the problem instances could be solved within 100 seconds. The results are shown in Table 6.

Table 6: Maximum number of groups for subproblems GSP

# SR	Fixed group schedules	Fixed allocations
$\Lambda = 1$	> 30 groups	9 groups
$\Lambda = 2$	> 30 groups	12 groups
$\Lambda = 4$	> 30 groups	15 groups

6.6 Conclusions

We tested the ILP model of the GSP for several problem instances. As we could see, it takes more time to solve the GSP, when the deadline is large. An explanation for this is that increasing the deadline might increase the number of possible group sequences, and therefore the number of feasible solutions.

Scaling the resource capacities and requirements also has a negative effect on the solver time. This is especially the case for the instances considering 1 or 2 spatial resources. The reason for this lays in the fact that the adjustments of the benchmark instances to instances where 1 or 2 spatial resources are considered, implies a relaxation of the resource constraints. The resource scaling also showed, that problem instances considering 1 spatial resource can be solved via the presented ILP model within reasonable time, if up to 7 or 8 groups are considered. For problem instances considering 2 spatial resources, maximally 9 or 10 groups can be considered in order to keep the solver time reasonable. Running a few more tests for problem instances concerning 4 spatial resources showed that these instances can be solved via the presented ILP model within reasonable time, if up to 11 or 12 groups are considered.

Adjustments of the release and due dates of the groups has more effect on the solver time. These adjustments reduce the schedule periods of the groups, which might imply that fewer group schedules are possible.

In the beginning of this section, we mentioned that phase 1 of the two-phase solution method forms the bottleneck of the GSP concerning the solver time. To see this, observe the following. In the second phase all group allocations are known. Furthermore, Table 1 and Table 6 show that the subproblem, in which all group allocations are known, can be solved within

reasonable time for larger input instances then the complete GSP. In phase two of the two-phase solution method, we add some groups to the model. However, the schedules for these extra groups are completely determined by the schedules of the original groups in the problem, which were also considered in phase 1. Therefore, the solver time of phase 2 is proportional to the solver time of the subproblem of the first phase, in which all allocations are known.

7 Conclusions and Recommendations

In this paper, we developed a group scheduling model which schedules and allocates activity groups to spatial resources. This model can be used as a pre-scheduling for the TCPSP with spatial resources. We derived an ILP formulation for the GSP, and we showed how the necessary input for the GSP can be extracted from the TCPSP. Furthermore, we proposed a strategy to translate the output of the GSP back to the TCPSP. Since the GSP is a relaxation of the TCPSP, it cannot predict in detail at which time buckets resource conflicts occur. Therefore, we proposed to fix the group allocation and sequences as in the output of the GSP, and not the exact start and completion times.

The GSP provides feasible schedules for the TCPSP with relaxed resource constraints. Furthermore, the GSP can be used to check feasibility of the TCPSP with respect to the spatial resources. The choice of the objective function for the GSP, has a large influence on the output of the GSP, and therefore on the feasible schedules for the complete TCPSP. We discussed a number of objective functions, which might lead to good solution possibilities for the TCPSP. Since these objectives showed better results for the allocation of the groups, than the sequencing of the groups, we proposed a two-phase solution method, where the first phase is concentrated on the allocation of the groups, and the second phase is concentrated on the sequencing of the groups.

We showed that the GSP is NP-hard. Even the subproblems of the GSP, where all group allocations are known or all group schedules are known, are NP-hard. However, since realistic input sizes for the GSP are not very large, we tried to solve the GSP to optimality. Test results showed that problem instances considering 1, 2 or 4 spatial resources could be solved via the presented ILP model within reasonable time, if up to 8, 10 or 12 groups, respectively, are considered. For the subproblem of the GSP, where all group allocations are known, problem instances where 1, 2 or 4 spatial resources are considered can be solved via the presented ILP model within reasonable time, if up to 9, 12 or 15 groups, respectively, are considered. For the subproblem of the GSP, where all group schedules are known, instances where 1, 2 or 4 spatial resources are considered can all be solved via the presented ILP model within reasonable time if up to 30 groups are considered (or possibly even more). Furthermore, increasing the deadline has a negative effect on the solver time, while increasing (decreasing) the release (due) dates has a positive effect on the solver time.

In Section 2.3, we stated a few objectives for this paper. Some of these objectives have been reached, some of them require further research. Therefore,

we recommend the following for further research.

- The GSP has been solved to optimality within reasonable time for small input sizes. Therefore, we recommend further research in the size of realistic problem instances for the GSP. If these input sizes are too large, solving the GSP to optimality requires too much time. Therefore, we recommend further research in heuristical procedures to solve the GSP.

- We proposed to translate the output of the GSP back to the TCPSP, by fixing the group allocations and sequences. We also proposed several objective functions for the group scheduling model. The combination of the objective function and the way in which the output of the GSP gets translated back to the TCPSP, has a very large influence on the solution possibilities for the remaining TCPSP. Therefore, we recommend further testings to see which objective function for the GSP, and which translation of the output of the GSP to the TCPSP, provides high quality solutions with respect to solving the TCPSP.

- We derived the GSP, as a pre-scheduling for the TCPSP. Within the TCPSP, the timing constraints are strong and the resource constraints are weak. Therefore, relaxing the resource constraints in the TCPSP, in order to derive the input for the GSP, does not affect the feasibility of the TCPSP. This observation does not hold for the RCPSP (in the RCPSP, the resource constraints are strong). We recommend further research to see if the GSP can also be used as a pre-scheduling for the RCPSP with spatial resources.

References

[1] AIMMS. The modelling system. www.aimms.com/.

[2] B.S. Baker, E.G. Coffman, and R.L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9:846–855, 1980.

[3] Joaquin Bautista, Raúl Suárez, Manuel Mateo, and Ramón Companys. Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 2404–2409, 2000.

[4] Ilker Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8):909–932, 1986.

[5] J.O. Berkey and P.Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429, 1987.

[6] J.J. Bisschop and M. Roelofs. *AIMMS: The Language Reference*. Paragon Decision Technology BV, Haarlem, The Netherlands, 1999.

[7] J.J. Bisschop and M. Roelofs. *AIMMS: The User's Guide*. Paragon Decision Technology BV, Haarlem, The Netherlands, 1999.

[8] P. Brucker, A. Drexel, R.H. Mohring, K. Neumann, and E. Pesch. Resource-constraint project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[9] P. Brucker and S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149:302–313, 2003.

[10] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.

[11] J.H. Cho and Y.D. Kim. A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, 48(7):736–744, 1997.

[12] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packing - a survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Company, Boston, 1997.

[13] E.W. Davis. Project scheduling under resource constraints: Historical review and categorization of procedures. *AIIE Transactions*, 5(4):297–313, 1973.

[14] Ronald de Boer. *Resource-Constrained Multi-Project Management-A Hierarchical Decision Support System*. PhD thesis, University of Twente, 1998.

[15] Erik L. Demeulemeester and Willy S. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492, 1997.

[16] Ulrich Dorndorf, Erwin Pesch, and Toan Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46(10):1365–1384, 2000.

[17] C. W. Duin and E. Van Sluis. On the complexity of adjacent resource scheduling. *Journal of Scheduling*, 9(1):49–62, 2006.

[18] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. *Proceedings of the IEEE 1992 Int. Conference on Robotics and Automation*, pages 1186–1192, 1992.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[20] Tom Guldemond. Time-constrained project scheduling. Master's thesis, University of Twente, 2005.

[21] W. Herroelen and E. Demeulemeester. *Recent Advances in Branch-and-Bound Procedures for Resource-Constrained Project Scheduling Problems*, chapter 12, pages 259–276. Wiley, 1995.

[22] K. Hess and R. Kolisch. Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints. *International Journal of Production Research*, 38(1):207–228, 2000.

[23] R. Kolisch. Integrated scheduling, assembly area-, and part-assignment for large scale make-to-order assemblies. *International Journal of Production Economics*, 64:127–141, 2000.

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

[25] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141:241–252, 2002.

[26] Andrea Lodi, Silvano Martello, and Daniele Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123:379–396, 2002.

[27] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[28] P.R. McMullen and G.V. Frazier. Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. *International Journal of Production Research*, 36(10):2717–2741, 1998.

[29] K. Neumann and C. Schwindt. Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *Operation Research Spektrum*, 19(3):205–217, 1997.

[30] K. Neumann, C. Schwindt, and J. Zimmermann. *Project scheduling with time windows and scarce resources*. Springer, Berlin Heidelberg New York, 2002.

[31] Wilhelmus Petronella Maria Nuijten. *Time and resource constrained scheduling : a constraint satisfaction approach*. PhD thesis, Technical University of Eindhoven, 1994.

[32] Michael Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, Upper Saddle River, New Jersey, 2002.

[33] PSPLIB. Project scheduling problem library kiel. http://129.187.106.231/psplib/.

[34] E. Remila. Approximate strip packing. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 31, Washington, DC, USA, 1996. IEEE Computer Society.

[35] M.E. Salveson. The assembly line balancing problem. *Journal of Industrial Engineering*, 6:18–25, 1955.

[36] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

[37] Joel P. Stinson, Edward W. Davis, and Basheer M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10(3):252–259, 1978.

[38] Sunset Software Technology. Optimization tools. www.sunsetsoft.com/.

Appendices

Appendix 1: ILP-formulation of GSP

$$\sum_t gs_{gt} = 1 \qquad \forall g = 1, ..., I$$

$$z_{gt} \leq z_{gt-1} + gs_{gt} \qquad \forall g = 1, ..., I, t = 2, ..., T$$

$$z_{g1} = gs_{g1} \qquad \forall g = 1, ..., I$$

$$\sum_{l=1}^{L_{\lambda(g)}} b_{gl} = 1 \qquad \forall g = 1, ..., I$$

$$y_{gl} \leq y_{gl-1} + b_{gl} \qquad \forall g = 1, ..., I, l = 2, ..., L_{\lambda(g)}$$

$$y_{g1} = b_{g1} \qquad \forall g = 1, ..., I$$

$$\sum_{l=1}^{L_{\lambda(g)}} y_{gl} \geq l_g \qquad \forall g = 1, ..., I$$

$$y_{gl} + y_{hl} \leq 1 + w_{gh} \qquad \forall g = 1, ..., I, h = 1, ..., I, g \neq h, \lambda(g) = \lambda(h), l = 1, ..., L_{\lambda(g)}$$

$$z_{gt} + z_{ht} \leq 1 + (1 - w_{gh}) \qquad \forall g = 1, ..., I, h = 1, ..., I, g \neq h, t = 1, ..., T$$

$$SG_g = \sum_t gs_{gt} * t \qquad \forall g = 1, ..., I$$

$$CG_g = \sum_t (z_{gt} - z_{gt+1} + gs_{gt+1}) * t \qquad \forall g = 1, ..., I$$

$$D_g = CG_g - SG_g + 1 \qquad \forall g = 1, ..., I$$

$$D_g \geq D\min_g \qquad \forall g = 1, ..., I$$

$$SG_g + TL_{gh} \leq CG_h \qquad \forall h = 1, ..., I, G_g \in P_h$$

$$gs_{gt}, z_{gt} \in \{0, 1\} \qquad \forall g = 1, ..., I, t = 1, ..., T$$

$$b_{gl}, y_{gl} \in \{0, 1\} \qquad \forall g = 1, ..., I, l = 1, ..., L_{\lambda(g)}$$

$$w_{gh} \in \{0, 1\} \qquad \forall g = 1, ..., I, h = 1, ..., I$$

$$SG_g \in \{ESG_g, LSG_g\} \qquad \forall g = 1, ..., I$$

$$CG_g \in \{ECG_g, LCG_g\} \qquad \forall g = 1, ..., I$$

$$D_g \in \{0, T\} \qquad \forall g = 1, ..., I$$

Appendix 2: Improved ILP-formulation of GSP

$$
\begin{aligned}
D\min_g &\le CG_g - SG_g && \forall g = 1,...,I \\
l_g &= CL_g - SL_g && \forall g = 1,...,I \\
SG_g &\le CG_h - TL_{gh} && \forall h = 1,...,I, G_g \in P_h \\
w_{gh} &\ge \frac{\left(CL_h - SL_g - L_{\lambda(g)} * y_{gh}\right)}{L_{\lambda(g)}} && \forall g = 1,...,I, h = 1,...,I, \lambda(g) = \lambda(h), \\
& && g \neq h, ESG_g \le LCG_h, ESG_h \le LCG_g \\
w_{gh} &\ge \frac{\left(CL_g - SL_h - L_{\lambda(g)} * (1 - y_{gh})\right)}{L_{\lambda(g)}} && \forall g = 1,...,I, h = 1,...,I, \lambda(g) = \lambda(h), \\
& && g \neq h, ESG_g \le LCG_h, ESG_h \le LCG_g \\
SG_g &\ge CG_h - T * (1 + v_{gh} - w_{gh}) && \forall g = 1,...,I, h = 1,...,I, \\
& && g \neq h, \lambda(g) = \lambda(h) \\
SG_h &\ge CG_g - T * (2 - v_{gh} - w_{gh}) && \forall g = 1,...,I, h = 1,...,I, \\
& && g \neq h, \lambda(g) = \lambda(h) \\
w_{gh} &\in \{0,1\} && \forall \lambda = 1,...,\Lambda, g \neq h, \lambda(g) = \lambda(h), \\
& && ESG_g \le LCG_h, ESG_h \le LCG_g \\
v_{gh} &\in \{0,1\} && \forall \lambda = 1,...,\Lambda, g \neq h, \lambda(g) = \lambda(h), \\
& && ESG_g \le LCG_h, ESG_h \le LCG_g \\
y_{gh} &\in \{0,1\} && \forall \lambda = 1,...,\Lambda, g \neq h, \lambda(g) = \lambda(h), \\
& && ESG_g \le LCG_h, ESG_h \le LCG_g \\
SG_g &\in \{ESG_g, LSG_g\} && \forall g = 1,...,I \\
CG_g &\in \{ECG_g, LCG_g\} && \forall g = 1,...,I \\
SL_g &\in \left\{0, L_{\lambda(g)} - l_g\right\} && \forall g = 1,...,I \\
CL_g &\in \left\{l_g, L_{\lambda(g)}\right\} && \forall g = 1,...,I
\end{aligned}
$$