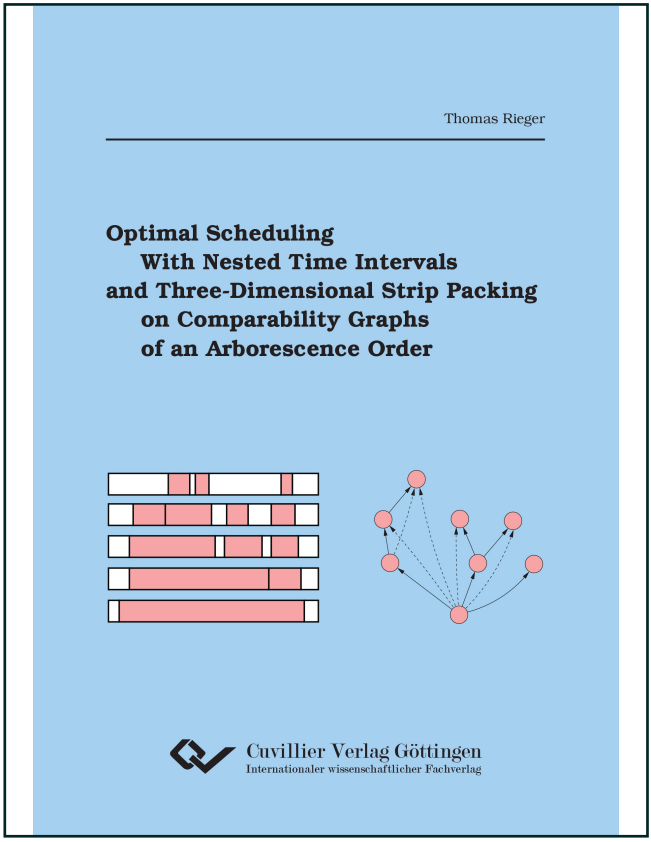
Thomas Rieger (Autor)

Optimal Scheduling with Nested Time Intervals and Three-Dimensional Strip Packing on Compara-bility Graphs of an Arborescence Order



https://cuvillier.de/de/shop/publications/7014

1 Introduction

REAL-world problems, such as the optimization of production processes of companies, often require a decision-making w.r.t. different conflicting objectives and predefined framework conditions. These conditions can be caused by physical characteristics of manufactured products or arise in the context of a production site environment. In business reality, practical problems are often handled by long-established experts based on their experience and knowledge of required performance and quality. Unfortunately, if the practical problems are quite complex even with this information it will prove difficult to find satisfactory practical solutions. In this case the decision-maker will be grateful to receive helpful support and advices by scientific knowledge. In the realm of mathematical optimization, one such decision-making process is known as scheduling. Applying scheduling and combinatorial optimization in industry frequently results in cost savings for the company. On the other hand, some practical applications are founded on theoretical problems which may in turn be relevant in the field of mathematical optimization.

This thesis is concerned with scheduling problems that arise in the practical problem of rail car maintenance. In reality, rail cars are serviced in regular time intervals in service halls of a maintenance company. Depending on the degree of wear, different types of necessary maintenance steps need to be performed one after another and without interruption in a predefined technological order for each rail car. A typical service hall contains several parallel rail tracks, each of which is equipped with several machines performing the same type of maintenance step. However, in some service halls there exists only a single gate for each rail track and a dead end at the opposite side of this gate. Consequently, the maintenance operator has to devise a schedule s.t. each rail car can be routed along the tracks whenever one of its maintenance steps is finished, avoiding collisions with other currently serviced rail cars, until the maintenance work for all rail cars is finished.

Motivated by this application we introduce a new type of job characteristic to the world of scheduling in the first part of this thesis, and investigate its impact on already known scheduling problems. In the second part of this thesis we further consider a particular three-dimensional strip packing problem. The connection between both problems and hence both parts of this thesis is due to the fact that they are based on the same type of restriction.

Outline of the Thesis. As previously said, this thesis is mainly divided into two parts, a first part that is dedicated to scheduling and a second strip packing part. At the beginning of each part we provide a brief overview of its contents. However, in Chapter 2 we start with summarizing basic notation and definitions, and moreover provide a theoretical foundation needed for understanding both parts. In particular, we introduce the term nested intervals that is the essential requirement for the central mathematical problems considered in this thesis. Moreover, we also present graph classes as well as some combinatorial optimization problems together with known results and basic notions for these problems that are relevant in this thesis.

Chapter 3 is devoted to scheduling. We first briefly introduce the framework and notation used within this chapter. Based on the motivating application in rail car maintenance, we explicitly formulate the new resulting scheduling problems, namely two variants of a flexible job shop with work centers and nested time intervals of jobs as well as the makespan objective. The following section of this chapter is dedicated to classifying the computational complexity of both variants. Beside $\mathcal{NP}$-hardness results for both scheduling variants we further present an approximation algorithm for one variant restricted to a single work center, i. e. to a parallel machine environment. Afterwards, we introduce a linear MIP model formulation for the flexible job shop variants and list some lower bounds for the minimum makespan. For generating feasible schedules we present two heuristical approaches, namely a greedy method as well as a shifting bottleneck heuristical method. In addition, we also deal with an exact branch&bound approach that is based on an activity network tailored to the scheduling problems considered in this section. For all approaches of this section, we finally discuss computational results for input data associated with the application in rail car maintenance. At the end of this chapter, we further consider the alternative objective of minimizing the total completion time instead of makespan minimization as brief excursion.

In chapter 4, we transfer the new type of restriction that is based on nested intervals and initially formulated for scheduling problems in the preceding chapter to strip packing problems. Instead of considering time intervals associated with jobs, in this chapter intervals are used in the context of the positioning of boxes. The resulting problem is a strip packing problem on comparability graphs of an arborescence order. In the first section, we start with an introduction to the framework of standard strip packing problems and introduce notation used in the remainder of this chapter. In the second section of this chapter, we finally consider strip packing problems that are based on comparability graphs of an arborescence order and introduce two mathematical formulations that are transferred from both formulations introduced for standard packing problems before. Moreover, we also include gravity constraints, present a preprocessing method, an MIP model formulation as well as a heuristical method for strip packing problems on comparability graphs of an arborescence order. Finally, we discuss computational results for these methods.

Acknowledgements. Without the support and encouragement of many people this thesis would not have been completed. It is my inner need to express my sincere gratitude to these persons in the following. First of all, I would like to thank my supervisor Uwe Zimmermann for giving me the chance to become a member of MO at TU Braunschweig,

for your confidence and patience in me, giving me freedom in my research interests, and for providing me constructive and helpful support throughout the process of writing up this thesis. I finally made it. Furthermore, many thanks to Sigrid Knust who instantly agreed to be co-referee for this thesis. I would like to thank my colleagues Andreas Tillmann and Frederik Fiand for motivating and valuable discussions in the context of this thesis and for your support in the field of teaching at TU. Many thanks to Silke Thiel for your never ending intention of creating a pleasant working environment. An extra special thank goes to Ronny Hansmann for giving me lots of essential advice in the context of this thesis and scientific work in general and, again, to Andreas Tillmann for untiring support in the finalization process of this thesis. Last but not least and from the bottom of my heart, I would like to express my heartfelt thanks to my family. Thanks for your support and for always motivating me.

And finally, thank you for reading this thesis.

2 Preliminaries

In this chapter, we briefly introduce the framework, definitions and notation of the fields Complexity Theory, Graph Theory and Discrete Optimization that are relevant in the remainder of this thesis. For an easier reading, we postpone definitions and notation that are only regarded either in the context of scheduling or packing problems to the respective beginning of Chapters 3 and 4 of this thesis. However, we already introduce a general definition of the term nested intervals in Subsection 2.3 that is overall connecting both parts of this thesis.

For further reading, we refer to the well-known literature of Garey & Johnson (1979) for Complexity Theory, to Brandstädt et al. (1999) and particularly to Golumbic (2004) – that is also highly relevant for chapter 4 of this thesis – for the field of Graph Theory and finally to Schrijver (2003) and Korte & Vygen (2012) for Discrete Optimization.

2.1 Mathematical Problems and Their Complexity

A *mathematical problem* $\mathfrak{P}$ is described by all of its instances $\mathfrak{I} \in \mathfrak{P}$. We distinguish between two types of problems, namely decision problems and optimization problems. In a *decision problem* a question that only allows a "yes" or a "no" answer is asked w.r.t. a regarded instance $\mathfrak{I}$ used for specifying the problem. We say that an *algorithm solves* such *a decision problem* $\mathfrak{P}$ if it answers the question for every $\mathfrak{I} \in \mathfrak{P}$, i. e. if it always finds the correct one of both possible outcomes. In contrast to this, an instance of an *optimization problem* is fully described by a solution space X, that is, a *set of solutions* of problem $\mathfrak{P}$, and an *objective function* $f : X \to \mathbb{R}$ that associates a weight $f(x)$ with each solution $x \in X$. Based on the objective function f, optimization problems are commonly divided into the classes of *minimization problems* and *maximization problems*, depending on whether function f has to be minimized or to be maximized. Consequently, the possible outcomes of an *algorithm* that *solves* a minimization problem $\mathfrak{P}$ are *infeasibility* if $X = \varnothing$, *unboundedness* if for every $r \in \mathbb{R}$ there is some $x \in X$ s.t. $f(x) < r$, or an *optimal solution* $x^* \in X$ with *optimal objective value* $f(x^*) = \min_{x \in X} f(x)$ for each instance $\mathfrak{I} \in \mathfrak{P}$. Analogously, we may also define algorithms that solve maximization problems.

Even though we distinguish between decision problems and optimization problems at the beginning of this subsection, there is an important dependence between both kinds

of problems that is also used in later methods. In particular, we may define a threshold $k \in \mathbb{R}$ for the objective function of a minimization problem and formulate an associated decision problem:

$$\text{Does there exist an } x \in X \text{ satisfying } f(x) \leq k?$$

One of the most important issues of Complexity Theory is to measure the performance of an algorithm developed for solving a problem w.r.t. its time complexity. Since this time also depends on external factors like CPU-power etc. we are interested in a measure that is strictly based on the input instance $\mathfrak{I}$ of a problem. More precisely, for each input instance $\mathfrak{I}$ of a problem $\mathfrak{P}$ we define the *input length* $|\mathfrak{I}| = n$ as length that is used by some encoding to store the data defining instance $\mathfrak{I}$. The *time complexity* of an algorithm for solving a problem $\mathfrak{P}$ is $\mathcal{O}(g(|\mathfrak{I}|))$, if $k \cdot g(\mathfrak{I})$ is an upper bound on the total number of steps (performed elementary operations) the algorithm needs at most for solving any input instance $\mathfrak{I}$ of a problem $\mathfrak{P}$ for fixed input length $|\mathfrak{I}|$, and some function $g : \mathbb{N} \to \mathbb{R}$ as well as a constant $k \in \mathbb{N}$. Clearly, when asking for the time complexity of an algorithm we are only interested in the tightest estimation, that is, the smallest upper bound.
In particular, an algorithm solves a problem in *polynomial time* if the time complexity of the algorithm can be bounded by a polynomial $p(|\mathfrak{I}|)$, i. e. if the time complexity is $\mathcal{O}(|\mathfrak{I}|^k)$ for some constant $k \in \mathbb{N}$. If there exists an algorithm that solves a decision problem $\mathfrak{P}$ in polynomial time then $\mathfrak{P}$ is denoted as *polynomially solvable problem*. The class of all polynomially solvable problems is denoted by $\mathcal{P}$.

However, the notion polynomially solvable problems depends on the encoding of the input, and we assume that the numerical input data is binary encoded. For example, if an algorithm has time complexity $\sum_{j=1}^{n} w_j$ for some given numeric input data $w_1, \ldots, w_n$ then the algorithm is not polynomially bounded from above. In fact, based on a binary encoding, in this case the time complexity is an exponentially growing function of the length of an input string. An algorithm is called *pseudopolynomial* if its time complexity is polynomial in the numeric value of the input, but possibly exponential in the length of the input.

In the following, we introduce another class of decision problems that is denoted by $\mathcal{NP}$. For problems $\mathfrak{P}$ contained in this class, we do not call for the existence of a polynomial-time algorithm for solving $\mathfrak{P}$, but we require that for each yes-instance there is a certificate which can be checked in polynomial time. A binary string $\mathfrak{C}$ of length polynomially bounded in the size of the instance $\mathfrak{I}$ is called *yes certificate for* $\mathfrak{I}$ if it clarifies that the answer for instance $\mathfrak{I}$ is "yes". Consequently, a decision problem is contained in the class $\mathcal{NP}$ if there is a *certificate-checking algorithm* which, for given instance $\mathfrak{I}$ and string $\mathfrak{C}$, answers in polynomial time in the size of $\mathfrak{I}$ whether $\mathfrak{C}$ is a yes certificate for $\mathfrak{I}$.

For both complexity classes $\mathcal{P}$, $\mathcal{NP}$ it is easily seen that $\mathcal{P} \subseteq \mathcal{NP}$. However, it is currently not known if there exists a decision problem in $\mathcal{NP}$ which is definitely not solvable in polynomial time, and hence not contained in $\mathcal{P}$. Proving $\mathcal{P} \neq \mathcal{NP}$, or otherwise $\mathcal{P} = \mathcal{NP}$, would answer one of the probably most interesting open question in mathematical optimization.

A decision problem $\mathfrak{P}_1$ is said to be *polynomially (Karp-) reducible* to another decision problem $\mathfrak{P}_2$, if there exists a polynomial-time algorithm which constructs an instance $\mathfrak{I}_2$

of $\mathfrak{P}_2$ for every instance $\mathfrak{I}_1$ of $\mathfrak{P}_1$ s.t. $\mathfrak{I}_1$ has "yes"-answer if and only if $\mathfrak{I}_2$ has a "yes"-answer. In this case, we also write $\mathfrak{P}_1 \propto \mathfrak{P}_2$. We say that a decision problem $\mathfrak{P}_1$ is *$\mathcal{NP}$-hard* if $\mathfrak{P}_2 \propto \mathfrak{P}_1$ for every decision problem $\mathfrak{P}_2$ contained in class $\mathcal{NP}$. If additionally $\mathfrak{P}_1 \in \mathcal{NP}$ then decision problem $\mathfrak{P}_1$ is said to be *$\mathcal{NP}$-complete*. In order to prove that a decision problem $\mathfrak{P}_1$ is contained in class $\mathcal{P}$ it is sufficient to verify $\mathfrak{P}_1 \propto \mathfrak{P}_2$ for some $\mathfrak{P}_2 \in \mathcal{P}$. On the other hand, we can prove that a decision problem $\mathfrak{P}_1$ is $\mathcal{NP}$-complete by proving that both $\mathfrak{P}_1$ belongs to $\mathcal{NP}$ as well as $\mathfrak{P}_2 \propto \mathfrak{P}_1$ for some $\mathcal{NP}$-complete decision problem $\mathfrak{P}_2$. Consequently, if some $\mathcal{NP}$-complete decision problem would be proven to be solvable in polynomial time, then this would imply $\mathcal{P} = \mathcal{NP}$.

Based on the existence of a pseudopolynomial algorithm we further classify $\mathcal{NP}$-complete problems. More precisely, an $\mathcal{NP}$-complete problem for which a pseudopolynomial algorithm exists is said to be *weakly $\mathcal{NP}$-complete*. If, in contrast, it is proven that a pseudopolynomial algorithm cannot exist for an $\mathcal{NP}$-complete problem unless $\mathcal{P} = \mathcal{NP}$ then we speak of a *strongly $\mathcal{NP}$-complete problem*.

The preceding concept for classifying decision problems can also be carried over to optimization problems. More precisely, an optimization problem $\mathfrak{P}$ is said to be $\mathcal{NP}$-hard if it is associated with an $\mathcal{NP}$-hard decision problem as described at the beginning of this subsection. If the corresponding decision problem is weakly $\mathcal{NP}$-complete then the optimization problem is an *$\mathcal{NP}$-hard* problem *in the weak sense*, or analogously *strongly $\mathcal{NP}$-hard* if the decision problem is strongly $\mathcal{NP}$-complete.

Many combinatorial optimization problems can be formulated by the following *program in canonical form*

$$\max\{f(x) \mid Ax \leq b, x \geq 0, x \in \mathbb{N}_0^{n_I} \times \mathbb{R}^{n-n_I}\}$$

where $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $n_I \in \{0, 1, \ldots, n\}$ corresponds to the total number of integer variables of the program, and $f(x)$ is the objective function that only depends on variables $x \in \mathbb{N}_0^{n_I} \times \mathbb{R}^{n-n_I}$ as well as on constants $c \in \mathbb{Q}^n$.

If f is a *linear function* we also write $f(x) = c^T x$ and consider the following types of programs. If $n_I = 0$ then the program contains only continuous variables x and is therefore also denoted as LINEAR PROGRAM (LP). Conversely, for $n_I = n$ each variable is integer-valued and the program is said to be an INTEGER (LINEAR) PROGRAM (IP). For all $n_I \in \{1, \ldots, n-1\}$ the program contains n_I integer variables as well as $n - n_I$ continuous variables and is denoted as MIXED INTEGER (LINEAR) PROGRAMM (MIP).

Otherwise, if f is a *multilinear function* that can be written as $f(x) = \sum_{I \subseteq \{1,\ldots,n\}} c_I \prod_{i \in I} x_i$ and if further $n_I = 0$ then we speak of a MULTILINEAR PROGRAMM (MLP).

2.2 Algorithms and Methods for Solving Problems

For a wide variety of combinatorial optimization problems it is not difficult to construct an algorithm that generates some feasible solution of the problem in a relatively small amount of time. Such an algorithm that ensures a solution of the problem is determined when the algorithm terminates is denoted as *heuristical method*. However, the objective value of the resulting solution of a purely heuristical method may be far away from the

optimal objective value of the problem. If we are moreover able to both guarantee that the resulting solution is close to the optimal solution of the problem as well as a polynomial time complexity of the algorithm then a heuristical method is particularly denoted as approximation. In the following we will point out this definition more precisely.

Let $\mathfrak{A}$ be a polynomial-time algorithm that constructs a feasible solution for every instance $\mathfrak{I}$ of a minimization problem $\mathfrak{P}$. Furthermore, let $z_{\mathfrak{A}}(\mathfrak{I})$ denote the objective value generated by algorithm $\mathfrak{A}$, and let $z^*(\mathfrak{I})$ be the optimal objective value for instance $\mathfrak{I}$.

Then $\mathfrak{A}$ is an *absolute approximation algorithm* for $\mathfrak{P}$, if $|z_{\mathfrak{A}}(\mathfrak{I}) - z^*(\mathfrak{I})| \leq k$ is satisfied for every instance $\mathfrak{I} \in \mathfrak{P}$ and some constant $k \in \mathbb{R}_+$. Such an absolute approximation algorithm is only known for a few $\mathcal{NP}$-hard minimization problems which leads us to the following definition of an approximation algorithm w.r.t. its relative performance guarantee. For this purpose, we restrict to instances with non-negative optimal objective value $z^*(\mathfrak{I})$.

A polynomial-time algorithm $\mathfrak{A}$ is said to be a *approximation algorithm* with *approximative ratio* $a(|\mathfrak{I}|)$, if $z_{\mathfrak{A}}(\mathfrak{I}) \leq a(|\mathfrak{I}|) \cdot z^*(\mathfrak{I})$ is satisfied for every instance $\mathfrak{I} \in \mathfrak{P}$, where $a : \mathbb{N} \to \mathbb{R}_+$. If the performance ratio $a(|\mathfrak{I}|)$ is given by a constant function then we particularly speak of a *constant factor approximation algorithm*. If $\mathfrak{A}$ is an approximation algorithm for a problem $\mathfrak{P}$ then in this thesis we also say that $\mathfrak{A}$ *approximatively solves* $\mathfrak{P}$.

If we are interested in actually finding an optimal solution of a combinatorial optimization problem then we need so-called *exact methods*. A quite intuitive exact method for solving a minimization problem may be described as follows.

Based on the relation between a minimization problem $\mathfrak{P}$ and its associated decision problem as stated at the beginning of this section we are able to formulate a search procedure for solving an instance $\mathfrak{I}$ of problem $\mathfrak{P}$. In particular, if the optimal objective value $z^*(\mathfrak{I})$ can be bounded by some values $z_l^1, z_u^1 \in \mathbb{Z}$ in advance s.t. $z_l^1 \leq z^*(\mathfrak{I}) \leq z_u^1$ then a *binary search procedure* localizes the position of $z^*(\mathfrak{I})$ as follows. It consecutively solves an associated decision problem for threshold $\tilde{z} \coloneqq (z_u^t - z_l^t)/2$ in iteration t, and updates the currently known best bounds on $z^*(\mathfrak{I})$ depending on the outcome of the decision problem, that is, either $z_l^{t+1} \coloneqq \tilde{z}, z_u^{t+1} \coloneqq z_u^t$ or otherwise $z_l^{t+1} \coloneqq z_l^t, z_u^{t+1} \coloneqq \tilde{z}$. If $z^*(\mathfrak{I}) \in \mathbb{Z}$ then the total number of iterations of the procedure is hence bounded by $\mathcal{O}(\log(z_u^1 - z_l^1))$. Consequently, the time complexity of the resulting binary search algorithm heavily depends on the computational complexity of the decision problems that need to be solved within each iteration of the procedure.

An alternative approach, that is probably the most famous exact solution method in the mathematical optimization community, is the so-called *branch&bound* method. Since this approach is very popular we confine ourselves to briefly introduce the essential ingredients of this procedure applied to an instance $\mathfrak{I}$ of a minimization problem $\mathfrak{P}$ with set of feasible solutions X. As its name implies a branch&bound method is based on two main tools, namely a branching procedure as well as a bounding procedure.

Given a subset $\tilde{X} \subseteq X$ a *branching* procedure returns two or more strict subsets $X_1, X_2, \ldots$ of $\tilde{X}$ s.t. the union of all of these subsets covers $\tilde{X}$. A recursive application of this procedure – starting with set X – defines a search tree that is also denoted as *branch&bound tree* where each subset is associated with a *(branch&bound) node*. In the remainder of this thesis, each of these nodes is also said to be a *subproblem* of the initial problem associated with the root node in this tree. If the branching procedure particularly defines

subsets by fixing variables of an associated program then the selection of these variables is denoted as *branching policy*. If the selection of the variables of a branching policy moreover only depends on a predefined rule then this rule is said to be the *branching rule* of the method. The above-described branching procedure is linked with a so-called *node (subproblem) selection* policy that decides which node is chosen next for branching.

In the second basic tool of this method, the *bounding* procedure, we compute bounds on $\min\{f(x) \mid x \in \tilde{X}\}$, i. e. on the minimum objective value over a subset $\tilde{X}$ associated with a subproblem. Usually, an upper bound for a subproblem is generated by fast heuristical methods, e.g. a greedy heuristical method, whereas the lower bound is calculated by relaxing some constraints and/or a subset of integer variables of a program formulated for the subproblem. If all integer variables are relaxed then we particularly speak of a branch&bound method that is based on LP-*relaxations*. The key idea of branch&bound method is that a subproblem whose lower bound is at least as large as the objective value of the currently best found feasible solution, then the subproblem may be discarded from the branch&bound tree. This step is called *pruning*. Obviously, such a subproblem does not contain any feasible solution with objective value strictly smaller than the one of the currently best found feasible solution.

The efficiency of a branch&bound method strongly depends on the interaction of both described main tools. For instance, if the lower bounds computed for the subproblems are quite weak then the branch&bound tree increases in size. On the other hand, a relative high computation time for generating bounds overall increases the total computation time of the method.

2.3 Nested Intervals and Relations

An *interval* $\mathcal{I}_i$, associated with some index $i \in \mathbb{N}$, is defined by a *coordinate* $x_i \in \mathbb{Q}_+$ and a *width* $w_i \in \mathbb{Q}_+$ s.t. $\mathcal{I}_i = (x_i, x_i + w_i)$. Let $\mathcal{I}_i$, $\mathcal{I}_j$ be a distinct pair of intervals (on the line) given by coordinates $x_i, x_j \in \mathbb{Q}_+$ and widths $w_i, w_j \in \mathbb{Q}_+$. Then $\mathcal{I}_i$ is said to be *contained in* $\mathcal{I}_j$, denoted by $\mathcal{I}_i \subseteq \mathcal{I}_j$, if $x_j \leq x_i$ and $x_j + w_j \geq x_i + w_i$. If further $\mathcal{I}_i \neq \mathcal{I}_j$ then $\mathcal{I}_i$ is said to be *strictly contained in* $\mathcal{I}_j$, denoted by $\mathcal{I}_i \subset \mathcal{I}_j$.
Furthermore, the pair of intervals is defined to be

- *concurrent* denoted by $\mathcal{I}_i \parallel^c \mathcal{I}_j$,
  if either $\mathcal{I}_i$ is contained in $\mathcal{I}_j$ or alternatively $\mathcal{I}_j$ is contained in $\mathcal{I}_i$.

- *subsequent*,
  if both intervals do not intersect, that is, if $\mathcal{I}_i \cap \mathcal{I}_j = \varnothing$.

- *non-overlapping*,
  if $\mathcal{I}_i$ and $\mathcal{I}_j$ are either concurrent or subsequent.

Accordingly, if $\mathcal{I}$ is a family of intervals, $|\mathcal{I}| \geq 2$, then we denote $\mathcal{I}$ as *family of concurrent / subsequent intervals* if $\mathcal{I}_i$ and $\mathcal{I}_j$ are concurrent / subsequent for every pair of distinct intervals $\mathcal{I}_i, \mathcal{I}_j \in \mathcal{I}$. Quite important in the context of this thesis, we define the

more general family of nested intervals as follows. *A family of intervals* $\mathcal{I}$ is said to be *nested* if every pair of distinct intervals $\mathcal{I}_i, \mathcal{I}_j \in \mathcal{I}$ is non-overlapping.

Note that in the literature a family of nested intervals is sometimes defined equivalently to our definition of a family of concurrent intervals. For instance in Fridy (2000) this definition is used in the context of the so-called Nested Intervals Theorem. In this thesis, however, each pair of intervals contained in a family of nested intervals is also allowed to be subsequent.

More generally and not exclusively formulated for pairs of intervals, let S be a set and let $\mathfrak{R} \subseteq S \times S$ be a binary relation on S. Then $\mathfrak{R}$ is defined to be

(i) *reflexive* on S if $(x,x) \in \mathfrak{R}$ for all $x \in S$,

(ii) *transitive* on S if for each triple of elements $x, y, z \in S$:

$$(x,y) \in \mathfrak{R} \wedge (y,z) \in \mathfrak{R} \Rightarrow (x,z) \in \mathfrak{R},$$

(iii) *antisymmetric* on S if for all $x, y \in S$ with $(x,y), (y,x) \in \mathfrak{R}$ we have $x = y$.

If $\mathfrak{R}$ satisfies $(i) - (iii)$ for given set S then $\mathfrak{R}$ is defined as *partial order* of S and the pair $(S, \mathfrak{R})$ as *partially ordered set (poset)*.

2.4 Graphs, Relevant Graph Classes and Further Notions

Let V denote a finite set of *vertices*. A *graph* G is given by a pair $G = (V,E)$ where $E \subseteq V \times V$ denotes the set of edges of G. An *edge* e corresponds to a pair $e = e_{ij} = (v_i, v_j)$ of vertices $v_i, v_j \in V$, $i \neq j$. We say two vertices $v_i, v_j \in V$ are *adjacent* in G if there is an edge $e = (v_i, v_j)$ in E. Moreover, vertex $v_i \in V$ / edge $e \in E$ is said to be *incident* to an edge $e \in E$ / a vertex $v_i \in V$ if $e = e_{ij}$ for some vertex $v_j \in V$. If the pair of vertices associated with edge e is ordered we define e to be a *directed* edge. In this case, e is called *arc* in the remainder of this thesis and denoted by $a_{ij} = (v_i, v_j)$ where indices i and j indicate the order of the pair of vertices v_i, v_j. Otherwise, i. e. if $e = (v_i, v_j) = (v_j, v_i)$ an edge is defined to be *undirected*. If set E contains only edges we denote the graph to be *undirected*. Analogously, if a graph contains only arcs we define the graph to be *directed* or equivalently as a *digraph*. In this case, we denote the digraph by $D = (V,A)$ where A is defined as the set of arcs of D in order to guarantee a separate notation of graphs and digraphs. A pair of graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ / digraphs $D_1 = (V_1, A_1)$, $D_2 = (V_2, A_2)$ is said to be *isomorphic* if there exists a bijective function $f : V_1 \to V_2$ s.t. $(v_i, v_j) \in E_1$ / $(v_i, v_j) \in A_1$ if and only if $(v_{f(i)}, v_{f(j)}) \in E_2$ / $(v_{f(i)}, v_{f(j)}) \in A_2$. In a directed / undirected *multigraph* more than one edge / arc may exist for a single pair of vertices $v_i, v_j \in V$. Moreover, a *loop* is defined by an edge / arc (v,v), $v \in V$. In this thesis, we do not consider multigraphs and loops either for graphs or digraphs. If for a given graph $G = (V,E)$ the set of vertices V is associated with some weight given by a function $w : V \to \mathbb{Q}$, we add w to the pair (V,E), that is, G is then denoted by the triple $G = (V,E,w)$. If otherwise each arc of a digraph is associated with some weight given by a function $w : A \to \mathbb{Q}$ then we also speak of a *network* N instead of a weighted digraph and write $N = (V,A,w)$.