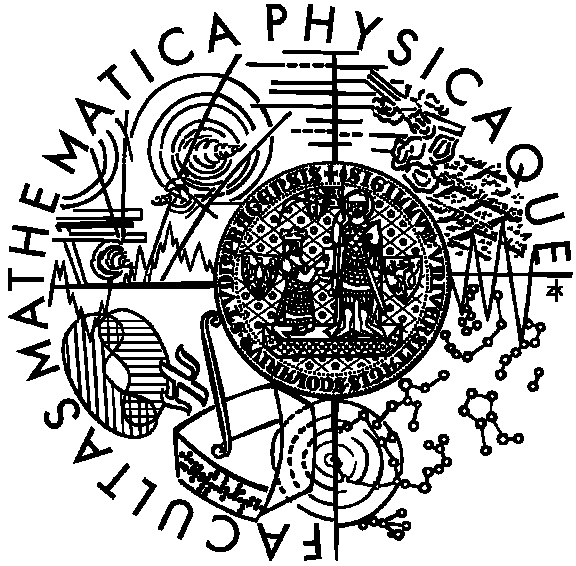
Charles University in Prague
Faculty of Mathematics and Physics

Master Thesis



Nguyen Son Tung

Constrained activity sequencing

Department of Theoretical Computer Science and Mathematical Logic
Supervisor of the Thesis: Doc. RNDr. Roman Barták, Ph.D.
Study Programme: Computer Science, Software systems

Prague, 2010

Acknowledgement

I would like to express my deep and sincere gratitude towards professor Doc. RNDr. Roman Barták, Ph.D. for his valuable advice and helpful attitude that led me to finish my Thesis.

I also would like to very thank RNDr. Pham Huu Uyen, CSc. and company The Kite, s.r.o, for the inspiration to the problem, helpfulness and support.

I hereby declare that I wrote the Thesis myself using only the referenced sources. I agree with lending the Thesis.

In Prague, July 6th 2010. Nguyen Son Tung

Contents

Název práce: Constraint activity sequencing
Autor: Nguyen Son Tung
Katedra: Katedra teoretické informatiky a matematické logiky
Vedoucí diplomové práce: Doc. RNDr. Roman Barták, Ph.D.
E-mail vedoucího: bartak@ktiml.mff.cuni.cz

Abstrakt: Mnoho rozvrhovacích problémů lze považovat za problémy s hledáním posloupností aktivit splňujících určité podmínky a omezení. Typickým příkladem je rozvrhování letů v leteckém průmyslu, kde úlohou je přiřazení segmentů letů a servisních aktivit k jednotlivým letadlům. Tato diplomová práce se zaměřuje na hledání takovýchto omezených posloupností aktivit. Cílem práce je navrhnout formální model problému, včetně přesné specifikace podmínek a účelové funkce, schopný najít (sub)optimální posloupnosti aktivit. Navrhovaný model je založen na technikách Programování s omezujícími podmínkami.

Klíčová slova: Rotation assignment, Airline industry, Optimization, Constraint Programming

Title: Constraint activity sequencing
Author: Nguyen Son Tung
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Doc. RNDr. Roman Barták, Ph.D.
Supervisor's e-mail address: bartak@ktiml.mff.cuni.cz

Abstract: Many scheduling problems can be seen as activity sequencing problems, where the activity sequence in demand satisfies certain constraints. A typical example is scheduling in the airline industry where the task is to assign to each aircraft a segment of flight and non-flight activities while guaranteeing certain required properties. This diploma thesis deals with such type of constrained activity sequencing. The aim is to propose a formal model of the problem, including specification of all constraints and objectives, capable of finding (near)optimal sequences of activities. The proposed model is based on Constraint programming techniques.

Keywords: Rotation assignment, Airline industry, Optimization, Constraint Programming

Chapter 1

Introduction

The topic of this thesis deals with one of the most important components of the airlines planning process - the Rotation assignment problem. More specifically, it deals with the problem of determining which aircrafts should operate which flight and non-flight activities (a sequence of such activities is called a $rotation$) while respecting that all the activities must be operated by exactly one aircraft and other certain numbers of additional rules must hold. Also, in the optimization version of the problem, such assignment of the activities to aircrafts should be done in a way that minimizes the cost associated with operating the aircrafts and violations of certain rules.

The rotation assignment problem has been very extensively investigated over years in the field of Operation research, as utilizing aircraft use is one of the most important concerns of the airlines industry. Every change, every improvement over utilization of aircrafts can have a huge impact on costs and profits for an airliner.

Many solution approaches have been proposed for many variations of the problem. Most of them deal with it as a problem of Mathematical programming, where the problem is expressed in a mathematical model and solved by mathematical methods. But still, there are not many Constraint programming approaches towards the problem as Constraint programming is relatively young comparing with other problem solving techniques.

1.1 Related problems

As we will show later in chapter 3, the rotation assignment problem is in fact an instance of the NP-Complete problem of Set partitioning [10] and as such it is related to many theoretical and practical problems. From the theoretical related problems we can name the Set covering problems, the Bin packing problem and other partitioning, covering NP-Hard problems [10]. From the practical problems, the Cutting stock problem [20] is one of the most related, the task of the problem is to cut a long roll of paper into patterns of smaller rolls, so that the cost associated with each pattern is minimized while the demands for smaller rolls are satisfied – in this sense, we can see that there's a relation between the patterns and the schedules for individual aircrafts and a relation between the paper rolls and the activities. Also, the Vehicle routing problem [19] can be considered as a related problem, where the task is to service a number of customers with a fleet of vehicles – in this case, we can relate the vehicles to the aircrafts and the positions of the customers in locations can be viewed as the positions of the activities in time slots.

1.2 Related works

As already said, the rotation scheduling problem is mostly modeled and solved by mathematical methods, e.g., by Linear programming (LP) techniques. This approach is the most common in the industry (see an exhaustive survey of the techniques in [8] or [9]), thanks to the fact that mathematical methods, particularly linear programming, has been exhaustively studied and many software toolkits are available and optimized over years.

One of the first proposals regarding the problem is the work of A. Levin from 1971 [21] where he proposed a network flow model for the problem and then solved by using LP techniques called Dantzig-Wolfe decomposition and Delayed column generation [22].

Another significant collaboration of 6 authors in the papers of Barnhart et al. [23] emerged in 1998, where the authors solved a combined fleet assignment and aircraft routing problem by an approach based on maintenance feasible strings of activities, that are combined to create feasible routes, within the framework of delayed column generation and a technique called Branch-and-Price [24].

A perfect example for showing the fundamentals of delayed column generation and current mathematical approaches is covered in the book from Michael Pinedo [1]. The rotation assignment problem is abstractly expressed as a combinatorial problem of finding a set of paths of certain properties in a graph. Such graph is then considered as a pricing sub-problem. Because of existence of many such paths, the delayed column generation is deployed to efficiently generate paths. Within the Delayed column generation framework the problem is again modeled as a matrix, a system of linear equations, with an objective function, where each column of the matrix represents a path satisfying the required properties. Because there are many such paths, it is not possible to include all of such columns in the matrix, thus one starts with an initial solution set of columns and generates only additional columns that might improve the solution and one does so only when it is needed, e.g., when optimality is not reached yet. The column generation process is then continued by deducing certain values from the existing set of columns, so called dual variables, which indicates how much each “resource” variables (in our case the activities) might affect the cost of the solution if it will be included in the next generated column. These variables are then projected onto the pricing sub-problem graph to change the weights of the graph’s arcs. After that, a new path from the changed graph is generated and is projected back to the column generation matrix, where the column generation process is iterated again.

In the work of L.-M. Rousseau et al. [14] the authors dealt with the Vehicle routing problem by using constraint programming to solve the pricing sub-problem within the column generation algorithm. The pricing sub-problem model proposed by Rousseau is also based on graph and it uses $successor$ and $predecessor$ variables to describe a path, as well as variables for arrival times and vehicle capacity. The basic constraints are $all\_different$ constraints over the successors, constraints maintaining consistency between successors and predecessors, and time and capacity propagation constraints.

In the Ph.D. thesis of M. Grönkvist [5] the author used constraint programming to get a good initial solution for the column generation process to solve the Rotation assignment problem, since a good initial solution helps converge to optimum much faster. Grönkvist also

build a CP model on a graph to find paths with certain properties in it. The CP model also uses $successor$ and $predecessor$ variables to describe a path and $all\_different$ constraints over the successors to achieve disjointedness of the paths.

However, we shall notice that Grönkvist and Rousseau model their problems in CP as consistency models, where desired properties are strictly required, and optimization is done outside of the CP framework.

Nevertheless, an attempt to perform some kind of optimization in CP has been made, in the Master thesis of E. Kilborn[4] the author used a similar model to the model used by Grönkvist (in fact, Grönkvist was his thesis supervisor) with an extension to the model in the sense that if an activity cannot be included in the solution due to violations of required properties then such activity is simply discarded (such activity is represented in the model as $successor(x) = x$ for some activity x). The optimization is then about minimizing the number of the discarded activities. This is a quite benevolent approach towards solving the problem, as the definition of the problem states that all activities must be allocated to the resources.

I should also mention that my employer is company The Kite, s.r.o. [25], a midsize company in Prague, specializing on optimization of logistics and airlines planning processes for the Czech airliner ČSA and others. And because the results of this thesis will be verified and deployed in their systems, I had access to some of their optimization solutions for the Rotation assignment problem. The first one is called “Complex solution”, the second one “Aircraft usage reduction” and the third one is “FIFO/LIFO”. The first two algorithms are perfect examples of the above mentioned LP techniques, based on column generation technique and proprietary search heuristics they are very fast and able to get good solutions in a short time. The last one, is an algorithm based on search heuristics and backtracking, but sometimes when facing with complicated conflicts it fails to assign all activities to the aircrafts.

In this thesis I have chosen to model and solve the optimization version of the Rotation assignment problem within the Constraint programming framework. I’ve decided to do so in order to figure out whether a CP approach to the problem is practically possible and whether it has advantages over other approaches – if the answer would be positive then the CP model would be further developed and incorporated into existing solutions of the company. Next to it, because the CP techniques represent a new approach which has been successfully applied to many optimization problems, but has not been fully applied in the field of rotation assignment yet, it would be an interesting challenge to figure it out.

The CP model for the problem that I will propose uses the concept of the paths in a graph it uses variables $successor$, $predecessor$ and the constraint $all\_different$ over $successor$ to achieve disjointedness of the paths. In this sense it shares the common concept with the existing works, But such basic model alone, would be sufficient only in case of feasibility check, not for optimization purposes at all. In order to achieve the above mentioned goals we will see that many further adaptions and improvements has to be done.

In the end, I will mention an attempt with a different approach towards the optimization version of the rotation assignment problem. In the work of D. Sosnowsk, J.

Rolim [6] the authors used a combination of probabilistic and local search approaches, particularly the simulated annealing technique, where some local perturbations on an existing solution are made and then accepted on the basis of probability.

1.3 Thesis outline

The thesis is divided into six parts:

Chapter 2: The Rotation Assignment Problem
In this part, the problem is exactly and formally formulated.

Chapter 3: An Abstract Model
In this part, an abstract graph model describing the problem is defined.

Chapter 4: Constraint Programming techniques
In this part, the concepts of Constraint programming are introduced.

Chapter 5: A Constraint Programming approach
In this part, Constraint programming models for the problem are defined. A complete algorithm for solving the problem is introduced.

Chapter 6: Computation results
In this part, computations results are presented.

Chapter 7: Conclusion
In this part, final remarks and conclusion are said.

Chapter 2

The Rotation Assignment Problem

The purpose of this chapter is to describe the problem in details. I will specify the important characteristics of the variables we are going to operate with, I will describe how the solution to the problem looks like, the constraints on it and the characteristic that determines the solution quality.

2.1 Problem specification

Let us consider the input of the Rotation assignment problem as a set of unary resources $R = \{r_i \mid 1 \leq i \leq m\}$, representing a given set of m aircrafts, and a set of non-preemptive activities $A = \{a_i \mid 1 \leq i \leq n\}$, representing a given set of *n* rotations.

Furthermore, for each resource and activity let us focus on their following important characteristics.

Specification of activities variables

Each activity a_i has the following a priori known unchangeable attributes:

- s_i defines the time point when the activity starts
- p_i defines the duration of the activity
- e_i defines the time point when the activity ends, where $e_i = s_i + p_i$
- bt_i defines the non-negative time period during which the activity can consume the allocated resource before it starts
- at_i defines the non-negative time period during which the activity can consume the allocated resource after it ends
- sl_i defines the starting location of the activity
- el_i defines the ending location of the activity
- ac_i defines a subset of R to which the activity can be allocated

Additionally, each activity a_i has a decision variable ar_i determining a resource from R to which the activity is to be allocated to.

Activities are divided into 2 types:

- *Flight activities*
- *Reservation activities*

The reservation activities have identical starting locations and ending locations, i.e., for the reservation activity a_i it holds $sl_i = el_i$. The reservation activities are a priori allocated to their respective resources only, i.e., for the reservation activity a_i it holds that $ac_i = \{r_k\}$ for some $r_k \in R$.

2.2 Solution to the problem specification

The solution to the Rotation assignment problem is a schedule, an allocation of the activities to the resources. In our case, the schedule is represented by a set S of disjoint sequences of the activities $S_{r_i} = \left(a_{1_{r_i}}, a_{2_{r_i}}, \dots, a_{k_{r_i}}\right)$, where $a \in S_{r_i} \Leftrightarrow ar_a = r_i$, $a \in A, r_i \in R$.

Furthermore, on each of the above mentioned sequences S_{r_i} and on every activity a_i we want to impose the following constraints C. These constraints are further divided into *Soft constraints* and *Hard constraints*. With the soft constraints we require that a schedule violating a soft constraint must be penalized by an associated penalty cost. Whereas with the hard constraints, we require a proper schedule to satisfy all such constraints.

Soft constraints are:

- (C1) For each pair of activities (a_i, a_j) from S_{r_i}, where $j \geq i + 1$, the non-overlapping property $e_i \leq s_j$ must hold.
- (C2) For each pair of activities (a_i, a_j) from S_{r_i}, where $j \geq i + 1$, there is a time distance between the end of a_i and the start of a_j, i.e. it must hold that $(s_j - e_i) \geq d_{ij}$, where $d_{ij} = \max\{at_i, bt_j\}$ if both a_i and a_j are flying activities, otherwise $d_{ij} = \min\{at_i, bt_j\}$.

  Note: Observe that there's a certain relation between the constraints C1 and C2 over the distance $(s_j - e_i)$; C1 requires $(s_j - e_i) \geq 0$, whereas C2 requires $(s_j - e_i) \geq d_{ij} \geq 0$. Thus once C1 is violated C2 is violated as well.

Hard constraints are:

- (C3) For each pair of activities (a_i, a_j) from S_{r_i}, where $j = i + 1$, the starting location of $a_j$must be the same as the ending location of a_i, i.e. $sl_j = el_i$.
- (C4) Each activity a_i must be compatible with its allocated resource r_i, i.e. $ar_i \in ac_i$.
- (C5) Each activity must be allocated to exactly one resource, that means for any resource $r_j \neq r_i$ and its nonempty sequence of activities S_{r_j}it must hold that $S_{r_i} \cap S_{r_j} = \emptyset$ and $\forall\, a_i \in A\ \exists\, r_i : a_i \in S_{r_i}$, hence $\bigcup_{r_i \in R} S_{r_i} = A$.
- (C6) The starting times of activities in the same sequence must be increasing, i.e. $\forall (a_i, a_j) \in S_{r_i}, j \geq i + 1,\ s_j > s_i$.

2.3 Schedule quality evaluation

To evaluate the quality of the schedule and its consistency with the constraints C we consider an objective function $F(S, R, C)$, which determines the cost value for a set S of the sequences of activities A allocated to resources R violating the soft constraints of C.

The objective function is formulated as the following:

$$F(S,R,C) = \sum_{\substack{r \in R \\ S_r \in S}} \left[\pi^a(r) + \sum_{\substack{a_i, a_j \in S_r \\ i<j}} \pi^C(a_i, a_j) \right] \tag{2.1}$$

Where each cost sub-function π is defined as follows:

- $\pi^a(r)$ calculates the cost value for assigning activities to resource r, i.e.

$$\pi^a(r) = \begin{cases} c_r, & S_r \neq \emptyset \\ 0 & otherwise \end{cases} \tag{2.2}$$

Where c_r is a predefined constant.

- $\pi^C(a_i, a_j)$ determines the cost which is proportional to $\mathrm{T} = s_j - e_i$ time units of a possible violation of either the constraint (C1) or (C2), i.e.,

$$\pi^C(a_i, a_j) = \begin{cases} C_a^x + C_b^x \tau + C_c^x \tau^2, & \tau \geq Lt^x \\ C_d^x, & 0 < \tau < Lt^x \end{cases} \tag{2.3}$$

In case of violation of (C1) when $\mathrm{T} < 0$:

$\tau = |T|,\ C_a^x = C_a^1, C_b^x = C_b^1,\ C_c^x = C_c^1, C_d^x = C_d^1, Lt^x = Lt^1.$

In case of violation of (C2) when $0 \leq \mathrm{T} < d_{ij}$:

$\tau = d_{ij} - T,\ C_a^x = C_a^2, C_b^x = C_b^2,\ C_c^x = C_c^2, C_d^x = C_d^2, Lt^x = Lt^2$

Where $C_a^1, C_b^1, C_c^1,\ C_d^1,\ Lt^1, C_a^2,\ C_b^2,\ C_c^2,\ C_d^2,\ Lt^2$ are predefined real constants smaller than infinity (∞).

Note: Each $\pi^C(a_i, a_j)$ is in fact a constant since it can be a priori determined independently on a schedule.

Definition 2.1.: We call the schedule a *feasible schedule* iff the schedule satisfies all the hard constraints of C.

Definition 2.1.: We call the schedule an $optimal\ schedule$ iff the schedule is *feasible,* there's no other feasible schedule that has lower cost, and its cost is less than infinity (∞).

Definition 2.3.: We call a sequence of activities S_{r_i} allocated to a resource r_i a $resource\ schedule\ S_{r_i}$.

Note: For us a schedule is simply an allocation of the activities to the resources. Thus, even a resource schedule S_{r_i} is a representation of a schedule, albeit not necessarily a feasible one.

Chapter 3

An Abstract Model

In this chapter I will describe the problem as an abstract model, a graph model, which lays the foundations for solving the problem. I will focus on how to represent the activities, the relations between the activities and finally, how to define a solution to the problem in the abstract model.

3.1 A graph model

Model 3.1: Weighted connection network

I have chosen to model the problem with a so-called *weighted connection network* [15]. The approach to model the problem as a graph is very common. Particularly, the following proposed graph was inspired by a graph model which can be found in the book of M. Pinedo [1], where the author defined five types of nodes and arcs to capture, among other issues, the origination of an activity, the termination of an activity and certain time compatibility between two activities. However, since we want to capture the resources compatibility of the activities, time and location compatibility between two consecutive activities and certain cost associated for each two consecutive activities, we will have to define our model differently. Let us propose the following graph model:

The *weighted connection network is a directed graph*

$$G = (V, E, c)$$

where V is a set of vertices and $|V| = |A| + |R| + 2 = n + m + 2$, $E \subseteq V \times V$ is a set the arcs and $c: V \times V \to \mathbb{R}$ is a weight function. (See section 2.1.0 for the definitions of A and R.)

We define V as four types of vertices - *source*, *sink*, *resource vertices* and *activity vertices*:

- Activity vertices represent the activities which are to be scheduled, i.e. $\forall a \in A \; \exists v_a \, \epsilon \, V$.
- Resource vertices represent the resources to which activities can be allocated to, i.e. $\forall r \in R \; \exists v_r \, \epsilon \, V$.
- *Source* and *sink* are two special vertices from which (to which) arcs to (from) other vertices can be linked.

Next, we define E and four types of arcs - *source arcs, sink arcs, resource arcs* and *activity arcs*:

- Source arcs lead from the source vertex to every resource vertex, i.e. $\forall r \in R \ \exists e = (v_{source}, v_r) \epsilon E$.
- A resource arc going from a resource vertex to an activity vertex expresses that the activity associated with the activity vertex can be allocated to the particular resource, i.e. $\forall r \epsilon ac_a \ \exists e = (v_r, v_a) \epsilon E, \ \forall a \in A$.
- An activity arc between two activity vertices means that a connection between them is possible, i.e.,
  $\forall a_i, a_j \in A \ (a_i, a_j)$ satisfies $C3$ and $C6 \wedge ac_i \cap ac_j \neq \emptyset \Rightarrow \exists e = \left(v_{a_i}, v_{a_j}\right) \epsilon E$
- And at last, sink arcs go from each activity vertex to the sink vertex, i.e. $\forall a_i \in A \ \exists e = \left(v_{a_i}, v_{sink}\right) \epsilon E$.

Finally the weight function $c: V \times V \rightarrow \mathbb{R}$ is defined as follows:

(3.1)

$$c(v_f, v_t) = \begin{cases} \pi^a(r) & v_f \text{ resource vertex for } r \wedge v_t \text{ activity vertex} \\ \pi^c(a_f, a_t) & v_f, v_t \text{ activity vertices} \\ 0 & otherwise \end{cases}$$

Note: Notice that, the graph model we have just proposed is by definition an acyclic directed graph. Since there's no node leading back to the source, neither to the resource vertices, originations of the activity arcs represent activities that start earlier than the activities represented by the terminations of the arcs, and the last there's no arc leading from the sink.

3.2 Problem solution in the Abstract model

Notation: From now we will denote a path with a vertex v_i to vertex v_j by a symbol $v_i \rightsquigarrow v_j$.

Observation 3.1: For an abstract model graph $G = (V, E, c)$, any path $source \rightsquigarrow sink$ contains at least one resource vertex and at least one activity vertex.

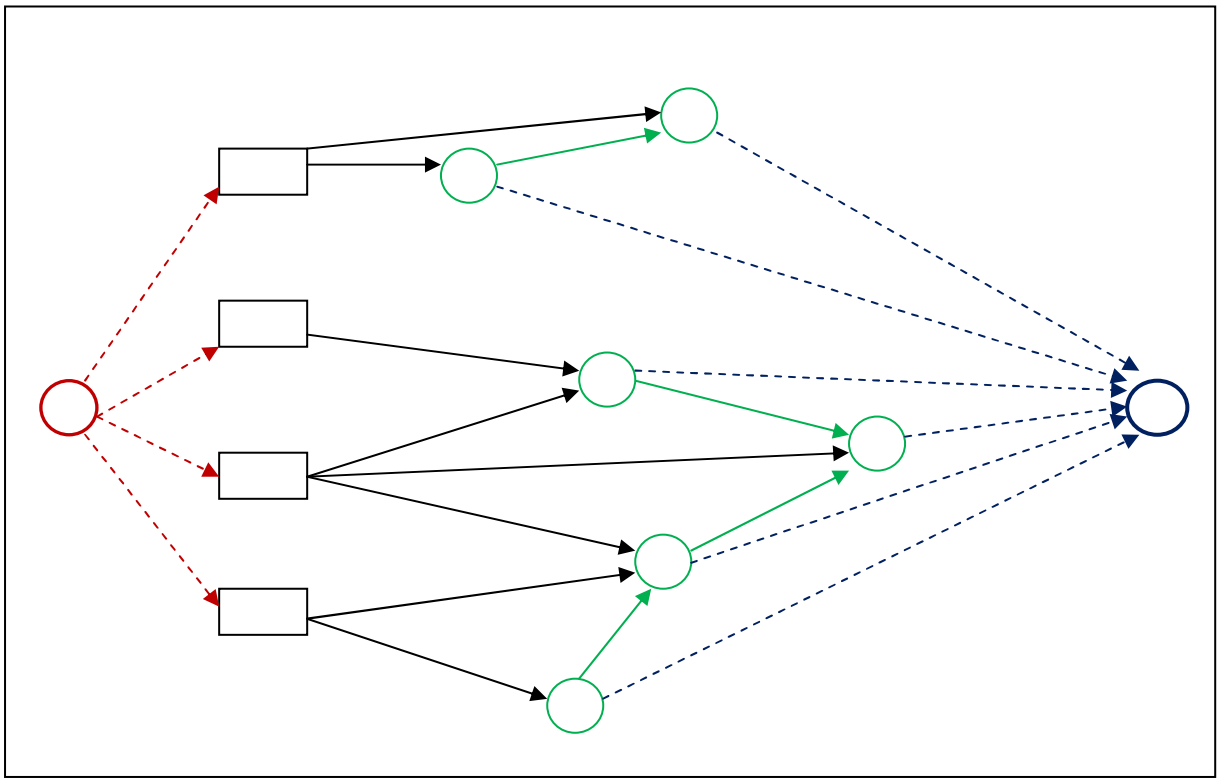
Proof: By the definition of the model, there's no arc leading directly from the source to the sink in G, neither a direct arc from the source to an activity vertex, neither a direct arc from a resource vertex to the sink, thus any path $source \rightsquigarrow sink$ must contain at least a resource vertex and at least one activity vertex.

Observation 3.2: In an abstract model graph $G = (V, E, c)$ any path $p \, source \rightsquigarrow sink$ contains exactly one resource vertex, the ending vertex of the first arc of the path, i.e. $p = \left(v_{0_p} = source, v_{1_p} = v_r, \dots, v_{k_p} = sink\right)$ for some $v_r \epsilon V$. And the path represents a resource schedule.

Proof: Firstly, let us prove that the path contains exactly one resource vertex. By the definition of the model, there's no arc leading to resource vertices in G, except those from the source, and because there's no arc leading to the source vertex path p cannot contain more than one resource vertex. From the previous observation 3.1 it is implied that there's exactly one resource vertex on p.

Now, let us proof that the path represents a resource schedule. In section 2.2.0 we defined that a resource schedule is sequence of the activities allocated to a resource. In this case it is true; the activities which are expressed by the activity vertices of the path (from observation 3.1. there's at least one activity vertex) are allocated to the resource which is expressed by the only resource node of the path.

Note that, the resource schedule represented by the path might not necessarily satisfy the hard constraints C.



The source vertex and arcs are in red, the resource vertices and arcs are in black, the activity vertices and arcs are in green, the sink vertex and arcs are in blue.

Figure 3.1 An example of a weighted connection network.

Observation 3.3: An *optimal solution* for the Rotation assignment problem AP can be formulated as a set SP of paths $source \rightsquigarrow sink$ in graph $G_{AP} = (V, A, c)$ as follows:

(3.2)

$$SP := \underset{\substack{P\ set\ of\ paths\ source \rightsquigarrow sink \\ P\ in\ G_{AP}}}{\operatorname{argmin}} \sum_{p \in P} \bar{c}(p)$$

Such that

$$\sum_{p \in P} y_{v_a p} = 1, \qquad \forall v_a\ activity\ vertex \in V \tag{3.3}$$

$$\sum_{\substack{p \in P \\ v_r resource\ vertex \in p}} y_{v_a p} z_{v_r v_a} = 1, \qquad \forall v_a\ activity\ vertex \in V \tag{3.4}$$

Where

$$\bar{c}(p) := \sum_{\substack{(v_r, v_i) \in p \\ v_r\ res.\ vertex}} c(v_r, vi) + \sum_{\substack{v_i \rightsquigarrow v_j \subseteq p \\ v_i\ not\ res.\ vertex}} c(v_i, v_j), \quad p\ source \rightsquigarrow sink$$

$$y_{v_a p} = \begin{cases} 1 & v_a \in p \\ 0 & otherwise \end{cases}$$

$$z_{v_r v_a} = \begin{cases} 1 & (v_r, v_{a_i}) \epsilon\ E \\ 0 & otherwise \end{cases}$$

Proof**:** The schedule represented by the set of paths $SP = \{p_1, p_2, \dots, p_k\}$ satisfies all the hard constraints of C and its cost is minimum because:

- A resource schedule represented by a path $p_i \in SP$ is a resource schedule for exactly one resource due to observation 3.2 and it has at least one activity due to observation 3.1.
- A resource schedule represented by a path $p_i \in SP$ satisfies constraints $C3$ and $C6$ because all activity arcs of p_i satisfy the constraints by the definition of the activity arcs.
- The schedule represented by SP satisfies constraint $C5$ because:

  All activities are allocated to the resources, i.e. $\forall a \in A\ \exists S_r\ a \in S_r$, i.e. $\forall v_a\ activity\ vertex \in V\ \exists\ path\ p \in SP$ s.t. $y_{v_a p} = 1$.

For contradiction, suppose that equation 3.3 holds but there's an activity a that is not allocated to any resource. That would mean that for $v_a \in V$ representing activity a $y_{v_a p} = 0, \forall\, path\ p \in SP$, i.e. $\sum_{p \in SP} y_{v_a p} = 0$.
But that is in contradiction with the premise that equation 3.3 holds.

Furthermore, each activity is allocated to exactly one resource.
For contradiction, suppose that equation 3.3 holds but there's an activity a that is allocated to more than one resource, i.e. $a \in S_{r_i} \wedge a \in S_{r_j}, r_i \neq r_j$. By definition, it would mean that for $v_a \in V$ representing activity a $\exists p_i, p_j \in SP$, $p_i \neq p_j, y_{v_a p_i} = 1 \wedge y_{v_a p_j} = 1$, thus $y_{v_a p_i} + y_{v_a p_j} > 1$. By definition, $y_{v_a p_k} \geq 0$ $\forall p_k \in SP$, thus it must hold that $y_{v_a p_i} + y_{v_a p_j} + \sum_{p_k \in SP, p_k \neq p_i, p_k \neq p_j} y_{v_a p_k} > 1$.
But that is in contradiction with the premise that equation 3.3 holds.

- The schedule represented by SP satisfies constraint $C4$ because:

  All activities are allocated to their compatible resources only.
  For contradiction, suppose that equations 3.3, 3.4 hold but $\exists$activity a and resource r s. t. $a \in S_r \wedge r \notin ac_a$. By definitions, for $v_a \in V$ representing a and path $p_i \in SP$ representing S_{r_i} it holds $v_a \in p_i$, thus $y_{v_a p_i} = 1$.
  By definition $r_i \notin ac_i \rightarrow (v_{r_i}, v_a) \notin E$ for v_{r_i} representing resource r_i, thus $z_{v_{r_i} v_a} = 0$.
  Equation 3.3. holds, so $\sum_{p \in P} y_{v_a p} = y_{v_a p_i} + \sum_{p_j \in SP, p_j \neq p_i} y_{v_a p_j} =$ $1 + \sum_{p_j \in SP, p_j \neq p_i} y_{v_a p_j} = 1$, hence $\sum_{p_j \in SP, p_j \neq p_i} y_{v_a p_j} = 0$. By definition $\forall p_j \in SP$ $y_{v_a p_j} \geq 0$, thus $\forall p_j \in SP\ \ p_j \neq p_i,\ y_{v_a p_j} = 0$.
  Thus

$$\sum_{\substack{p \in SP \\ v_r\ resource\ vertex \in p}} y_{v_a p} z_{v_r v_a} = y_{v_a p_i} z_{v_{r_i} v_a} + \sum_{\substack{p_j \in SP \\ p_j \neq p_i}} y_{v_a p_j} z_{v_{r_j} v_a}$$

$$= 1 * 0 + \sum_{\substack{p_j \in SP \\ p_j \neq p_i}} 0 * z_{v_{r_j} v_a} = 0$$

  But this is in contradiction with the premise that 3.4 holds.

- The schedule represented by SP is a feasible schedule.
  From the previous proofs we can observe that is true because the schedule satisfies all the hard constraints.
  Consequence: Any set of paths $source \rightsquigarrow sink$ that satisfy equations 3.3 and 3.4 represents a feasible schedule and vice versa, every feasible schedule can be represented by such set of paths.

- The sum $\sum_{p \in SP} \bar{c}(p)$ is equal to the cost of the schedule represented by SP.
  To prove that, let us prove that the cost of a path computed by function $\bar{c}$ is equal to the cost of a resource schedule represented by the path first.

Consider arbitrary path $p = (v_{source}, v_r, v_{1_p}, \dots, v_{k_p}, v_{sink}) \in SP$ representing a resource schedule S_r, we will prove that $\bar{c}(p) = \pi^a(r) + \sum_{\substack{a_{i_r}, a_{j_r} \in S_r \\ i<j}} \pi^C(a_{i_r}, a_{j_r})$.

According to observation 3.1 and 3.2 there's exactly one resource vertex and at least one activity vertex in the path. That implies that there's exactly one resource arc, which is (v_r, v_{1_p}). From the definition of the weight function c, $c(v_r, v_{1_p}) = \pi^a(r)$. Also by definition, $\forall$pair of activity vertices $v_{i_p}, v_{j_p} \in p$ representing a pair of activities $a_{i_r}, a_{j_r} \in S_r$ it holds that $c(v_{i_p}, v_{j_p}) = \pi^C(a_{i_r}, a_{j_r})$. Thus for path p representing resource schedule $S_r = (a_{1_r}, a_{2_r}, \dots, a_{k_r})$ it holds that

$$\bar{c}(p) = c(v_{source}, v_r) + c\left(v_r, v_{1_p}\right) + \sum_{1 \le i<j \le k} c\left(v_{i_p}, v_{j_p}\right) + c(v_k, v_{sink}) =$$

$$= 0 + \pi^a(r) + \sum_{1 \le i<j \le k} \pi^C(a_{i_r}, a_{j_r}) + 0 = \pi^a(r) + \sum_{\substack{a_{i_r}, a_{j_r} \in S_r \\ i<j}} \pi^C(a_{i_r}, a_{j_r})$$

Because path p was an arbitrary path, it holds for all paths of SP.

Let us now prove that the cost of the schedule $S = \left\{S_{r_{1_S}}, S_{r_{2_S}}, \dots, S_{r_{k_S}}\right\}$ represented by $SP = \left\{p_{1_{SP}}, p_{2_{SP}}, \dots, p_{k_{SP}}\right\}$ is equal to $\sum_{p \in SP} \bar{c}(p)$. From the definition of function 2.1 we know that the cost of S is a sum of the costs of each resource schedule of S, from the previous proof we know that the cost of a resource schedule is equal to the cost of a path representing it, thus:

$$\sum_{p \in SP} \bar{c}(p) = \sum_{\substack{p \in SP \\ S_r represented\ by\ p}} [\pi^a(r) + \sum_{\substack{a_{i_r}, a_{j_r} \in S_r \\ i<j}} \pi^C(a_{i_r}, a_{j_r})] =$$

$$\sum_{S_r \in S} [\pi^a(r) + \sum_{\substack{a_{i_r}, a_{j_r} \in S_r \\ i<j}} \pi^C(a_{i_r}, a_{j_r})] = \sum_{S_r \in S} [\pi^a(r) + \sum_{\substack{a_i, a_j \in S_r \\ i<j}} \pi^C(a_i, a_j)]$$

- The schedule represented by SP has the lowest cost because:

  For contradiction suppose that there's another schedule $\tilde{S}$ which is represented by a set of paths $\widetilde{SP}$ and whose cost is lower than the cost of schedule S represented by SP. From the previous proof we can see that the cost of $\tilde{S}$ is equal to the cost of $\widetilde{SP}$ and the cost of S is equal to the cost of SP. Hence the cost of $\widetilde{SP}$ is lower than the cost of SP. But that is in contradiction with the minimum cost of SP.

Observation 3.4: The Rotation assignment problem (ROT) is a special case of the Minimum set partitioning problem (MSP) [10]. That is, ROT is NP-Hard.

Proof: The MSP problem is formulated as follows, for a given universe U of m elements and a given set of subsets $SB = \{SB_1, SB_2, \dots, SB_n\}$ of U, a set partition is a set $X = \{SB_{1_X}, SB_{2_X}, \dots, SB_{k_X}\}$ s.t. $X \subseteq SB$ and $\bigcup_{SB_i \in X} SB_i = U$, $\bigcup_{i \neq j} SB_i \cap SB_j = \emptyset$. The optimization task of the MSP is to find a set partition X with the minimum cost with a given cost function $Cost(SB_i)\ \forall SB_i \subseteq U$ and $Cost(X) \coloneqq \sum_{SB_i \in X} Cost(SB_i)$.

Since it is known that MSP is a NP-Hard problem [10], in the complexity theory in order to show that the ROT problem is NP-Hard as well, we need to show that there exists a reduction, a transformation of one problem to the other, i.e. $\exists f$ polynomial time complex s.t. $\forall x \in$ MSP iff $f(x) \in$ ROT.

We will construct the reduction f as follows:

- $\forall SB_i \in SB$ create a resource r_i, i.e. resource set $R \coloneqq \{r_i | SB_i \in SB\}$
- For convenience, for each element of U assign it a unique index from 1 to m, so we can denote U as $U = \{elmt_1, elmt_2, \dots, elmt_m\}$
- For $\forall elmt_k \in U$ create an activity a_k such that:
  - $s_k = i, p_k = 0.99, e_k = s_k + p_k, bt_k = at_k = 0, sl_k = el_k = Hanoi$
  - $ac_k = \{r_i\ | SB_i \in SB,\ elmt_k \in SB_i\}$
- All the constants for π^C are 0, i.e. $\pi^C(a_i, a_j) = 0\ \forall a_i, a_j$
- $\forall r_i \in R\ c_{r_i} = \begin{cases} Cost(SB_i), & \forall a \in A \text{ s.t. } r_i \in ac_a \rightarrow a \in S_{r_i} \\ \infty, & otherwise \end{cases}$

Notation: $\bar{f}$ means reduction f applied only on elements or set of elements of U.

Observe that, $\forall SB_i = \{e_{1_i}, e_{2_i} \dots, e_{k_i}\} \subseteq U \quad \exists Sr_i = \bar{f}(SB_i), Sr_i = \{a_{1_{r_i}}, a_{2_{r_i}} \dots, a_{k_{r_i}}\}$ with the cost $c_{r_i} = Cost(SB_i)$. Also observe that, any schedule S satisfies constraints $C3, C4$ automatically due to the values of the attributes of the activities and that we can always linearly order them in the way to make the constraints hold. Also by definition, any arbitrary resource schedule $S_r = \{a_{1_r}, a_{2_r}, \dots, a_{k_r}\}$ satisfies $C6$ because
$\forall a_i, a_j \in S_r, i < j$: $s_i = i < j = s_j$.

"$f(x) \in ROT \rightarrow x \in MSP$":

Observation: If S is an optimal and feasible schedule, then $\forall S_{r_i} = \{a_{1_{r_i}}, a_{2_{r_i}} \dots, a_{k_{r_i}}\} \in S$ $\exists\ SB_i = \{e_{1_i}, e_{2_i} \dots, e_{k_i}\}$ s.t. $\bar{f}^{-1}(S_{r_i}) = SB_i$.

Proof: because $\forall a_{k_{r_i}} \in S_{r_i}$: $a_{k_{r_i}} \in S_{r_i} \rightarrow r_i \in ac_{k_{r_i}}$, thus $\exists\ e_{k_{r_i}} \in SB_i$.

Then it means $|SB_i| \geq |Sr_i|$. Let us prove that $|SB_i| = |Sr_i|$. For contradiction, if $|SB_i| > |Sr_i|$ then $\exists\ e_{x_i} \in SB_i\ \ \bar{f}(e_{x_i}) = a_{x_i} \notin S_{r_i}$ but $r_i \in ac_{x_i}$. Suppose that $S_{r_i} \neq \emptyset$ then by definition $\pi^a(S_{r_i}) = c_{r_i} = \infty$, thus $F(S, R, C) = \infty$, which is not optimal.

An optimal and feasible schedule S after reduction f^{-1} represents an optimal partition X because:

- By the definition of a feasible schedule, for $\forall S_{r_i}, S_{r_j} \in S: S_{r_i} \cap S_{r_j} = \emptyset$, thus $\forall \left[SB_i = \bar{f}^{-1}(S_{r_i}),\ SB_j = \bar{f}^{-1}\left(S_{r_j}\right)\right]: SB_i \cap SB_j = \bar{f}^{-1}(S_{r_i}) \cap \bar{f}^{-1}\left(S_{r_j}\right) = \bar{f}^{-1}\left(S_{r_i} \cap S_{r_j}\right) = \bar{f}^{-1}(\emptyset) = \emptyset$.
- By definition, $\bigcup_{S_{r_i} \in S} S_{r_i} = A$, thus $U = \bar{f}^{-1}(A) = \bar{f}^{-1}\left(\bigcup_{S_{r_i} \in S} S_{r_i}\right) = \bigcup_{S_{r_i} \in S} \bar{f}^{-1}(S_{r_i}) = \bigcup_{\substack{SB_i = \bar{f}^{-1}(S_{r_i}) \\ S_{r_i} \in S}} SB_i$.
- Firstly, let us prove that the cost of S is equal to the cost of X because:

$$F(S,R,C) = \sum_{\substack{r_i \in R \\ S_{r_i} \in S}} [\pi^a(r_i) + \sum_{\substack{a_i, a_j \in S_{r_i} \\ i<j}} \pi^C(a_i, a_j)] = \sum_{S_{r_i} \in S} \pi^a(r_i) + 0 = \sum_{S_{r_i} \in S} c_{r_i}$$
$$= \sum_{SB_i = \bar{f}^{-1}(S_{r_i}) \in S} Cost(SB_i) = Cost(X)$$

S has the minimum cost, thus X has the minimum cost as well. Because for contradiction, if there's another $\bar{X}$ s.t. $Cost(\bar{X}) < Cost(X)$, then there exists a schedule $\bar{S} = f(\bar{X})$ whose cost $Cost(\bar{X}) = F(\bar{S}, R, C) < F(S, R, C) = Cost(X)$, but that is in contradiction with the minimum cost of S.

- Thanks to the definitions of $ac_k \forall\ a_k \in A$ and the cost values:

  $\forall S_{r_i} \in S\ \bar{f}^{-1}(S_{r_i}) = SB_i \in SB$. Thus $X \bigcup_{\substack{SB_i = \bar{f}^{-1}(S_{r_i}) \\ S_{r_i} \in S}} SB_i \subseteq SB$, i.e. only "allowed" resource schedules are optimal ones, because if $\exists S_{r_i}\ \bar{f}^{-1}(S_{r_i}) = SB_i \notin SB$ then there is not a defined resource r_i then $S_i \notin SP$ either.

"$x \in MSP \rightarrow f(x) \in ROT$":

Analogically, by the definition, for an optimum partition X, it holds that X has minimum cost thus a schedule S reconstructed by reduction f has the minimum cost, X covers all elements of U thus S contains all activities, sets in X are disjoint thus corresponding resource schedules are disjoint, other hard constraints hold because of the definition of the reduction for elements of U to the activities and $X \subseteq SB, SB$ defines a set of feasible resource schedules.

Consequence: After showing that the Rotation assignment problem is NP-Hard, we can conclude that it is highly improbable that there exists a polynomial time algorithm for the problem, unless NP = P. And that in spite of all our efforts, our designed algorithm cannot guarantee polynomial time complexity and in some cases it can have exponential time complexity.

Chapter 4

Constraint Programming techniques

In this chapter I will very briefly introduce the concept of Constraint Programming (CP) and its essentials and techniques. The knowledge about CP I've received is from the lectures of Professor Roman Barták [2], for further reading about CP I recommend, for instance, the book by F. Rossi, P. van Beek, T. Walsh [3].

4.1 Constraint programming

Constraint programming is a problem solving paradigm, which lies in precise definitions and separation of *variables*, *constraints* and *search algorithms* involved in solving a combinatorial problems. A combinatorial problem is then solved by being defined as an *instance* of the Constraint Satisfaction Problem, i.e. as a finite set of variables and a finite set of constraints between the variables.

4.1.1 Variables

Variables usually represent decisions or alternatives for some significant objects of the problem which is to be solved. Each variable x has a *domain* $D(x)$ expressing possible values the variable can take.

4.1.2 Constraints

Obviously, constraints play a central role in CP. A constraint is simply a relation defined on a set of variables limiting possible combination of values of the involved variables. Constraints can be of various kinds, e.g. an arithmetic formula, a Boolean formula, or a very abstract relation. Constraints can include various number of variables, called as the *arity* of a constraint, and can be stated either *implicitly* (e.g. by an arithmetic formula), or *explicitly* (i.e., a constraint is enumerated as a set of tuples of values that satisfy the constraint). An assignment of values to variables that satisfy all the constraints involved in an instance of the CSP is called a *solution* of the CSP.

Note:
A constraint $c(x_1, \dots, x_k)$ is satisfied if $\exists(d_1, \dots, d_k) \in c$ for $d_1 \in D(x_1), \dots, d_k \in D(x_k)$

Example 4.1: Let us demonstrate the basics of CP on a simple CSP example. Consider a 3 integer variables X, Y, Z, whose domains are $D(X) = \{1,2\},\ D(Y) = \{1\},\ D(Z) = \{1,2\}$ and constraints are $X \neq Y, Z \neq X$. Then the solution to the example is for instance assignment $X = 2,\ Y = 1,\ Z = 1$.

4.2 Constraint propagation

Constraint propagation is one of the key ideas of CP. The idea is that the constraints are not only used to test the validity of a solution, but also as a way to remove the values from the domain (a process called *domain reduction)*, deduce new constraints and detect combination of values of variables that can never be in a solution (such values are so-called *inconsistencies*). This process can be very complex and solving an instance of CSP is NP-complete (after all, sometimes we solve NP-hard problems as a CSP [11])

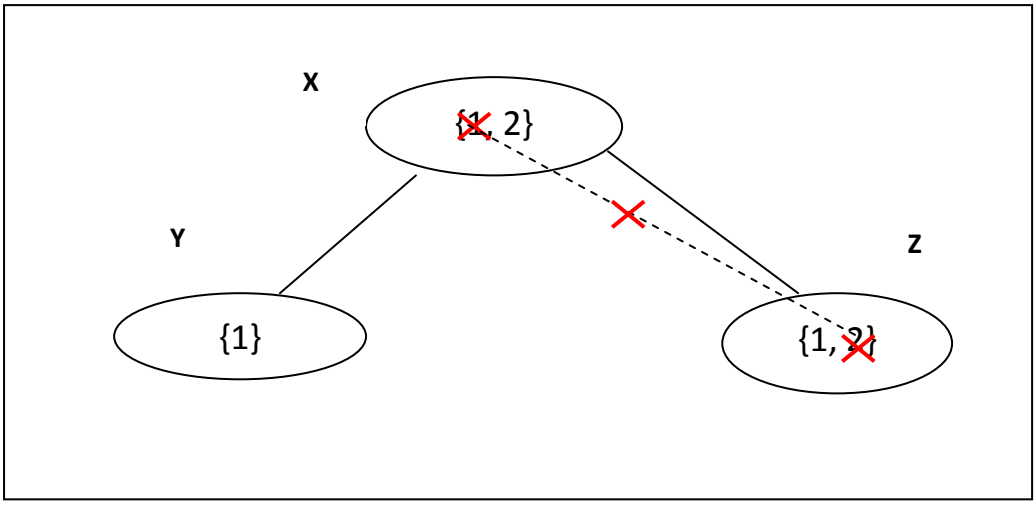
4.2.1 Consistency techniques

One of the ways to achieve a solution in CP is through removing inconsistent values from the variables' domains. Such techniques are called consistency techniques, which were first introduced in the work of Waltz[12].

Since consistency techniques are crucial components of constraint propagation, several techniques have been proposed. The names of basic consistency techniques are derived from the graph notions, where constraints are represented as a $constraint\ graph$ where each node represents a variable and each arc corresponds to a binary constraint (it was shown that arbitrary CSP can be transformed to an equivalent binary CSP [13]). The consistency techniques varies in complexity, from the simplest $node\ consistency$, which removes from the variables' domains values inconsistent with unary constraints, to $arc\ consistency$, which removes values from the variables' domains that are inconsistent with binary constraints, to complex $path\ consistency$ and $K-consistency$ techniques [2].

To grasp the ideas of consistency techniques, let us briefly describe the most widely used arc consistency algorithm AC-3 [2] and demonstrate in on example 4.2 below. The algorithm is based on repeated $revisions$ of the arcs, removing inconsistencies from the domains of variables involved in the respective binary constraints represented by the arcs, till a consistent state is reached or some domain becomes empty. The AC-3 algorithm is efficient in the way that it performs re-revisions only for those arcs that are possibly affected by a previous revision.

Example 4.2: Let us solve the problem from example 4.1 by AC-3. By looking at constraint $X \neq Z$, we can conclude that $X \neq Z$ no revisions were performed on neither arc (X, Z) nor (Z, X). By analyzing the constraint $X \neq Y$ and arc (X, Y) we can deduce that, if X will be assigned to 1 then there's no value in $D(Y)$ to make the constraint $X \neq Y$ hold (there's no $support\ value$ for 1), thus the revision removes value 1 from $D(X)$. We do not need to revise arc (Y, X)- this is the smart trick of AC3 of not performing unnecessary re-revisions from deducing, that since 1 didn't have a support value in $D(Y)$ then no value in $D(Y)$ needed 1 as a support value either. But now, the domain $D(X)$ has changed and hence $X \neq Z$ might not hold and we need to revise arc (Z, X) again. Revision of arc (Z, X) removes value 2 from $D(Z)$. After performing AC3 the domains are in a consistent state with $D(X) = \{2\}, D(Y) = \{1\},\ D(Z) = \{1\}$. Below is figure 4.1 illustrating the situation.



AC3 removes local inconsistencies
Figure 4.1

4.2.2 Search strategies

Due to that most of constraint propagation techniques are incomplete in the sense of not removing all inconsistencies, some kind of search is needed in order to find a solution of a problem. Usually, search is implemented by the means of some tree search algorithm. Roughly spoken, there are two general components of a tree search algorithm, the way to go "forward", i.e. when and which decisions are taken (in case of a CSP it is which values will be assigned to variables) in the search – so called *search heuristics*, and the way to "backward", i.e. how the algorithm shall behave when a contradiction occurs – referred as *backtracking strategy*.

As mentioned above, one of the essential components of searching for a solution are search heuristics. In CP, the way how one can define the search heuristics is by variable and value ordering. The strategy for selecting variables and values at each point can be crucial for the performance of a search algorithm. One of the often mentioned strategies is that one can select a variable that is likely to fail and a value that is likely to succeed. But of course, the ordering doesn't have to be statically set but may depend on the context a search has performed so far.

4.2.3 Optimization in CP

Optimization in CP is mostly done in a very simplistic way. The often mentioned approach is by defining an objective function over the variables involved in the optimization and expressing the function as a cost (CSP) variable, on which the optimization constraints are applied, e.g. at the beginning a solution is found in a normal way without optimizing, then, the cost of the just found solution is used as a bound value to constrain the cost of the consequent solutions to be better, after that, one can restart or continue the search.

The approach described above is simple but it would be more efficient, if we efficiently exploit of the structure of the objective function or use the knowledge from the previous solution's computation. Possible ways how to improve the approach is, for instance, using parallel search, where results of one search line can be congruently propagated to other search lines; or using a heuristic function that optimistically forecasts the cost of the current solution and stop the search if even the optimistic heuristic cannot reach the required result.

Chapter 5

A Constraint Programming approach

So far I've been talking about the connection network model and constraint programming just to lay foundations for a constraint programming model for the Rotation assignment problem. So let us go ahead to formulate the basic constraint model and then, step by step, we will improve and modify it to be more efficient.

5.1 Specification of the CP model

The constraint programming model is naturally derived from the graph model in section 3.1.0.

In the graph model, we found out that a solution to the rotation assignment problem is a set of disjoint paths $source \rightsquigarrow sink$ that have the least cost. To describe such paths we need the variables that are sufficient to express that. A natural way how to describe a path is that for each node of the path, representing an activity, we consider its successor node on the path, representing the activity that can start after the respective activity. For this purpose we define for a node x a variable called $Successor(x)$ whose domain $D(Successor(x))$ expresses all possible successor nodes of x.

So, together x and $Successor(x)$ represent a connection, but of course it is still not sufficient since in this context it is possible that a path can have a repeated node or it can have a node in common with a different path, thus the paths are not disjoint. To prevent this situation to happen we use a constraint called $all\_different\ (x_1, x_2, \dots, x_n)$ [16]. This standard constraint restricts that all its variables in the argument must be assigned to different values. Thus, by using the constraint in the form of $all\_different\ (Successor(x_1), Successor(x_2), \dots, Successor(x_n))$ we can achieve that there's no node that would be a successor node of multiple nodes, thus all the paths are disjoint.

One might ask, how can we be sure that all the paths constructed by the $Successor$ variables and the $all\_different$ constraint are the paths $source \rightsquigarrow sink$, as it might happen that the paths create cycles? Note that, it cannot happen in our CP model, since it is built on the graph model, which we already noticed that it is acyclic. So, the paths in the CP model can never create cycles.

However, by enforcing disjointedness of the paths $source \rightsquigarrow sink$ we encounter a problem. All the desired paths start from the source and end at the sink, so they cannot be disjoint. Also, in the solution set of the paths there are multiple arcs emanating from the source node. How can we resolve these problems? In the CP model we will not consider the source and the sink nodes and arcs simply don't exist in the CP model. But that creates a new problem - what are successors of the last nodes, how can we quickly recognize what is the first or the last node of a path by just looking at the variables $Successor$? We can again

recall the graph model 3.1 and its m resource nodes and apply them in this situation. We will make them to represent the first nodes but also the last nodes of the paths. How can we make them be the first nodes? In the graph model we notice that, for each activity that is compatible with its resource there's a resource arc leading from the particular resource vertex to the activity node, so since there aren't the source and the sink nodes, the resource vertices can already represent the first nodes. However, in order to make them the last nodes, we will extend the domains of the $Successor$ variables representing the activities by the resources that they can be allocated to. And at last, to model the situation that a resource is not used we will add a loop arc going from the vertex representing the particular resource back to itself. In order to do so, for each resource r we will extend the domain $D(Successor(r))$ by value r. Notice that, in effect, after all these extensions we have introduced new cycles into the model, but it will not make any problem as the cycles are disjoint.

However, is it sufficient to describe our problem so far? The answer is no, because obviously we are missing a way how to express to which resource activities can be allocated to. For this purpose we define a variable $Resource(x)$ for a node x, whose domain enumerates all possible resources compatible with the activity represented by node x. And for the nodes representing the resources $Resource$ variables will be by default assigned to the respective resources.

While the variable $Resource(x)$ is sufficient to describe a schedule alone, it is weak in the terms of practical use and propagation strength, and we must find a way how to link it with the variables $Successor$. This is the point where we can recall the observation 3.2, that a path $source \rightsquigarrow sink$ represents a schedule for exactly one resource, thus all activities represented by the nodes of the path are allocated to the same resource, thus two activities creating a connection must be allocated to the same resource! We can capture this relationship by a simple constraint $Resource(Successor(x)) = Resource(x)$ or by using the standard $Element$ constraint in the form of $Element(Successor(x), Resource, Resource(x))$. The constraint $Element(Index, VarArray, Value)$ forces the variable at index $Index$ in the variable list $VarList$ to take the value of $Value$. Note that, by introducing $Resource$ variables and binding it with $Successor$ variables we also solve a problem, when a path can have multiple resource nodes - when the last activity node of a sub-path of the path can lead to another different resource node and this process goes on till the last activity node of the path leads back to the first starting resource node. But by enforcing that all the nodes of a path must have the same $Resource$ value, we've avoided this problem.

So far the basic model captures the description of the paths but since the topic of this thesis is about optimization in CP, the model is not sufficient in this sense. We have to incorporate the objective function 3.2 of the problem into the model. As it has been already spoken in section 4.1.5, we can introduce a variable $Cost$ into the model to represent the objective function, on which we will perform optimization. Consequently, we have to introduce a mechanism how to link the $Cost$ variable with other components of the model. For these purposes we will use a similar approach to the one described in section 3.2 to represent the cost of the solution, which covers the cost for resources usage and the penalty costs of every conflicting activity pairs. Thus, with the weight function c defined in section 3.1.0 the $Cost$ variable is defined as follows:

$$Cost = \sum_{1 \le i \le m} c(\, i, Successor(i)) \; + \sum_{\substack{m < i \le m+n \\ j \in AllSuccs(i)}} c(\, i, j)$$

$$AllSuccs(i) = \begin{cases} \{Successor(i)\} \cup AllSuccs\big(Successor(i)\big), & Successor(i) > m \\ \emptyset, & otherwise \end{cases}$$

Model 5.1 A CP model for the Rotation assignment problem CSP-ROTAS-Basic

For a given weighted connection network $G = (V, E, c)$ defined in section 3.1.0 we define a CP model CSP-ROTAS-Basic as the following:

$$Successor(i) = Compatibles(i) \cup \{i\} \qquad \forall\, 1 \le i \le m$$

$$Resource(i) = \{m\} \qquad \forall\, 1 \le i \le m$$

$$Successor(i) = PossibleSuccessor(i) \qquad m+1 \le i \le m+n$$

$$Resource(i) = PossibleResources(i) \qquad m+1 \le i \le m+n$$

$$Element\,\big(Successor(i), Resource, Resource(i)\big) \qquad \forall\, 1 \le i \le m+n$$

$$all\_different(Successor)$$

$$Cost = \sum_{1 \le i \le m} c(\, i, Successor(i)) \; + \sum_{\substack{m < i \le m+n \\ j \in AllSuccs(i)}} c(\, i, j)$$

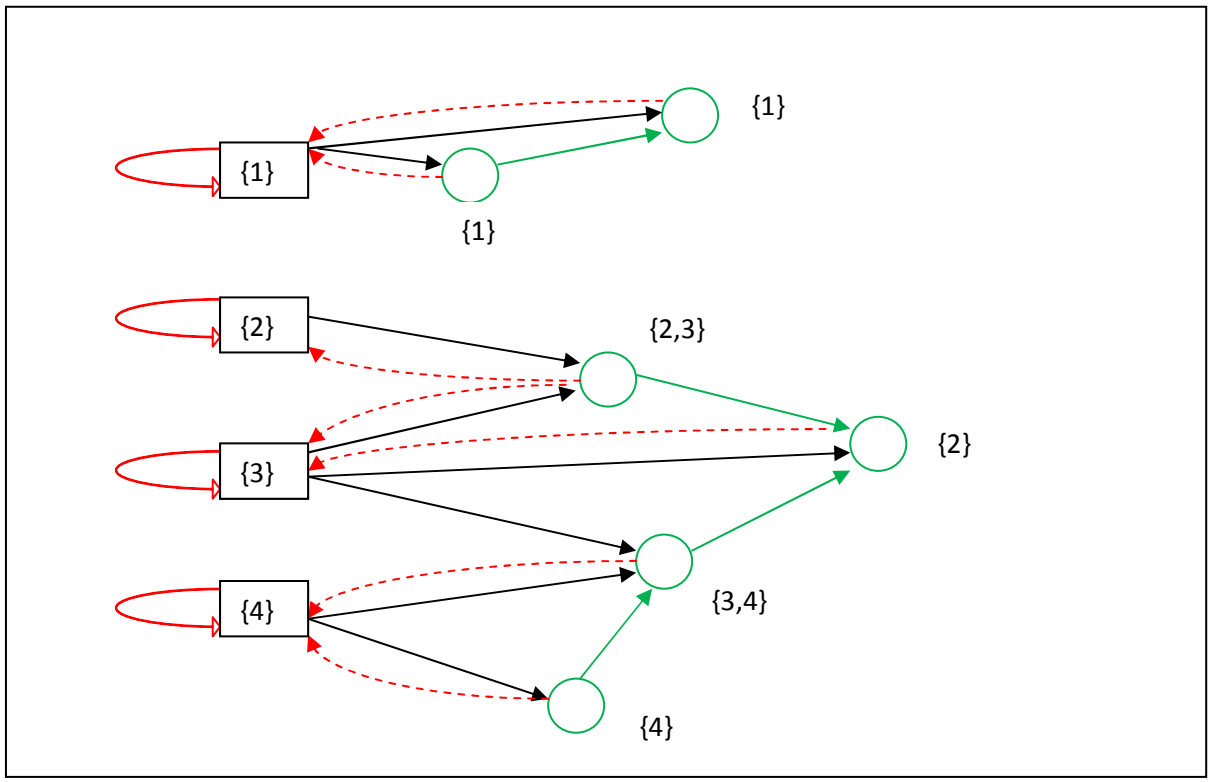
Where

$$Compatibles(i) = \{\, activity\_node\_j \mid (i, activity\_node\_j) \in E \,\}$$

$$PossibleResources(i) = \{\, resource\_node\_r \mid (resource\_node\_r, i) \in E \,\}$$

$$PossibleSuccessor(i) = \{j \mid (i, j) \in E\} \cup PossibleResources(i)$$

$$AllSuccs(i) = \begin{cases} \{Successor(i)\} \cup AllSuccs\big(Successor(i)\big), & Successor(i) > m \\ \emptyset, & otherwise \end{cases}$$



An example of the CP model derived from figure 3.1. The newly introduced arrows leading back to the resource are in red, the source and sink vertices and arcs are gone. The domains of $Resource$ variables are in brackets. $Successor$ variables are represented by the arrows implicitly.

Figure 5.1

5.2 Searching for the best solution

5.2.1 Search heuristics

As we have already talked about, the search heuristics can have a huge impact on the performance a search. In CP, the search heuristics include variable and value ordering. What are good heuristics for our case? Although I tried to branch on the Successor variables and I tried to use the often recommended way ordering, i.e., by selecting a variable with the smallest domain, and the smallest value, it was not possible to get a solution in a reasonable time. Thus, I‘ve chosen different search heuristics.

Firstly, I've chosen to branch on the $Resource$ variables, i.e. for each activity we choose to which resource it will be allocated to. I've decided to do so in order to reduce the number of alternatives at each point for the search process, i.e. by the size of the domains of the variables. Because the number of resources to which an activity can be allocated to is much smaller than the number of possible successor activities, which is in our case virtually all activities that have later start time. The choice was also proven experimentally, when branching over the $Successor$ variables a solution was not reached even after 30 minutes for a weekly schedule, whereas with the same data when branching over the $Resource$ variables solution was reached in less than 5 seconds.

Regarding the question which $Resource$ variable is selected for assignment first, I've chosen the approach of selecting a variable representing the activity with the earliest start time first, so when a contradiction occurs, e.g., a schedule‘s cost is too high, we can surely say that is because of the position of the earlier starting activities and backtrack accordingly in the search process. Thanks to this approach we will see later that it will help us further in propagating constraints.

For value selection, i.e., to which resource we will allocate an activity I've observed these cases:

- Optimal schedules with no cost have also connections with zero costs
- Although we can face alternatives with the same cost, those alternatives can differ in the terms of pre-assigned activities (recall *Reservation activities*), which can influence the outcome of scheduling. That is, it doesn't matter to which resource we assign a given activity x cost-wise, but the resources have different pre-assigned activities that start later than the activity x, and the choice to which resource we will assign x to can, of course, influence the future choices. To illustrate the situation let us look at figures 5.1 and 5.2. below:

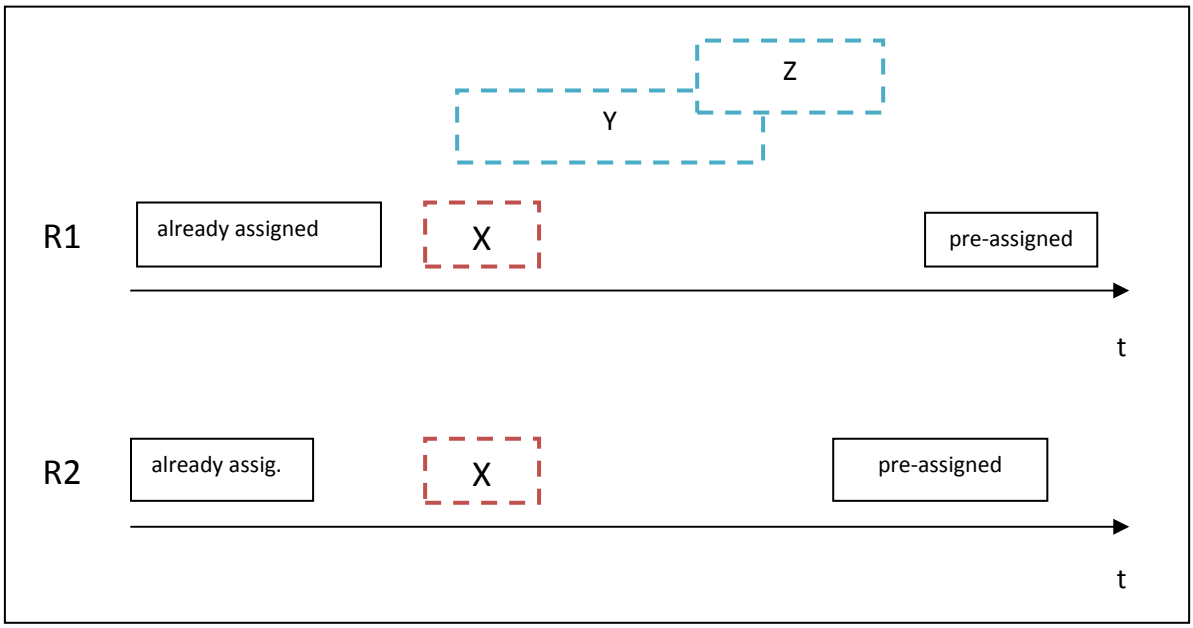


Figure 5.2

In figure 5.2, if we assign x to resource $R2$, then in order to avoid a penalty we will assign y to $R1$ and then there's no place to assign z to avoid a penalty. The situation would be completely different if we assign x to resource $R1$ first, then y to $R2$, then z to $R1$. A better strategy here would be that, for x we should have chosen a resource where "larger room" between x and the next pre-assigned activity is, so other later activities can fit in.

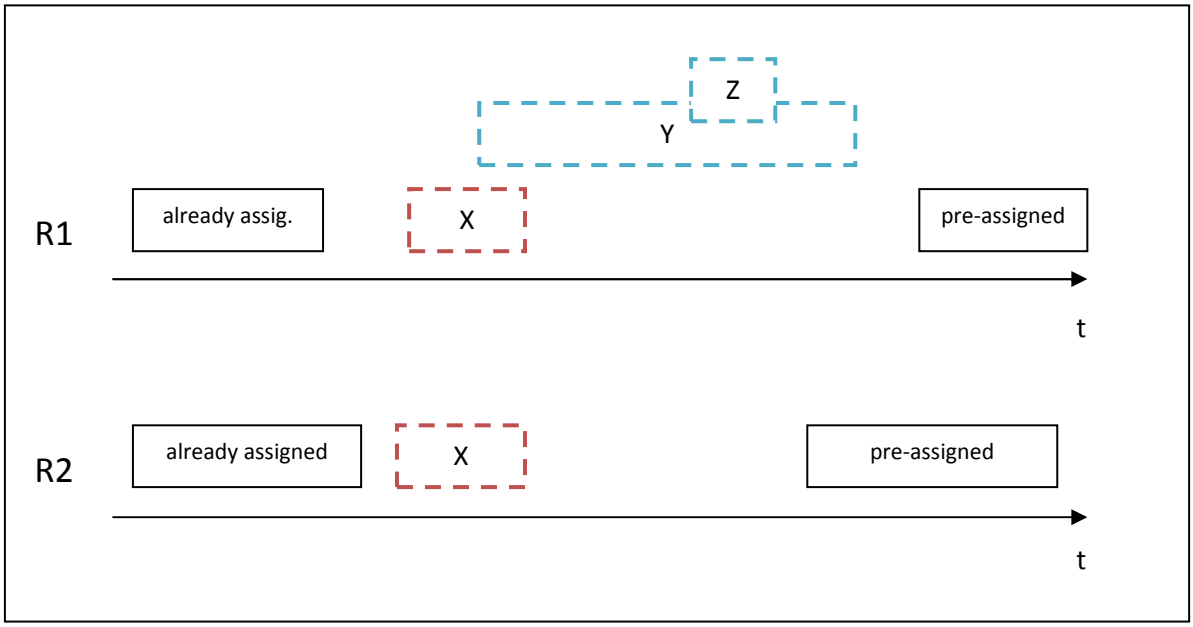


Figure 5.3

However, in a different situation at Figure 5.3, if we assign x to $R1$ because we want to keep the maximum time distance between x and the next pre-assigned activity, then there will be no resource to move y to without violating the constraints. The situation would be better if we moved x to $R2$ first. So, a better strategy here is to choose a resource where there is "smaller room" between a given activity and the next pre-assigned activity in order to save room for longer activities.

Although the two mentioned strategies are completely opposite, we should not choose only one of them. That's why our approach regarding value ordering is that we lexicographically order the alternatives according the cost first and then according to the time distance between a given activity and its next pre-assigned activity. Hence, when considering the alternatives, we branch on the alternative with the least cost first and then later on, we branch on the remaining alternatives; when facing with alternatives where some of them have the same the least cost, we branch on the alternative with the least cost and the largest time distance first, then we select the alternative with the least cost but with the smallest time distance, and later on we branch on the remaining alternatives.

5.3 Implementing and improving the basic CP model

5.3.1 More efficient constraint propagation

If we recall the brief introduction of constraint propagation and consistency techniques in section 4.2, then we would realize that the whenever the domain of a variable changes - be it a removal of values from the domain, a change of the bound of the domains, a variable assignment - it will trigger a cascade of constraint propagations and consistency checks on all the variables linked with the particular variable by some constraint. This process is necessary in order to propagate the changes as much as possible. However, sometimes such process doesn't have a real effect on reduction of the domains of variables and the computational overhead involved in the process can have a significant impact on the performance of a solving algorithm.

Notation: From now we will denote construction: **if** $premise\ event$ **then** $constrain$ with the meaning that when an event, a condition described in $premise\ event$ holds a restriction, a constraint described in $constraint$ is posted, added to the model.

An exemplary improvement can be achieved if we look closer how the constraint $Element\ (Successor(i), Resource, Resource(i))$ works. The constraint is triggered whenever there's a change in domain $D(Resource(i))$ or $D(Successor(i))$, then the constraint would go through all $x \in Successor(i)$ to check out whether $Resource(x) \cap Resource(i) \neq \emptyset$. This process can be triggered many times without having a real effect on constraint propagation. This was also observed by Kilborn [4] and Grönkvist [5], who implemented a similar procedure in their models.

To avoid such overhead the constraint $Element$ is removed from the model and we define a new constraint $TunnelRSC\ (Resource, Successor, Cost)$ as follows:

$$\textbf{if}\ Resource(i) = r\ \textbf{then}\ Element\ (Successor(i), Resource, r)$$

$$\textbf{if}\ Successor(i) = j\ \textbf{then}\ Resource(i) = Resouce(j)$$

5.3.2 Faster tunneling

Observation 5.1: With the variable ordering described in section 5.2.1, it holds that if activity y is to be assigned to resource r and all activities starting before y are already allocated and there's an activity x, the last activity assigned to r having the starting time earlier than y, then y must be the successor activity of x. More formally:

$$\textbf{if}\ Resource(y) = r \wedge \forall a\ s_a < s_y\ Resouce(a) \text{ is assigned } \wedge$$

$$\exists x\ Resource(x) = r, Successor(x) \text{ not assigned}, s_x < s_y \wedge$$

$$\nexists z\ Resouce(z) = r, s_x < s_z < s_y$$

$$\textbf{then}\ Successor(x) = y$$

Proof: From the premise we know that x is the last activity assigned to r starting before y and thanks to the variable ordering we know that all other activities, that will be assigned to r after assignment of y start later than y, and therefore, the activity that starts immediately after x must be y. Note that, although there might exist an activity that has starting time equal to starting time of x, but after assignment of y to r, the activity cannot be assigned to r anymore due to the constraint $C6$.

Thanks to this observation we can extend the functionality of the $TunnelRSC$ constraint so that when the premise of the observation holds we can for sure tunnel this event to the $Successor$ variables as $Successor(x) = y$. Although this is a small extension of the $TunnelRSC$ constraint, the performance of the model has been improved by average 5 times!

5.3.3 A different value ordering, introduction of redundancy to strengthen constraint propagation

If we look closer at the way how the constraints $Element\left(Successor(i), Resource, Resource(i)\right)$ and $all\_different(Successor)$ combined with the variable ordering work, we can see that there's a lot of inefficiency!

Firstly, when assignment $Successor(x_i) = x_j$ occurs, this change affects (reduces) the domains of other unassigned $\{ Successor(x_k) \mid k > j \}$ variables very poorly, since none of them have x_j in their domains (variables $Successor(x_e)$, $e < i$ might have had x_j in their domains but they are mostly assigned already).

Secondly, when assignment $Successor(x_i) = x_j$ occurs, the domains of $\{ Successor(x_k) \mid k > j \}$ variables are mostly not assigned yet, thus propagation of variable $Cost$ as defined cannot occur yet.

For these reasons I've changed the variable ordering in such a way that instead of selecting the variable representing the earliest activity first, the variable corresponding to the last starting activity is selected. It may sound unintuitive, but it fixes the above mentioned shortcomings:

- when $Successor(x_i) = x_j$ occurs, many other variables $Successor(x_e), e < i$ are still unassigned and might have x_j in their domains as well, so the constraint $all\_different(Successor)$ can do its job pretty well here.
- when assignment $Successor(x_i) = x_j$ occurs, the domains of $\{ Successor(x_k) \mid k > j \}$ variables are mostly assigned already, thus propagation of the cost variable is efficient

But now with the new variable ordering we can see that our use of constraint $Resource(i) = r \rightarrow Element\ (Successor(i), Resource, r)$ described in section 5.3.1 is pretty useless, because $Resource(Successor(i))$ is probably already assigned since $Successor(i)$ represents either an activity starting after i or it is a resource. With the new variable ordering it would be more effective to state that activities starting before i must be allocated to the same resource as i. This is the reason why we introduce new variables into the model – $Predecessor$, which for each activity i express the

activity that starts right before i, i.e., in our graph model it is the preceding node of a particular node on a path. Now, we can adapt the constraint $TunnelRSC$ into:

$$\textbf{if } Resource(i) = r \textbf{ then } Element\ (Predecessor(i), Resource, r)$$

$$\textbf{if } Resource(i) = r \textbf{ then } Element\ (\text{Successor(i), Resource, r})$$

$$\textbf{if } \text{Pre}decessor(j) = i \textbf{ then } Resource(i) = Resouce(j)$$

And bind the $Predecessor$ and $Successor$ variables by:

$$Predecessor(j) = i \leftrightarrow Successor(i) = j$$

Furthermore, with the new variable ordering we cannot use observation 5.1, but of course we can still apply an analogical idea, we just need to adapt it to:

if $Resource(x) = r \wedge \forall a\ s_x < s_a\ Resouce(a)$ is assigned $\wedge$
$\exists y\ Resource(y) = r, Predecessor(y)$ not assigned, $s_x < s_y\ \wedge$
$\nexists z\ Resource(z) = r, s_x < s_z < s_y$
then $Predecessor(y) = x\ \wedge Successor(x) = y.$

One might ask, why don't we also post constrain $all\_different(Predecessor)$ as it might also help the propagation. This is because of the same reasons mentioned above. When assignment $Predecessor(j) = i$ occurs, most of the unassigned variables are $Predecessor(e), e < i$, which of course don't have i in their domains $D(Predecessor(e))$ and therefore the overhead caused by constraint a$ll_different(Predecessor)$ doesn't pay off.

5.3.4 Getting a better initial bound

What if we take the soft constraint $C1$ as a hard one? Then we might not get an optimal solution, but thanks to this restriction we can make the search space much smaller while an achieved solution is still acceptable and then we can use this possible solution as a higher bound for the original problem (with $C1$ as a soft constraint). With $C1$ as a hard constraint in mind, we can state that any overlapping activities must be allocated to different resources and they can be neither successors nor predecessors of each others. And so, we can add these constraints into the model:

$\forall a \in A$:

$$(a, o)\ \notin C1 \quad \rightarrow \quad Resource(a) \neq Resource(o) \qquad \forall\ o \in A$$

$$(a, o)\ \notin C1 \quad \rightarrow \quad o \notin Successor(a) \qquad \forall\ o \in A$$

$$(a, o)\ \notin C1 \quad \rightarrow \quad a \notin Predecessor(o) \qquad \forall\ o \in A$$

After adding these constraints, the performance of the search improved so significantly that it is enough to run the model with these constraints added with 1/4 of allowed time first and then, if a result is reached, we use the result as a better initial bound for further optimizations.

5.3.5 Implementing the cost function

Although the $Cost$ variable is nicely formulated, but due to possible inefficiencies when every small change in the domains of $Successor$ constraint propagation can be triggered without any real effect on the $Cost$ variable, in practice we opted to implement propagation of the cost in a different way. We extend the constraint, or rather in this case the propagation procedure, $TunnelRSC\ (Resource, Successor, Cost)$ to continuously update the value of the $Cost$ variable. The constraint increases the lower bound of the $Cost$ variable after each assignment of $Successor$ variables, i.e., after an assignment, say, $Successor(x) = y$, it increases the lower bound of $Cost$ by a sum of costs of connections between x and its succeeding activities. Recall that, this is possible thanks to the new variable ordering where all variables $Successor$ representing activities starting after x are assigned already in the most situations. Also, we can observe that, if activity x doesn't violate constraints $C1$ or $C2$ with a particular activity, say z, then it doesn't violate the constraints with any other activities starting after z either. We can more formally describe the process as:

$$Cost = < 0, \infty)$$

$$\textbf{if } Successor(x) = y \wedge \forall a\ s_x < s_a\ Successor(a) \text{ is assigned}$$

$$\textbf{then } \min(Cost) \coloneqq \min(Cost) + \sum_{y \in ConftSucs(x)} c(x, y)$$

where

$$ConftSucs(x) = \begin{cases} \{Successor(x)\} \cup ConftSucs\big(Successor(x)\big), & c\big(x, Successor(x)\big) \neq 0 \\ \emptyset, & otherwise \end{cases}$$

5.3.6 Better representation of the cost for using resources

Although the cost for using a resource is computed in $TunnelRSC$ and is represented together with other costs by the $Cost$ variable, it would be handy if we have a way to count and restrict resource usage directly - for instance, we have some good solutions with the same cost, e.g., in some solutions there are more violations of the soft constraints, whereas some solutions have less violations of the soft constrains but uses more resource, then we would prefer the solution with the least resource use instead. The motivation of preferring the minimized resource usage is that sometimes we don't know exactly how costly it might be to operate a resource and only some estimations can be provided while the real cost can be much higher due to many possible complications. However, back to the problem, how can we quickly recognize whether a resource is used or not? If we remember the definition

of the first m artificial variables $Successor$ which represent resources, then we would recall that if $Successor(i) = i, 1 \leq i \leq m$, then it would imply that resource i is not used. And by simply counting how many such situations occurred we can tell how many resources are not used. In the model we introduce a new variable $NotUsedResources$ formally as follows:

$$NotUsedResources = \sum_{1 \leq i \leq m} \delta(Successor(i) = i)$$

$$\delta(b) = \begin{cases} 1 & b = true \\ 0 & b = false \end{cases}$$

5.4 Combining local search and CP

Because of the non-polynomial complexity nature of our problem, unless NP≠P, it is difficult to guarantee time needed to solve problems with large data inputs. For example, in average it takes about 5 seconds to find a good schedule for a week with around 250 activities, but for a month it takes 6 minutes to find a good schedule. To cope with this problem, I've decided to make a tradeoff between the run time requirements of the model and the quality of a solution by applying local search. Local search is a common term for heuristics which gradually improve solutions by small local changes. The advantage of local search is that it can find a local optimum fast on smaller data input, but inherently, the disadvantage is that while we can reach a local optimum, it doesn't mean that subsequent local optima can converge to a global optimum. However sometimes, actually this is our case, the main goal is not to get a global optimum but a good solution in a short time.

5.4.1 Large neighbourhood search

One of the possible ways how to perform a local search is that one selects a part of an initial solution over which changes are made in order to gradually improve the initial solution. This approach, called $Large\ neighbourhood\ search$, was proposed by P. Shaw [17] and was successfully applied, for instance, for solving the Vehicle routing problem [18].

There are many ways how to implement the Large neighborhood search. In this case, I've chosen the way by splitting the time range of input data into consecutive smaller periods, called $windows$, in each of them a local improvement is performed.

Because of the nature of our problem I had to make some modification to have the approach working well. Because activities can lie in the border between two neighbor windows, a good solution in the first window can prevent us to have an even better solution in the second window due to placing of the activities in the border. Therefore, I let the windows to overlap each other by a small portion. Also, thanks to the fact that our CP model is build over a graph model, I can easily fix all the activities that are optimized previously and lie outside of the actual window to their already allocated resources, and then let the CSP solver [7] to find a solution in the actual window, that together with the fixed activities must form a better solution than a known best solution. In our approach we make several

iterations (passes) over an input, and for each pass we increase the size of windows gradually (until the size reaches a maximal value). In the end, after the pass with the largest possible window, we decrease the size of windows to just one day to try to resolve complex conflicts that were not solved in previous passes due to larger input at once.

Thanks to applying this customized approach the performance of the search for large input data improved significantly, the run time of the program has decreased by one third. The whole algorithm is written in a pseudo code below as algorithm 5.1.

5.4.2 Search methods

I should mention that the CP package that I use is called Gecode [7]. It offers two search engines for optimization of a CSP. The first is called Restart and the other one is Brach-and-Bound (BB). Roughly spoken, in Restart once a solution is found, the search process is restarted using the cost of the previous solution to bound the search for better solutions; in BB once a solution is found, the cost of the solution is posted to other partial, not completely solved instances yet to restrict them to be better and the search process is continued.

I have chosen for the BB approach, because principally it uses smaller computation overhead than Restart.

5.5 The complete approach for the Rotations Assignment problem based on CP

Finally, in this section I present a final CP model and a complete solution for the Rotations assignment problem as well.

Model 5.2: A CP model for the Rotation assignment problem CSP-ROTAS

For a given weighted connection network $G = (V, E, c)$, defined in section 3.1.0, we define a CP model *CSP-ROTAS* as the following:

$$Successor(i) = Compatibles(i) \cup \{i\} \qquad \forall\, 1 \le i \le m$$

$$Resource(i) = \{m\} \qquad \forall\, 1 \le i \le m$$

$$Successor(i) = PossibleSuccessor(i) \qquad m+1 \le i \le m+n$$

$$Predecessor(i) = PossiblePredecessor(i) \qquad m+1 \le i \le m+n$$

$$Resource(i) = PossibleResources(i) \qquad m+1 \le i \le m+n$$

$$Cost = \sum_{1 \le i \le m} c\big(i, Successor(i)\big) + \sum_{\substack{m < i \le m+n \\ j \in AllSuccs(i)}} c(i, j)$$

$$NotUsedResources = < 0, m >$$

$$NoOverlap(Successor, Resource)\ *$$

$$all\_different(Successor)$$

$$TunnelRSC\ (Resource, Successor, Cost)$$

$$Predecessor(j) = i \leftrightarrow Successor(i) = j$$

Where

$$Compatibles(i) = \{\, activity\_node\_j \mid (i, activity\_node\_j) \in E \,\}$$

$$PossibleResources(i) = \{\, resource\_node\_r \mid (resource\_node\_r, i) \in E \,\}$$

$$PossibleSuccessor(i) = \{j \mid (i, j) \in E\} \cup PossibleResources(i)$$

$$PossiblePredecessor(i) = \{j \mid (j, i) \in E\} \cup PossibleResources(i)$$

$$AllSuccs(i) = \begin{cases} \{Successor(i)\} \cup AllSuccs\big(Successor(i)\big), & Successor(i) > m \\ \emptyset, & otherwise \end{cases}$$

$$NotUsedResources = \sum_{1 \le i \le m} \delta(Successor(i) = i)$$

$$\delta(b) = \begin{cases} 1 & b = true \\ 0 & b = false \end{cases}$$

$TunnelRSC\ (Resource, Successor, Cost)$ is defined as:

if $Resource(x) = r \wedge \forall a\ s_x < s_a\ Resouce(a)$ is assigned $\wedge$

$\exists y\ Resource(y) = r, Predecessor(y)$ not assigned, $s_x < s_y\ \wedge$

$\nexists z\ Resouce(z) = r, s_x < s_z < s_y$

then $Predecessor(y) = x\ \wedge Successor(x) = y$

if $Resource(i) = r$ **then** $Element\ (Predecessor(i), Resource, r)$

if $Resource(i) = r$ **then** $Element\ (Successor(i), Resource, r)$

if $Predecessor(j) = i$ **then** $Resource(i) = Resouce(j)$

$NoOverlap$ is defined as:

$\forall a \in A$:

$$(a, o) \notin C1 \quad \rightarrow \quad Resource(a) \neq Resource(o) \qquad \forall\, o \in A$$

$$(a, o) \notin C1 \quad \rightarrow \quad o \notin Successor(a) \qquad \forall\, o \in A$$

$$(a, o) \notin C1 \quad \rightarrow \quad a \notin Predecessor(o) \qquad \forall\, o \in A$$

Note: $NoOverlap$ is not always added to the model, it depends on whether we strictly want a schedule without overlapping of activities or to check out existence of such schedule.

Algorithm 5.1: CP-LOCALSEACH-ROTAS

data

R : a set of resources, A : a set of activities, C : constraints C on schedules
window : time window
ϖ_{max} : maximal size for time window, ϖ_b : basic size for time window
$[t_s, t_e]$: the whole planning period
$[t_s^{curr}, t_e^{curr}]$: current planning period
Δ_δ : look up range, Δ_t : multiplier for timeout, Δ_w : multiplier for window
T : maximal timeout allowed for solution finding
τ_b : a constant for default timeout
τ_{1d} : a constant for timeout for one day fine tuning
MINRESOURCE : a flag to minimize resource use only
MINCOST : a flag to minimize the cost

procedure CP-LOCALSEACH-ROTAS
$G \longleftarrow$ createConnectionNetwork(R, A, C)
CSP-ROTAS $\longleftarrow$ createCPModel(G)
BestSol $\longleftarrow$ Fix all activities on their default resources according to the input *
$[t_s^{curr}, t_e^{curr}] \longleftarrow [t_s, t_s + window]$
$T \longleftarrow \tau_b$; $window \longleftarrow \varpi_b$
if *BestSol* has some violations of constraints C **then repeat**
Unfix all activities inside $[t_s^{curr} - \Delta_\delta, t_e^{curr} + \Delta_\delta]$ on G
CSP-ROTAS $\longleftarrow$ createCPModel(G)
tmpSol $\longleftarrow$ Solve(*CSP-ROTAS*, *BestSol*, MINCOST, T)
if *tmpSol* $\neq$ FAIL **then**
BestSol $\longleftarrow$ *tmpSol*
Store *BestSol* to G, Fix all activities of G according to *BestSol*
$[t_s^{curr}, t_e^{curr}] \longleftarrow [t_s^{curr} + window, t_e^{curr} + window]$
if $t_s^{curr} \geq t_e$ **then**
if *window* = 1 day **then**
$window \longleftarrow t_e - t_s$
$T \longleftarrow$ all time left
else if $window \geq \varpi_{max}$ **or** $window \geq t_e - t_s$ **then**
$window \longleftarrow$ 1 day; timeout $\longleftarrow \tau_{1d}$; $\Delta_\delta \longleftarrow 0$
do not increase *window* and T any more later on
else
$T \longleftarrow T * \Delta_t$; $window \longleftarrow window * \Delta_w$
$[t_s^{curr}, t_e^{curr}] \longleftarrow [t_s, t_s + window]$
until BestSol has no violation of C **or** BestSol is good enough **or** no time left

if *BestSol* has no violation of C **or** is good enough **then**
$T \longleftarrow$ all time left
Unfix all activities inside $[t_s - \Delta_\delta, t_e + \Delta_\delta]$ on G
CSP-ROTAS $\longleftarrow$ createCPModel(G)
tmpSol $\longleftarrow$ Solve(*CSP-ROTAS, BestSol*, MINRESOURCE, T)
if *tmpSol* $\neq$ FAIL **then**
BestSol $\longleftarrow$ *tmpSol*
return *BestSol*
end CP-LOCALSEACH-ROTAS

procedure Solve(*CSP-ROTAS*, *BestSol*, runMode, T)
    if runMode = MINRESOURCE **then**
        $CSP\text{-}ROTAS^X \longleftarrow CSP\text{-}ROTAS +$
            restrict $NotUsedResources > BestSol.NotUsedResources$
        $CSP\text{-}ROTAS^X \longleftarrow CSP\text{-}ROTAS^X$ + restrict $Cost \leq BestSol.Cost$
    else
        $CSP\text{-}ROTAS^X \longleftarrow CSP\text{-}ROTAS$ + restrict $Cost < BestSol.Cost$

    $CSP\text{-}ROTAS^Y \longleftarrow CSP\text{-}ROTAS^X$ + *NoOverlap* constraint
    tmpSol ⟵ Find a solution for $CSP\text{-}ROTAS^Y$ using **BB** search with 1/3*T timeout
    if *tmpSol* ≠ FAIL **and** *tmpSol.Cost* < *BestSol.Cost* **then**
        $CSP\text{-}ROTAS^X \longleftarrow CSP\text{-}ROTAS^X$ + restrict $Cost < BestSol.Cost$

    BestSol ⟵Find a best solution for $CSP\text{-}ROTAS^X$ using **BB** search with 2/3*T
    if *BestSol* = FAIL **then**
        BestSol ⟵ *tmpSol*

    return *BestSol*
end Solve

Note: Although it is not important for the CP or the graph model, in practice in our case the activities are from the input by default assigned to some resources. The character and the quality of the default assignment can vary and can be unpredictable. But since we don't have any solution in the beginning, we can use it as a starting point (something is still better than nothing).

Chapter 6

Computation results

In this chapter I would like to present the experimental results of my work.

Testing environment:

Hardware:

CPU Intel Xeon 3.2GHz, 2GB RAM. OS Windows server 2003 R2

Data:

During the work I had an opportunity to access to two databases provided by company The Kite, s.r.o [25]:

1. Database AVES is artificially generated, has small numbers of reservation activities and has been pre-optimized, so it is "easy" to optimize
2. Database AVESX has data from real life which includes many reservation activities, has been deliberately so it can have problematic parts with unsolvable conflicts

Software:

Optimization module WK version 4.1. R02, build 1012 from June 20 2010

In the optimization module WK, in which was working with, I had access to 3 of 4 other algorithms solving the same problem:

- Algorithm "FIFO/LIFO" is based on proprietary search heuristics and backtracking.
- Algorithm "Complex Solution" is based on proprietary search heuristics and the Column generation technique.
- Algorithm "Aircraft usage reduction" is also base on proprietary search heuristics and the Column generation technique with stronger emphases on optimizing the resource usage.
- The fourth one is the most complex one. Although I didn't have access to this algorithm directly, it is known that it produces the best results of the three above but is very slow - the algorithm is usually let run over night.

All these algorithms have been developed over years and the first three are already mentioned in the introduction chapter, they represent the standard approach in the industry by using linear programming techniques. Although they are proprietary algorithms, we can observe their working characteristic from the results, see the remarks on the results below. In the following paragraphs I'll present comparison results with the 3 accessible algorithms.

Additionally, I tested three versions of the CP approach:

- ROTAS Final - This is the full model
- ROTAS No-Tunnel - This is the full model without the tunnel constraint described in section 5.3.2
- ROTAS No-Local - This is the full model without the large neighbourhood search
- ROTAS Simple - This is completely the basic model 5.1 without search heuristics, variable ordering and local search. Only the smallest domain size and the smallest value of $Resource$ or $Successor$ variables approach of was used.

Note that, the ROTAS No-Tunnel version is tested in each case twice to demontrate the quality of the solution within the same run time given to other ROTAS models and needed run time in order to achieve the same results as other models.

Testing parameters:

During the testing the following the cost scheme is defined for functions from section 2.3.0: The constants can be defined by the user.

- For $\pi^a(r)$ c_r = 6000 $\forall r \in R$
- For $\pi^C(a_i, a_j)$: $C_a{}^1 = 0, C_b{}^1 = 100, C_c{}^1 = 100, C_d{}^1 = 0,\ Lt^1 = 0$
  $C_a{}^2 = 50, C_b{}^2 = 5, C_c{}^2 = 5, C_d{}^2 = 0,\ Lt^2 = 0$

Table 6.1: Results on AVES for Aircraft type 319

Period in days	Resources in total	Activities total	Reserve activities
7 (17.-23.6.'10)	16	262	0

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	9	61350
Complex Solution	1	10	67300
Aircraft reduction	less than 1	9	61350
ROTAS No-Tunnel	5	8	FAIL
ROTAS No-Tunnel	45	8	56000
ROTAS No-Local	5	8	56000
ROTAS Simple	1000	-	-
ROTAS Final	5	8	56000

Period in days	Resources in total	Activities total	Reserve activities
14 (24.6-7.7.'10)	16	429	0

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	9	68200
Complex Solution	1	10	74150
Aircraft reduction	1	9	68200
ROTAS No-Tunnel	26	8	64200
ROTAS No-Tunnel	45	8	63300
ROTAS No-Local	26	8	63300
ROTAS Final	26	8	63300

Period in days	Resources in total	Activities total	Reserve activities
30 (17.6-16.7.'10)	16	664	0

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	9	83450
Complex Solution	1	10	89400
Aircraft reduction	1	9	83450
ROTAS No-Tunnel	170	8	79900
ROTAS No-Tunnel	70	8	83650
ROTAS No-Local	70	8	79450
ROTAS Final	70	8	79900

From the results we can see that the three other algorithms are much faster than ours. This is due to their working characteristics, which takes an already optimized schedule with numbers of small conflicts and they focus on solving these conflicts only, in this case most of them solve the conflicts by just putting the conflicting activities on new resources. Obviously the approach gives the high speed for the algorithms but overall, using new resources comes, of course, costly. That's where our "ROTAS Final" algorithm trades off some amount of time spending on solving for better utilization of resources resulting in better schedules' costs.

Comparing ROTAS Final with other ROTAS models we might find it interesting that the No-Local version performs slightly better in the period of 30 days where the cost is even lower than the ROTAS-Final version, this is because it spends the whole given time to solve the month and in case of AVES data which are "easy" to solve it pays off. From the results we can also see that the No-Tunnel version is very behind others in the terms of the run time. I also tested the ROTAS-Simple version on the period of a week but when no solution was provided even after 1000 seconds, I think it sufficient to demonstrate that it is practically unusable comparing with others.

Table 6.2: Results on AVESX for Aircraft type 320

Period in days	Resources in total	Activities total	Reserve activities
7 (27.10.-2.11.'10)	11 (3 are blocked)	429	157

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	-	FAIL-4unsched. acts.
Complex Solution	1	-	FAIL-1unsched. acts.
Aircraft reduction	1	8	218427650
ROTAS No-Tunnel	300	8	180028950
ROTAS No-Tunnel	43	8	238715700
ROTAS No-Local	43	8	180028950
ROTAS final	43	8	180028950

Period in days	Resources in total	Activities total	Reserve activities
14 (3.11.-17.11.'10)	11 (3 are blocked)	590	194

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	-	FAIL-4unsched. acts.
Complex Solution	1	8	257591700
Aircraft reduction	1	8	257591700
ROTAS No-Tunnel	65	8	239835750
ROTAS No-Local	65	8	256942850
ROTAS final	65	8	239835750

Period in days	Resources in total	Activities total	Reserve activities
31 (26.10.-25.11.'10)	11 (3 are blocked)	1038	352

Algorithm	Run time in seconds	Used resources	Schedule cost
LIFO/FIFO	1	-	FAIL-9unsched. acts.
Complex Solution	2	8	286822150
Aircraft reduction	2	8	288156150
ROTAS No-Tunnel	1000	8	300542700
ROTAS No-Tunnel	252	8	300542700
ROTAS No-Local	252	8	566059200
ROTAS final	252	8	253790950

Since data from AVESX are difficult in the way that is has many reservation activities and some parts cannot be solved without having some conflicts left, the approach of other algorithms did not work well on it. The “LIFO/FIFO” and even once “Complex Solution” algorithms failed to find allocations for all activities. The LP algorithms are optimized for speed and they expect that the input is somewhat good so they can make some changes by focusing on and solving the conflicting activities and left the other activities the same as much as possible. But sometimes it is not possible and it is required to make significant changes in order to improve the cost of a schedule, and this is where our “ROTAS Final” model excels. It trades off a reasonable time (but not that all night) spent on problem solving for getting much better results. Considering that it takes only a little more than 4 minutes to solve a schedule for a month, it is a good compromise.

The ROTAS models perfectly exhibit their nature on AVESX data. The ROTAS No-Local version was on par with the Final version on AVES data, but with AVESX where the input difficulty and size rise the quality of solutions deteriorate quickly. But note that, it does not imply that the No-Local version is worse than the ones with local search. Theoretically, since the No-Local version uses the Branch-and-Bound method which completely explores the search space, it can eventually provide a global optimum. The reason of the No-Local version being worse here is that it was provided a limited run time. Interesting is also period 3.11.-17.11.’10 when the ROTAS No-Tunnel version performs on par with others. I think this is because there were not many changes possible to perform. Otherwise, the ROTAS No-Tunnel version is again way behind others in the term of the run time.

Note that, although due to the NP-Hard nature of the problem we cannot guarantee the time complexity of the model, unless NP = P, but from the experimental results (and practical use) we can see that the time complexity is at average about $O(n^2)$ where n is number of the activities.

Chapter 7

Conclusion

The field of the interest of this thesis has been extensively investigated. However, unlike other works mentioned in the introduction chapter, my work is mainly different in the way that it focuses on the optimization aspect of the problem within the framework of Constraint programming. And as far as I know, it is the only model that deals with the optimization version of the Rotation assignment problem within the framework of CP.

As showed in the introduction chapter, solving approaches for the problem evolves from one to another. The CP model presented in this thesis shares similarities in concept with other existing works ([1], [4], [5], [14]), but as its focus is different, the design of the model had to adapt and evolve significantly in order to meet its goal.

In this thesis I formally presented the optimization version of the problem, then an abstract model was defined and on which a solving CP model and a customized local search technique was designed. In order to perform the optimization in CP only, many associated issues needed to be solved and designed. Among the most important components we can name:

- the search heuristic – variable choice, variable and value ordering, which is crucial to the performance of the solution finding search.
- the efficient representation and implementation of the objective function,
- the tight tunneling between variables
- the practically more efficient use of constraints,
- the way to achieve a better initial bound for the solution
- and the way of better expressing resource usage.

In this thesis I've showed that it is possible to solve the optimization version of the Rotation assignment problem as a CSP, but not only that, from the computation results we can conclude that the CP approach and the solving algorithm can compete with other existing approaches, especially as a good balance between the run time and the quality of solutions.

From the experimental results we can also observe the characteristics of CP and LP. While CP is strong in exhaustive and fast scanning through the search space so we can get good results "from the scratch" in a reasonable time, LP techniques are fast and strong in making small tuning changes of an existing solution, the last steps towards the optimum.

In the end, I should mentioned that in the future, this work will be continually developing and evolving over time, as it is supported by real life application and is deployed to solve real life situations.

Bibliography

[1] M. Pinedo: Planning and Scheduling in Manufacturing and Services, Springer New York, 2005.

[2] R. Barták: Lecture notes on Constraint Programming. Faculty of Mathematics and physics, Charles University, Prague. http://kti.mff.cuni.cz/~bartak/constraints/index.html.

[3] F. Rossi, P. van Beek, T. Walsh (eds.): Handbook of Constraint Programming, Elsevier, 2006.

[4] E. Kilborn: Aircraft assignment: A Basic CP Model. Master thesis, Chalmers University of Technology, Sweden, http://www.math.unipd.it/~frossi/kilborn.ps.

[5] M. Grönkvist: A Constraint Programming Model for Tail Assignment. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 3011 (2004), Springer, 142-156.

[6] D. Sosnowsk, J. Rolim: Fleet Scheduling Optimization: A Simulated Annealing Approach. Practice and Theory of Automated Timetabling III, Springer Berlin, 2079 (2001), 227-241.

[7] Gecode, a toolkit for developing constraint-based systems and applications, ver3.3.1, www.gecode.org.

[8] Ram Gopalan and Kalyan T. Talluri: Mathematical models in airline schedule planning: A survey. Annals of Operations Research, 76 (1998), Springer, 155 – 185.

[9] M. Haouari, H. D. Sherali, F. Z. Mansour, N. Aissaoui: Exact approaches for integrated aircraft fleeting and routing at TunisAir. Computational Optimization and Applications Journal, 7 October 2009 (Online version), Springer Netherlands, www.springerlink.com/content/r83t1249j4366870.

[10] E. A. Rich: Automata, Computability and Complexity: Theory and Applications, Prentice Hall, 2007.

[11] R. Backofen, S. Will, E. Bornberg-Bauer: Application of constraint programming techniques for structure prediction of lattice proteins with extended alphabets. Bioinformatics, Oxford University Press, 15 (1999), 234-242, http://bioinformatics.oxfordjournals.org/cgi/content/abstract/15/3/234.

[12]Waltz, D.L.: Understanding line drawings of scenes with shadows. Psychology of Computer Vision, McGraw-Hill, New York, 1975.

[13]F. Rossi, V. Dahr, C. Petrie: On the equivalence of constraint satisfaction problems. In Proceedings of the 9th European Conference on Artificial Intelligence, 1990, Stockholm, 550-556.

[14]L.-M. Rousseau, M. Gendreau, and G. Pesant: Solving small VRPTWs with Constraint Programming Based Column Generation. In Proceedings of CPAIOR'02, 2002, Annals of Operations Research, Volume 130, Springer, 199-216.

[15] M. E. J. Newman: Analysis of weighted networks, Phys. Rev. E 70, 056131 (2004), http://pre.aps.org/abstract/PRE/v70/i5/e056131

[16] J.-C. Régin: A filtering algorithm for constraints of difference in CSPs. In Proceedings of AAAI-94, 1994, 362–367.

[17] P. Shaw: Using constraint programming and local search methods to solve vehicle routing problems. Fourth International Conference on Principles and Practice of Constraint Programming. Volume of Lecture Notes in Computer Science 1520, 1998, 417–431.

[18] G. Desaulniers, E. Prescott-Gagnon, L.-M. Rousseau: A large neighborhood search algorithm for the vehicle routing problem with time windows. Networks, Volume 54, Wiley Periodicals, 2009, 190–204,

[19] J. A. Ferland, L. Fortin: Vehicles scheduling with sliding time windows. European Journal of Operational Research, Volume 38, Issue 2, 25 January 1989, 213-226.

[20] J. Karelahti: Solving the cutting stock problem in the steel industry. Master Thesis, Helsinki University Of Technology, http://www.sal.hut.fi/Publications/pdf-files/tkar02.pdf.

[21] A. Levin: Scheduling and Fleet Routing Models for Transportation Systems. Transportation Science, 5, 1971, 232–255.

[22] G. B. Dantzig, P. Wolfe: Decomposition Principle for Linear Programs. Operations Research, 8, 1960, 101–111.

[23] C. Barnhart, N. L. Boland, L. W. Clarke, E. L. Johnson, G. L. Nemhauser, and R. G. Shenoi: Flight String Models for Aircraft Fleeting and Routing. Transportation Science, 32, 1998, 208–220.

[24] C. Barnhart et al., Branch-and-Price: Column Generation for Solving Huge Integer Programs. Operations Research, Volume 46, 1998, 316-329.

[25] Company The Kite, s.r.o., Radlická 751/113e, Prague, http://aero.kite.cz.