Jan PÖSCHKO

# Optimization of a Purlin Punching Process

**MASTER THESIS**

written to obtain the academic degree of a Master of Science (MSc)

Master programme Mathematical Computer Science

Graz University of Technology

Advisor: Dipl.-Ing. Dr.techn. Univ.-Doz. Johannes HATZL

Institute of Optimization and Discrete Mathematics (Math B)

Graz, October 2012

## Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

................................... ...................................

(date) (signature)

## Abstract

In the production of purlins (structural members of roof constructions), screw holes of different diameters are punched into sheet metal. This is done by a punching machine “on the fly,” allowing the metal to move faster when there are longer distances between punches.

We present a mixed-integer linear program expressing the following combinatorial optimization problem: Find an equipment of the punching machine with punches of certain diameters and determine a set of machine stops and respective punched holes to minimize the number of machine stops and distribute them equally.

Although a simplified variant of the problem is a specialization of the $\mathbb{NP}$-hard hitting set problem, we show that it can be solved in polynomial time using dynamic programming. However, the simplifications and the worst-case time behavior of this approach render it inefficient for practical use.

Therefore, we propose efficient heuristics using randomly generated machine configurations with static punch positions and a greedy approach to solve the underlying hitting set problem. In a second step, the punch positions are relaxed to account for possible adjustments that can be made to the machine during the process. Moreover, the waste of metal resulting in the beginning of the process is minimized using similar considerations.

We evaluate our algorithm by comparing its results to known optimal solutions of both test cases from practice and randomly generated hole patterns.

The algorithm was designed and implemented for two commissioning companies and is in actual use in manufactories all over the world, saving both time and material in the production process.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acknowledgements

I would like to express my deep gratitude to Dr. Johannes Hatzl, my research advisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I am indebted to Dr. Eranda Dragoti-Çela, Dr. Elisabeth Gassner, and Dr. Bettina Klinz, for their contributions to this project.

I would also like to extend my thanks to Stephan Putz and the staff of Zeman Bauelemente Produktionsgesellschaft for enabling and supporting this work.

Finally, I wish to thank my family and friends for their support and encouragement throughout my studies.

Jan Pöschko
Graz, October 2012

# Symbols and Definitions

| Symbol | Explanation | Definition |
|---|---|---|
| $\emptyset$ | empty set or empty function | |
| $\mathbb{Z}$ | set of integers | $\{\dots, -1, 0, 1, \dots\}$ |
| $\mathbb{N}$ | set of nonnegative integers | $\{0, 1, 2, \dots\}$ |
| $\mathbb{R}$ | set of real numbers | |
| $\mathbb{R}_0^+$ | set of nonnegative real numbers | $\{z \in \mathbb{R} \mid z \geq 0\}$ |
| $[x, y]$ | set of real numbers $\geq x$ and $\leq y$ | $\{z \in \mathbb{R} \mid x \leq z \leq y\}$ |
| $\lvert x \rvert$ | absolute value of a real number or size of a set | |
| $\lfloor x \rfloor$ | the largest integer not greater than $x$ | $\max\{n \in \mathbb{Z} \mid n \leq x\}$ |
| $\deg_G v$ | degree of vertex $v$ in graph $G$ | |
| $\operatorname{Img} f$ | the image of a function $f : X \to Y$ | $\{f(x) \mid x \in X\}$ |
| $f^{-1}$ | the inverse function of $f : X \to Y$ | $f^{-1} : Y \to X$ |
| $A^t$ | transposed matrix of $A = (a_{ij})_{ij}$ | $(a_{ij})_{ji}$ |
| $n!$ | the factorial of $n$ | $\prod_{k=1}^{n} k$ |
| $(s_i)_{1 \leq i \leq I}$ | tuple of elements | $(s_1, \dots, s_I)$ |
| $\bigotimes_{1 \leq i \leq I} S_i$ | Cartesian product of a family of sets $S_i$, $1 \leq i \leq I$ | $\{(s_i)_{1 \leq i \leq I} \mid s_i \in S_i\}$ |
| $\mathcal{O}(f(n))$ | functions asymptotically not greater than $f(n)$ up to constant factors | (see Definition 4.2) |
| $\mathbb{P}$ | polynomial-time problems on a deterministic Turing machine | (see Definition 4.6) |
| $\mathbb{NP}$ | polynomial-time problems on a nondeterministic Turing machine | (see Definition 4.7) |
| $x.y$ | attribute named $y$ of object $x$ | (see Definition 5.7) |

CHAPTER 1

# Introduction

“Why socks? They only create holes!”
(Albert Einstein)

In this thesis, we deal with the optimization of a punching process of purlins. Purlins are structural members of roof constructions, typically linked to other parts of the roof by screws in pre-made holes (see Figure 1.1). Punching these holes is a central part of the manufacturing process of purlins, apart from folding and cutting the metal. Speeding up this process is the main objective of this work. At the same time, metal going to waste in the production process can be minimized.

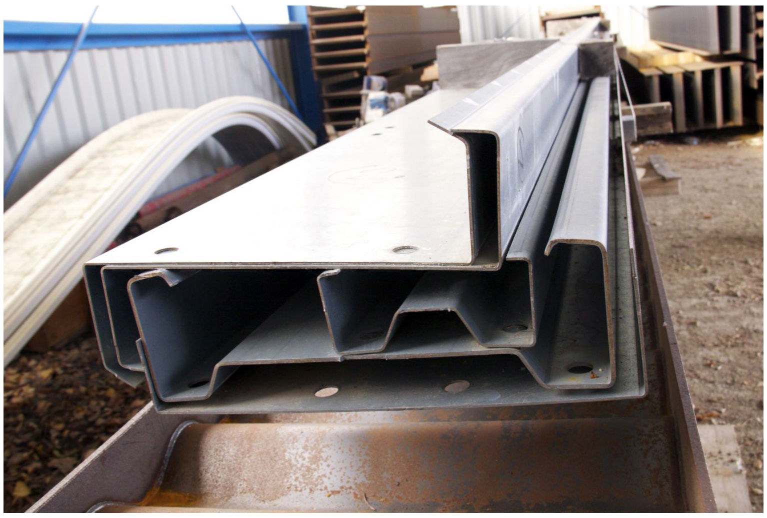


**Figure 1.1:** Photo of purlins.

This project was commissioned by Usitec Systemtechnik GmbH[1] and Zeman Bauelemente Produktionsgesellschaft[2] to the Institute of Optimization and Discrete Mathematics (Math B) at Graz University of Technology. The project was carried out with help from university staff. It involved modeling the practical problem, finding an algorithmic solution, and implementing it in C++.

**Related work** Much work has been done in the field of combinatorial optimization dealing with practical problems of industry. While we give a short outline of a few examples here, more information on the topic can be found in the book by Pardalos and Korotkikh [2003].

The *cutting stock problem* arises, for instance, when cutting rolls of paper of fixed width to rolls of various-sized widths with minimum waste. It can be formulated as a linear program and solved using delayed column-generation [Gilmore and Gomory, 1961, 1963]. In the one-dimensional case, branch-and-bound methods and dynamic programming can be used; modern algorithm reach optimality for practical problem sizes [Carvalho, 1998; Goulimis, 1990]. Branch-and-bound and dynamic programing are used in this thesis, too, to solve the presented mathematical model exactly and to examine the complexity of the punching problem.

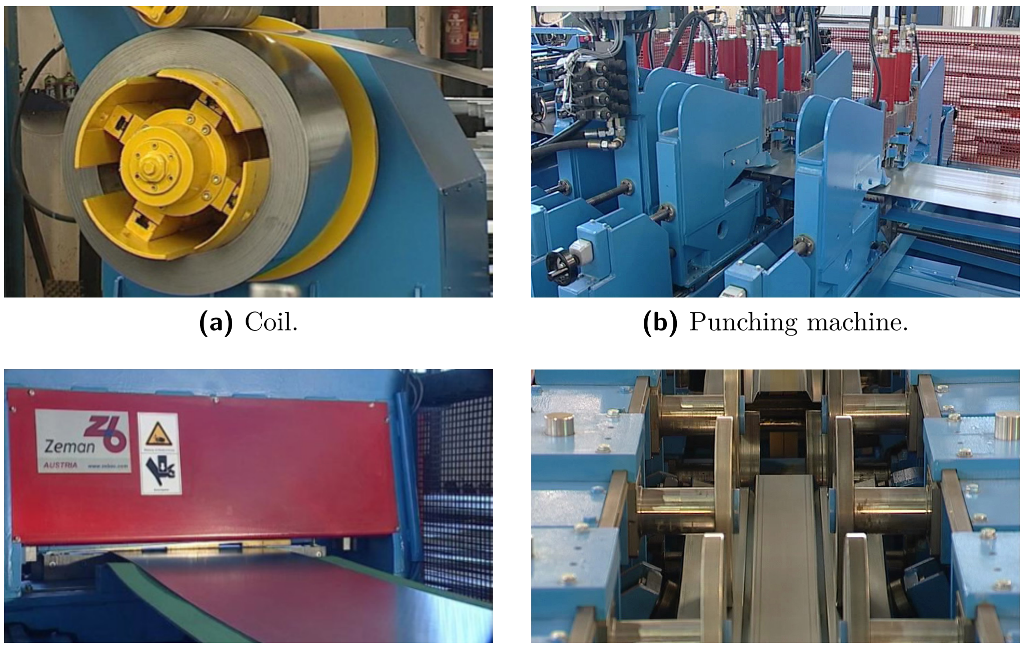
The *vehicle routing problem* asks for a cost-minimizing way of servicing a number of customers with a fleet of vehicles [Dantzig and Ramser, 1959]. It is related to the $\mathbb{NP}$-hard problems of job shop scheduling and the traveling salesman problem. An overview of exact and heuristic algorithms was compiled by Laporte [1992]. The punching problem is related to the set covering problem, another $\mathbb{NP}$-hard problem. Recent applications of vehicle routing include a large-scale approximation algorithm by Krumke et al. [2008] to optimize the routing of service units of the German Automobile Club (ADAC).

*Inventory problems* involve decisions on how much of a good to keep in stock, affecting the limits and profitability of future decisions. For instance, the question might be how much water to release from a dam for electricity. The problem might also involve *production* such as how much wheat to plant per year [Arrow, Karlin, and Scarf, 1958]. An overview of inventory theory is given by Hillier and Lieberman [2010].

As most combinatorial optimization problems are too hard to be solved exactly, they are often solved heuristically. Various *metaheuristics* such as simulated annealing, tabu search, and genetic algorithms are described by Glover and Kochenberger [2003].

In this thesis, we examine a novel optimization problem that cannot be reduced to existing problems directly. However, we draw ideas from random optimization [Li and Rhinehart, 1998] and local search [Glover and McMillan, 1986], for instance, as well as general principles from mathematical modeling, graph theory, and dynamic programming.

[1]Usitec Systemtechnik GmbH, Eisenstraße 35, A-4400 Steyr
[2]Zeman Bauelemente Produktionsgesellschaft mbH, St. Lorenzen 39, A-8811 Scheifling


**(a)** Coil. **(b)** Punching machine.

**(c)** Cutter. **(d)** Folder.

**Figure 1.2:** Photos of the production process of purlins.

**Organization of this thesis** The production process and the punching machine are described in greater detail in chapter 2. Basically, the production of purlins starts with metal running from a coil (see Figure 1.2a); holes are punched into the metal by a punching machine (see Figure 1.2b); finally, the metal is cut (see Figure 1.2c) and folded (see Figure 1.2d).

A mathematical model of the problem of estimating a time-optimal punching plan is shown in chapter 3. Finding an appropriate objective function was part of our work in this project and is presented in this chapter as well as methods to solve the resulting mixed-integer linear program.

In chapter 4, we examine the computational complexity of the punching problem. After discussing related work on the $\mathbb{NP}$-hard set cover problem, we prove that a simplified variant of the punching problem is efficiently solvable using a graph theoretic approach and dynamic programming.

Although having polynomial time and space requirements in the number of holes, the dynamic programming approach is inefficient and too limited in its assumptions for practical use, which is why we develop faster heuristics. Our approach is divided into two chapters. Chapter 5 deals with fixed machine configurations where punches have static positions, and chapter 6 describes heuristics to improve punching plans utilizing dynamic movements of punches.

After demonstrating the computational intractability of solving the mixed-integer linear program directly, we evaluate our algorithm by comparing it to optimal solutions to both hole patterns from practice and randomly generated test samples in chapter 7. Moreover, we discuss the impact of different heuristics and evaluate different parameter choices.

Finally, we briefly describe our conclusions, limitations of this thesis, and possible future work in chapter 8.

Some extra material is included in the appendix of this thesis. Formal definitions of some algorithms that would clutter the main part are given in appendix A. The implementation of the described mathematical model from chapter 3 in the AMPL modeling language is included in appendix B. Furthermore, an implementation of the dynamic programming solution from chapter 4 in Python is presented in appendix C. We do not include the implementation of our main algorithm in C++ because of its length. Instead, we visually depict a selection of patterns and solutions in appendix D.

The resulting program is in actual use in various factories equipped by Zeman Baulemente Produktionsgesellschaft, located in countries such as India, Iran, and Poland. Because it helps save time and material in the production process, the whole project was deemed a success by the commissioning companies.

CHAPTER 2

# Problem Description

In this chapter, an overview of the production process is given in section 2.1, including details of the punching machine in section 2.2 and an overview of the decisions that have to be taken when solving the punching problem in section 2.3.

## 2.1. Production Process

In the industrial production of steel purlins, metal running from a loop, a so-called *coil*, is drawn by a *feed roller* and processed in several consecutive steps (see Figures 1.2 and 2.1).

1. Holes are punched into the metal. These holes later serve as perforations for screws that link different purlins in roof constructions. A given pattern determines where holes are desired for each purlin.
2. Coming from a single coil with a length in the order of a hundred meters, the metal is cut into pieces with a length of a few meters. Each piece yields a single purlin.
3. The metal is rolled or cold-formed into so-called C-, U-, or Z-shapes (see Figure 1.1). Even though the metal is usually only a few millimeters thin, the shape of the purlins results in very high stability.

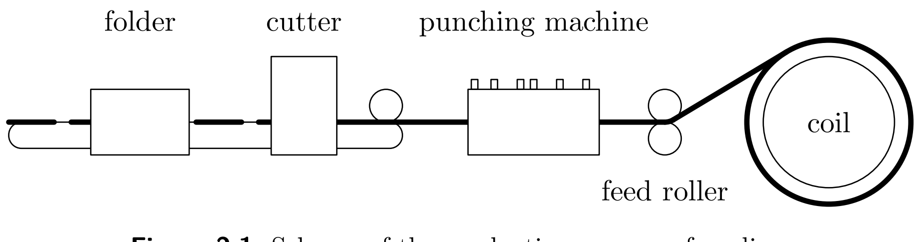


**Figure 2.1:** Schema of the production process of purlins.

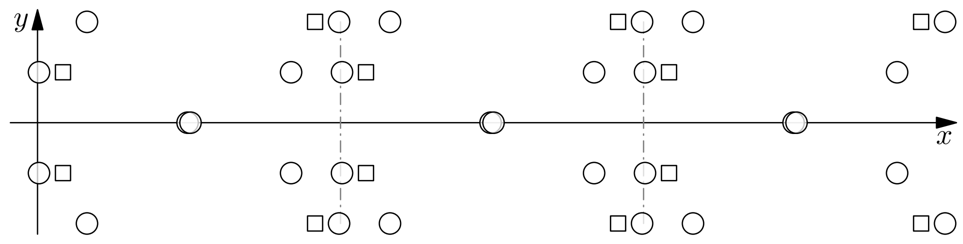


**Figure 2.2:** Example of a hole pattern consisting of three repetitions of the same block of holes. Higher $x$-coordinates represent positions running later from the coil. Different hole types are represented by different shapes (circles and squares, respectively), although in practice they are usually all round and differ only in diameter. The $x$-axis and the $y$-axis are scaled differently to provide a clearer picture.

Usually, the whole coil is processed in a single turn called a *batch.* A typical batch consists of several different types of purlins regarding their hole patterns, with several repetitions each.

The punching machine performs so-called *flying punching*, that is, it moves along with the metal within its mounting for a short period of time while performing the punch, before moving back to its initial position. Even though the object being moved in the process is the metal, it is considered to be the fixed part with the punching machine moving over it. This yields a more concise model, as the positions of the holes are the same in the given pattern and during the process.

Let the direction in which the metal moves be called the *$x$-direction*, where higher coordinates correspond to positions running later from the coil. Consequently, figures of hole patterns will have holes being punched earlier further to the left. Furthermore, let the *$y$-direction* be oriented orthogonal to the $x$-direction in the standard way, that is, positive $y$-positions are to the left when viewed in positive $x$-direction.

**Example 2.1.** An example pattern illustrating the coordinate system is shown in Figure 2.2. Depicted are three repetitions of the same block of holes. However, in practice the number of repetitions is much higher, usually in the range of 60 to 100. Two different hole types occur in the pattern.

As suggested in the example, holes typically appear symmetrically with respect to the $x$-axis in practical patterns. There are some exceptions, though, as depicted in Figures D.2 and D.3.

The holes in the pattern can have different shapes and diameters, determining their *type.* As the number of distinct types is very low and the exact shape and diameter does not influence the punching process, these two characteristics can be abstracted to a finite set of types.

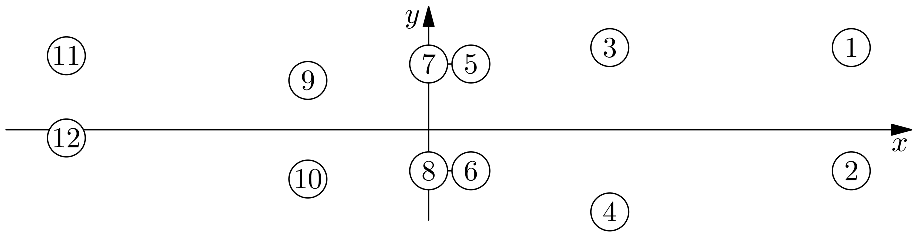


**Figure 2.3:** The punching machine primarily considered in this work. The punches are numbered from 1 to 12 sorted with descending $x$- and $y$-coordinates. Punches are moveable in both $x$- and $y$-direction with restrictions on their positions and distances. Punches 5 to 8 form the double stamp; the pairs of punches 5/7 and 6/8 are linked together statically each, that is, their respective $y$ positions are the same in each punching step and their distances in $x$-direction remain constant throughout the process.

## 2.2. Punching Machine

The punching machine primarily dealt with in this work consists of six pairs of *punches*. To simplify notation, let a pair of two punches be called a *stamp*. To conform with existing practice, the punches are numbered 1 to 12 in reverse $x$-direction, that is, the position of punch $i$ is not smaller than that of punch $j$ for $i < j$.

Each punch is equipped with a certain *tool*, corresponding to a certain type of hole. This equipment has to be determined before the process starts and stays constant throughout the whole batch. There might be a restriction on which tools can be assigned to each punch.

The two central stamps (punches 5 to 8) are linked together statically, forming the so-called *double stamp*; the other stamps—two to the left, two to the right—can be moved in $x$-direction relative to the double stamp between consecutive punching steps, with a certain upper limit on the distance being moved. The *center* of the machine is defined as the $x$-position of punches 7 and 8, with *relative positions* being defined as the distances of stamps to this center. Consequently, the relative position of the double stamp stays constant. An illustration of the punching machine is shown in Figure 2.3.

There are limitations on the maximum distances of punches 1 and 12 to the center of the machine, and there has to be a certain minimum distance between all pairs of punches. This can be generalized to minimum and maximum distances between each pair of punches.

The $y$-positions of the punches are dynamic as well. The only restrictions are that there has to be a certain minimum distance between the two punches of a stamp, and that the the $y$-positions of two corresponding punches of the double stamp must be equal.

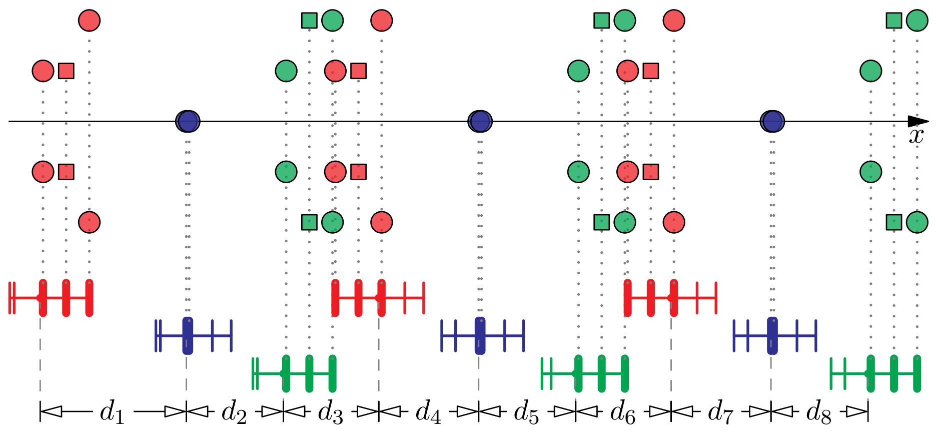


**Figure 2.4:** Solution to a given pattern of holes. Different colors denote different punching steps. The pattern of holes is shown at the top. Depicted below are the positions of the punches in each step, where active punches are drawn bold and are connected to their respective punched holes by dotted lines. At the bottom the machine positions are shown again with the distances $d_i$ between consecutive steps.

## 2.3. Decisions Constituting a Solution

Given a hole pattern and the characteristics of the punching machine, the optimization goal is to find

1. an equipment of the machine with tools,
2. a mapping from the given holes to punches that punch them, and
3. a sequence of positions of the individual punches

with the principal objective of minimizing the number of machine stops and distributing them equally. The reasoning behind this is that small movements of the machine take longer, as it cannot accelerate that fast. Especially, consecutive punching steps at the same position have to be avoided whenever possible, as the machine has to stop completely for a while in that case.

An equipment of the machine is given by an assignment of tools (corresponding to different hole types) to punches. Each hole has to be punched exactly once throughout the whole batch. The positions of the punches relative to the machine center have to be determined because they are dynamic and can be changed after each punching step.

A formal definition of the corresponding mathematical model is given in chapter 3.

**Example 2.2** (Example 2.1 continued)**.** Given the pattern depicted in Figure 2.2, a possible solution would be to equip punches 1, 2, 9, and 10 with the square hole type and punches 3 to 8, 11, and 12 with the circle hole type. The mapping from holes to punches and the positions of the punches in each of the nine punching steps are shown in Figure 2.4.

CHAPTER 3

# Mathematical Model

In this chapter, the punching problem is introduced formally. The parameters and decision variables are described in section 3.1. The constraints on these variables are shown in section 3.2 and eventually formulated as a mixed-integer linear program. Optimization objectives are discussed in section 3.3 resulting in the formal definition of the central *step* and *speed* punching problems as mixed-integer linear programs.

The parameter choices arising in practice as given by the commissioning companies are presented in section 3.4. We also give a short introduction to general methods for solving mixed-integer linear programs in section 3.5.

Adding to that, an implementation of the mathematical program in the AMPL modeling language [Fourer, Gay, and Kernighan, 2002] can be found in appendix B.1.

## 3.1. Parameters and Variables

The parameters in Table 3.1 describe the pattern to be punched.

**Table 3.1:** Parameters describing the hole pattern.

| **Parameter** | **Explanation** |
|---|---|
| $x_n \in \mathbb{R}$ | $x$-position of hole $n$ |
| $y_n \in \mathbb{R}$ | $y$-position of hole $n$ |
| $z_{nt} \in \{0,1\}$ | 1 if hole $n$ requires tool $t$, 0 otherwise |

**Example 3.1** (Examples 2.1 and 2.2 continued)**.** The first block of length 72000 in the pattern shown in Figure 2.2 can be described using the following parameters,

where tool 1 corresponds to the circle and tool 2 to the square hole types. All $x$- and $y$-coordinates are specified in tenth of millimeters.

| $n$ | $x_n$ | $y_n$ | $z_{n1}$ | $z_{n2}$ | $n$ | $x_n$ | $y_n$ | $z_{n1}$ | $z_{n2}$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 350 | −500 | 1 | 0 | 8 | 36350 | 0 | 1 | 0 |
| 2 | 350 | 500 | 1 | 0 | 9 | 60250 | −500 | 1 | 0 |
| 3 | 6050 | −500 | 0 | 1 | 10 | 60250 | 500 | 1 | 0 |
| 4 | 6050 | 500 | 0 | 1 | 11 | 65950 | −1000 | 0 | 1 |
| 5 | 11750 | −1000 | 1 | 0 | 12 | 65950 | 1000 | 0 | 1 |
| 6 | 11750 | 1000 | 1 | 0 | 13 | 71650 | −1000 | 1 | 0 |
| 7 | 35650 | 0 | 1 | 0 | 14 | 71650 | 1000 | 1 | 0 |

The parameters in Table 3.2 describe constraints to the punching machine as outlined in section 2.2. The corresponding constraints are described formally in section 3.2 by eqs. (3.13) and (3.15) to (3.19), and typical values of these parameters are presented in section 3.4.

**Table 3.2:** Parameters describing constraints to the punching machine.

| **Parameter** | **Explanation** |
|---|---|
| $T_{jt} \in \{0,1\}$ | 1 if tool $t$ can be assigned to punch $j$, 0 otherwise |
| $\overline{j}$ | the reference punch relative to which punch movements are measured |
| $X$ | maximum $x$-movement of a punch relative to the reference punch $\overline{j}$ between consecutive punching steps |
| $\underline{X}_{jk}$ | minimum $x$-distance between punch $j$ and punch $k$ |
| $\overline{X}_{jk}$ | maximum $x$-distance between punch $j$ and punch $k$ |
| $\underline{Y}_{jk}$ | minimum $y$-distance between punch $j$ and punch $k$ |
| $\overline{Y}_{jk}$ | maximum $y$-distance between punch $j$ and punch $k$ |
| $\underline{V}_j$ | minimum $y$-position of punch $j$ |
| $\overline{V}_j$ | maximum $y$-position of punch $j$ |

We introduce decision variables defined in Table 3.3.

**Table 3.3:** Decision variables.

| Variable | Explanation |
|---|---|
| $s_{ij} \in \mathbb{R}$ | the absolute $x$-position of punch $j$ in punching step $i$ |
| $\overline{s}_{ij} \in \mathbb{R}$ | the relative $x$-position of punch $j$ in step $i$ (relative to the machine center) |
| $v_{ij} \in \mathbb{R}$ | the $y$-position of punch $j$ in step $i$ |
| $u_{jt} \in \{0,1\}$ | 1 if tool $t$ is assigned to punch $j$, 0 otherwise |
| $h_{ijn} \in \{0,1\}$ | 1 if hole $n$ is punched by punch $j$ in step $i$, 0 otherwise |
| $a_{ij} \in \{0,1\}$ | 1 if punch $j$ punches any hole in step $i$, 0 otherwise |
| $\overline{a}_i \in \{0,1\}$ | 1 if any punch punches any hole in step $i$, 0 otherwise |

The variables $s_{ij}$ and $\overline{s}_{ij}$ account for dynamic $x$-movements of the punches and are related through

$$\overline{s}_{ij} = s_{ij} - s_{i\overline{j}}, \tag{3.1}$$

that is, the relative position of a punch $j$ is the difference of its absolute position to the position of the reference punch $\overline{j}$.

The variables $u_{jt}$ determine the equipment of the machine with different tools. This equipment remains constant throughout the whole process, so it does not depend on the step $i$.

The variables $h_{ijn}$ describe the punching plan by stating which hole gets punched by which punch in what step. The binary variables $a_{ij}$ and $\overline{a}_i$ depend on $h_{ijn}$ through the following relations. As each punch $j$ cannot punch more than one hole in a single turn $i$, the relation

$$a_{ij} = \sum_n h_{ijn} \qquad \forall i, j$$

holds. As $a_i$ has to be 1 if a single $a_{ij}$ has to be 1 and has to be 0 if all $a_{ij}$ are 0, the two constraints

$$\begin{aligned} \overline{a}_i &\geq a_{ij} && \forall i, j \\ \overline{a}_i &\leq \sum_j a_{ij} && \forall i, j \end{aligned}$$

have to be fulfilled. The variables $\overline{a}_i$ can be used to quantify the actual number of punching steps.

**Example 3.2** (Examples 2.1, 2.2 and 3.1 continued)**.** The first six punching steps in the solution shown in Figure 2.4 can be described using the following values of variables.

| $i \rightarrow$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $s_{i1} = s_{i2}$ | 11750 | 46700 | 71650 | 94100 | 118700 | 143650 |
| $s_{i3} = s_{i4}$ | 6050 | 42050 | 65950 | 89450 | 114050 | 137950 |
| $s_{i5} = s_{i6}$ | 350 | 36350 | 60250 | 83750 | 108350 | 132250 |
| $s_{i7} = s_{i8}$ | $-350$ | 35650 | 59550 | 83050 | 107650 | 131550 |
| $s_{i9} = s_{i,10}$ | $-6750$ | 29250 | 53150 | 78050 | 101950 | 125500 |
| $s_{i,11} = s_{i,12}$ | $-7850$ | 28150 | 52050 | 72350 | 98550 | 123250 |
| $v_{i,1}$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| $v_{i,2}$ | $-1000$ | $-1000$ | $-1000$ | $-1000$ | $-1000$ | $-1000$ |
| $v_{i,3}$ | 500 | 1000 | 1000 | 1000 | 1000 | 1000 |
| $v_{i,4}$ | $-500$ | $-1000$ | $-1000$ | $-1000$ | $-1000$ | $-1000$ |
| $v_{i,5}$ | 500 | 0 | 500 | 1000 | 0 | 500 |
| $v_{i,6}$ | $-500$ | $-1000$ | $-500$ | $-1000$ | $-1000$ | $-500$ |
| $v_{i,7}$ | 500 | 0 | 500 | 1000 | 0 | 500 |
| $v_{i,8}$ | $-500$ | $-1000$ | $-500$ | $-1000$ | $-1000$ | $-500$ |
| $v_{i,9}$ | 1000 | 1000 | 1000 | 500 | 1000 | 1000 |
| $v_{i,10}$ | $-1000$ | $-1000$ | $-1000$ | $-500$ | $-1000$ | $-1000$ |
| $v_{i,11}$ | 1000 | 1000 | 1000 | 500 | 1000 | 1000 |
| $v_{i,12}$ | $-1000$ | $-1000$ | $-1000$ | $-500$ | $-1000$ | $-1000$ |
| $h_{ijn}$ | $h_{1,1,6}$ | $h_{2,5,8}$ | $h_{3,1,14}$ | $h_{4,5,20}$ | $h_{5,5,22}$ | $h_{6,1,28}$ |
| | $= h_{1,2,5}$ | $= h_{2,7,7}$ | $= h_{3,2,13}$ | $= h_{4,6,19}$ | $= h_{5,7,21}$ | $= h_{6,2,27}$ |
| | $= h_{1,3,4}$ | $= 1$ | $= h_{3,3,12}$ | $= h_{4,9,18}$ | $= 1$ | $= h_{6,3,26}$ |
| | $= h_{1,4,3}$ | | $= h_{3,4,11}$ | $= h_{4,10,17}$ | | $= h_{6,4,25}$ |
| | $= h_{1,5,2}$ | | $= h_{3,5,10}$ | $= h_{4,11,16}$ | | $= h_{6,5,24}$ |
| | $= h_{1,6,1}$ | | $= h_{3,6,9}$ | $= h_{4,12,15}$ | | $= h_{6,6,23}$ |
| | $= 1$ | | $= 1$ | $= 1$ | | $= 1$ |

Each column represents a single punching step. As the considered punching machine consists of pairs of punches with identical $x$-positions, $s_{i,2j-1} = s_{i,2j}$ holds. The values of $h_{ijn}$ are very sparse, so only the variables having value 1 are shown here, with all others having value 0.

Note that the relative punch positions $\bar{s}_{ij}$ can be calculated using the absolute positions $s_{ij}$ through equation (3.1).

The equipment of the machine is given by the values of $u_{jt}$ as follows. The punches having the circle hole type (corresponding to tool 1 as in Example 3.1) induce

$$u_{11} = u_{21} = u_{51} = u_{61} = u_{71} = u_{81} = u_{11,1} = u_{12,1} = 1,$$

while the punches having the square hole type (tool 2) induce

$$u_{32} = u_{42} = u_{92} = u_{10,2} = 1,$$

with all other values of $u_{jt}$ being 0.

To elaborate further on this example, consider punching step $i = 2$, for instance. The two holes 8 and 7 are punched by the punches 5 and 7, respectively ($h_{258} = h_{277} = 1$). Thus, their $x$- and $y$-positions have to be equal, that is,

$$\begin{aligned} s_{25} &= x_8 = 36350 & \qquad v_{25} &= y_8 = 0 \\ s_{27} &= x_7 = 35650 & \qquad v_{27} &= y_7 = 0. \end{aligned}$$

Furthermore, the tools of the punches must match the hole type (which is 1 in both cases), that is,

$$\begin{aligned} u_{51} &= z_{81} = 1 & \qquad u_{52} &= z_{82} = 0 \\ u_{71} &= z_{71} = 1 & \qquad u_{72} &= z_{72} = 0. \end{aligned}$$

## 3.2. Constraints

Using the variables described in the previous section, the punching problem can be formalized by constraints discussed in this section.

In addition to potential other constraints, all indices are bounded by $1 \le i \le I$, $1 \le j \le J$, $1 \le n \le N$, and $1 \le t \le T$. Although the exact optimal number $I^*$ of punching steps is not known a priori, it can be assumed that in each step at least one hole is punched and therefore $I^* \le N$ holds. If it turns out that less punching steps are needed, $h_{ijn}$ can simply be set to 0 for $i > I^*$. Let such a step $i$ be called an *empty* punching step.

Determining a good upper bound for $I^*$ a priori could speed up the process of finding an optimal solution, as the number of binary decisions regarding $h_{ijn}$ is reduced. We examine this in section 7.1. Without any other bound, $I^* = N$ can be chosen in any case.

### Quadratic formulation

A direct formulation of the requirement that active punches are on correct $x$-positions would be

$$\sum_n x_n h_{ijn} = s_{ij} \qquad \forall i\ \forall j \in A_i \tag{3.2}$$

where

$$A_i = \{j \mid \exists n : h_{ijn} = 1\}$$

is the set of active punches in step $i$. The variable $h_{ijn}$ is 1 only in a step $i$ where punch $j$ punches hole $n$, so the sum on the left side becomes $x_n$ then. The $x$-position of punch $j$ in this step is given by $s_{ij}$ and has to equal the $x$-position of the punched hole. For inactive punches there is no corresponding restriction on their $x$-position.

Analogously, the $y$-position of active punches must equal the $y$-position of their punched hole, that is,

$$\sum_n y_n h_{ijn} = v_{ij} \qquad \forall i \; \forall j \in A_i. \tag{3.3}$$

Furthermore, active punches must have tools corresponding to their punched holes, that is,

$$\sum_n z_{nt} h_{ijn} = u_{jt} \qquad \forall i \; \forall j \in A_i \; \forall t. \tag{3.4}$$

However, this model still involves sets $A_i$ of active punches that are used to restrict other indices. The sets $A_i$ depend on the result (namely $h_{ijn}$), so are not suitable for a classic mixed-integer programming formulation of the problem. That can be avoided by introducing additional binary variables $a_{ij} \in \{0,1\}$ expressing the decision whether a punch $j$ is active in step $i$. Using these variables, eqs. (3.2) to (3.4) can be rewritten as

$$\begin{aligned} \sum_n x_n h_{ijn} a_{ij} &= s_{ij} a_{ij} && \forall i \; \forall j \\ \sum_n y_n h_{ijn} a_{ij} &= v_{ij} a_{ij} && \forall i \; \forall j \\ \sum_n z_{nt} h_{ijn} a_{ij} &= u_{jt} a_{ij} && \forall i \; \forall j \; \forall t \end{aligned}$$

with the additional constraint

$$\sum_i \sum_j h_{ijn} a_{ij} = 1 \qquad \forall n$$

which expresses the requirement that each hole is punched by an active punch exactly once. However, this formulation involves products of variables and is therefore quadratic but not linear.

## Linear formulation

To get a linear model, another approach has to be taken to formulate the requirement that $h_{ijn}$ can only be 1 if the active punch $j$ in step $i$ has the correct position and tool to punch hole $n$.

We use the auxiliary variable $\overline{h}_{ijn} \geq 0$ with the underlying idea that it shall be 0 only if punch $j$ can punch hole $n$ in step $i$. This is somehow the negation of $h_{ijn}$, although $\overline{h}_{ijn}$ does not have to be binary, not even integer.

We can enforce the idea

$$\overline{h}_{ijn} \begin{cases} = 0 & \text{only if punch } j \text{ can punch hole } n \text{ in step } i \\ > 0 & \text{otherwise (or nevertheless)} \end{cases}$$

regarding the $x$-position by demanding

$$\overline{h}_{ijn} \geq |x_n - s_{ij}| \qquad \forall i \; \forall j \; \forall n. \tag{3.5}$$

This ensures that $\overline{h}_{ijn} = 0$ only if $x_n = s_{ij}$. Similarly, we get

$$\overline{h}_{ijn} \geq |y_n - v_{ij}| \qquad \forall i \; \forall j \; \forall n \tag{3.6}$$

to ensure that $\overline{h}_{ijn} = 0$ only if the $y$-position of an active punch $j$ in step $i$ is correct, that is, $y_n = v_{ij}$. Furthermore,

$$\overline{h}_{ijn} \geq |z_{nt} - u_{jt}| \qquad \forall i \; \forall j \; \forall n \; \forall t \tag{3.7}$$

ensures that active punches are equipped with the correct tools.

Using the auxiliary variable $\overline{h}_{ijn} \geq 0$ we want to ensure that $h_{ijn} = 1$ only if $\overline{h}_{ijn} = 0$. This can be done in the standard way of expressing "if-then-relations" in mixed-integer linear programs [Smith and Taşkin, 2008]. Using an upper bound $\overline{M}$ on $\overline{h}_{ijn}$, we introduce another auxiliary (binary) variable

$$h'_{ijn} = \begin{cases} 0 & \text{only if } \overline{h}_{ijn} = 0 \\ 1 & \text{otherwise (or nevertheless)} \end{cases}$$

with the constraint

$$\overline{h}_{ijn} \leq \overline{M} h'_{ijn} \qquad \forall i \; \forall j \; \forall n. \tag{3.8}$$

Now if $\overline{h}_{ijn} > 0$, the variable $h'_{ijn}$ has to be 1. The upper bound $\overline{M}$ which applies anyway is no actual restriction on $\overline{h}_{ijn}$. If $\overline{h}_{ijn} = 0$, the variable $h'_{ijn}$ can be chosen to be either 0 or 1. By the additional constraint

$$h_{ijn} \leq 1 - h'_{ijn} \qquad \forall i \; \forall j \; \forall n \tag{3.9}$$

we ensure that $h_{ijn}$ is 0 whenever $\overline{h}_{ijn} > 0$ and thereby $h'_{ijn} = 1$. Conversely, $h_{ijn}$ can only be 1 if $\overline{h}_{ijn} = 0$ which means that punch $h$ can punch hole $n$ in step $i$.

A safe choice for the upper bound $\overline{M} \geq \overline{h}_{ijn}$ results from the maximum dimensions of the hole pattern, that is,

$$\overline{M} = 2 \max_n \max\{|x_n|, |y_n|, 1\}, \tag{3.10}$$

because it must be possible to choose $\overline{h}_{ijn}$ greater than any differences between any two reasonable $x$- and $y$-positions (see eqs. (3.5) and (3.6)). In the rather unrealistic case where $x_n = y_n = 0$ for all $n$ it shall still be possible to choose $\overline{h}_{ijn} = 0$ to account for eq. (3.7).

To ensure that every hole is punched exactly once, we demand

$$\sum_i \sum_j h_{ijn} = 1 \qquad \forall n. \tag{3.11}$$

Note that this can be relaxed to $\sum_i \sum_j h_{ijn} \geq 1$ equivalently, requiring that each hole is punched *at least* once, because multiple punches of the same hole can simply be ignored. Although a hole should not actually be punched several times as this can make it “floppy,” repeated punches in the theoretical solution do not have to be carried out and are irrelevant to the minimization problem. This relaxation might be useful when solving the model in practice.

Finally, we relate $h_{ijn}$ to $a_{ij}$ using the restriction

$$a_{ij} = \sum_n h_{ijn} \qquad \forall i \, \forall j. \tag{3.12}$$

## Machine constraints

In addition to the previous constraints regarding the positions of punches related to the holes they punch, the following general machine constraints have to be fulfilled.

Punches have to be equipped with feasible tools, that is,

$$u_{jt} \leq T_{jt} \qquad \forall j \, \forall t, \tag{3.13}$$

and each punch has to be assigned exactly one tool,

$$\sum_t u_{jt} = 1 \qquad \forall j. \tag{3.14}$$

Regarding dynamic movements of the punches, the $x$-distances between punches must be within their limits, that is,

$$\underline{X}_{jk} \leq s_{ij} - s_{ik} \leq \overline{X}_{jk} \qquad \forall i \, \forall j \, \forall k, \tag{3.15}$$

as well as the $y$-distances between punches,

$$\underline{Y}_{jk} \leq v_{ij} - v_{ik} \leq \overline{Y}_{jk} \qquad \forall i \, \forall j \, \forall k, \tag{3.16}$$

and the individual $y$-positions,

$$\underline{V}_j \leq v_{ij} \leq \overline{V}_j \qquad \forall i\, \forall j. \tag{3.17}$$

The $x$-movements of the punches are restricted by

$$|\overline{s}_{ij} - \overline{s}_{i-1,j}| \leq X \qquad \forall i > 1\, \forall j. \tag{3.18}$$

The relative positions of punches are coupled to the reference punch through

$$\overline{s}_{ij} = s_{ij} - s_{i\overline{j}} \qquad \forall i\, \forall j. \tag{3.19}$$

Note that restrictions of the form $A \geq |B|$ involving absolute values can easily be reformulated in a strictly linear way using the two constraints

$$\begin{aligned} A &\geq B \\ A &\geq -B. \end{aligned}$$

To ensure that there is no empty punching step followed by a non-empty step, we add the constraints

$$\overline{a}_i \leq \overline{a}_{i-1} \qquad \forall i > 1 \tag{3.20}$$

$$\overline{a}_i \geq a_{ij} \qquad \forall i\, \forall j \tag{3.21}$$

$$\overline{a}_i \leq \sum_j a_{ij} \qquad \forall i. \tag{3.22}$$

All relevant constraints can be summed up in the following definition.

**Definition 3.3** (Punching constraints)**.** Let $N$, $I \leq N$, $J$, $T$, and $1 \leq \overline{j} \leq J$ be integers. By *punching constraints* we denote the constraints given by eqs. (3.5) to (3.22).

## 3.3. Optimization Objectives

There are two optimization objectives: the speed of the process and the resulting waste. They can be used to formulate the mathematical models discussed throughout this thesis, the *step* and the *speed* punching problem.

### Speed

The speed of the process varies according to the distances the metal moves between consecutive punching steps (*movements*). During longer movements, it can accelerate more without losing precision. Contrarily, small movements slow down the process significantly. As the total movement of the metal is constant—it is given the length of the whole batch—maximizing individual movements implies minimizing the number of stops.

Although the effective speed of the machine is not given formally in practice, it is assumed to be faster when movements are distributed more equally. Designing an appropriate objective function was part of our work during this project.

Let the time, or more abstractly, the *costs*, needed to perform a movement of distance $d$ be denoted by $c(d)$ with a function $c : \mathbb{R}_0^+ \to \mathbb{R} \cup \{\infty\}$. Then favoring equally distributed movements specifically implies

$$c\left(d_1\right) + c\left(d_2\right) \geq 2c\left(\tfrac{d_1+d_2}{2}\right),$$

that is, splitting a distance $d_1 + d_2$ into two equal movements does not yield higher costs than any other splitting. This means that $c$ is a midpoint-convex function. Additionally demanding $c$ to be Lebesgue-measurable implies that $c$ is convex [Donoghue, 1969, p. 12].

There are several reasonable choices for convex cost functions:

1. Assuming that the speed $v$ of the machine is inversely proportional to the square of the moved distance $d$,

   $$v(d) = \frac{1}{d^2},$$

   results in the time

   $$c(d) = \frac{d}{v(d)} = \frac{1}{d}, \tag{3.23}$$

   a convex function for $d > 0$ with $c(0) = \infty$. This accounts well for the fact that *zero-movements* (the machine performing two consecutive punching steps in the same position) are to be avoided if somehow possible.

   In an implementation, dealing with values of $\infty$ can be circumvented by choosing an appropriate upper bound for the total costs as the value of $c(0)$, which is particularly possible when all occurring coordinates and distances are integer.

2. To get a linear objective function, $\frac{1}{d}$ can be approximated by a piecewise linear function, thereby also avoiding values of $\infty$. Given $\delta_r$, $0 \leq r \leq R$, a possible choice would be a function of the form

   $$c(d) = \frac{1}{\delta_0} + \sum_{r=1}^{R} \sigma_r d_r \tag{3.24}$$

   with slopes

   $$\sigma_r = \begin{cases} \frac{\frac{1}{\delta_r} - \frac{1}{\delta_{r-1}}}{\delta_r - \delta_{r-1}} & 1 < r \leq R \\ \frac{\frac{1}{\delta_1} - \frac{1}{\delta_0}}{\delta_1} & r = 1 \end{cases}$$

   and the “fragmentation”

   $$d = \sum_{r=1}^{R+1} d_r,$$

   where $0 \leq d_1 \leq \delta_1$ and $0 \leq d_r \leq \delta_r - \delta_{r-1}$ for $1 < r \leq R$ and $d_r \geq d_{r+1}$ for $0 \leq r \leq R$. An illustration of this linearization is shown in Figure 3.1.

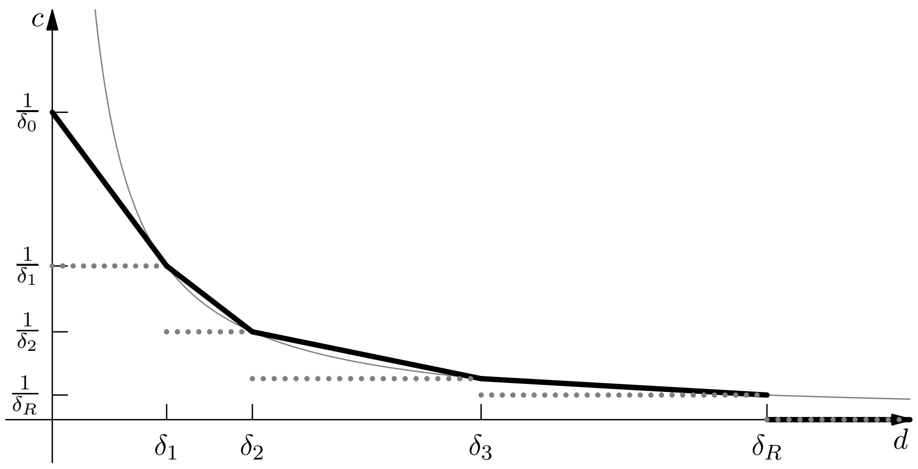


**Figure 3.1:** Linearizations of the continuous cost function $c(d) = \frac{1}{d}$. The piecewise linear function from eq. (3.24) is shown in bold black, while the piecewise constant function from eq. (3.25) is shown in dotted gray.

3. Similarly, a piecewise constant function of the form

$$c(d) = \begin{cases} \frac{1}{\delta_1} & 0 \leq d < \delta_1 \\ \dots & \\ \frac{1}{\delta_R} & \delta_{R-1} \leq d < \delta_R \\ 0 & d \geq \delta_R \end{cases} \tag{3.25}$$

   can be used to approximate $\frac{1}{d}$ (see Figure 3.1).

4. If all that matters is minimizing the number of stops of the machine, the simple specialization

$$c(d) = 1$$

   can be chosen.

Eventually, the costs of all movements $d_i$ between step $i-1$ and step $i$, $2 \leq i \leq I$, can be summed up to yield overall costs

$$c = \sum_{i \geq 2} c(d_i).$$

The presented objective functions can be used to formulate the following mathematical models that are discussed throughout the rest of this thesis.

**Definition 3.4** (Step punching problem)**.** By the *step punching problem* we denote an optimization problem of the form

$$\min \sum_i \overline{a}_i$$

subject to punching constraints.

*Remark.* This is equivalent to

$$\min \left(1 + \sum_{i \geq 2} c(d_i)\right) \qquad \text{with } c(d_i) = 1,$$

assuming that there is at least one punching step.

*Remark.* Minimizing the number of punching steps is a very natural objective in practice, as each stop of the machine needs a significant amount of time.

**Definition 3.5** (Speed punching problem)**.** Let $\delta_r > 0$ for $0 \leq r \leq R$. In the *speed punching problem*, we use the objective function

$$\min \sum_{i \geq 2} c(d_i),$$

where $c$ is the piecewise linear function from (3.24). This can be formulated as

$$\min \sum_{i \geq 2} \left(\frac{1}{\delta_0} + \sum_{r \geq 1} \sigma_r d_{ir}\right)$$

subject to punching constraints and the additional constraints

$$\begin{aligned}
\sigma_r &= \frac{\frac{1}{\delta_r} - \frac{1}{\delta_{r-1}}}{\delta_r - \delta_{r-1}} && \forall r > 1 \\
\sigma_1 &= \frac{\frac{1}{\delta_1} - \frac{1}{\delta_0}}{\delta_1} && \\
0 \leq d_{ir} &\leq \delta_r - \delta_{r-1} && \forall i > 1 \; \forall r > 1 \\
0 \leq d_{i1} &\leq \delta_1 && \forall i > 1 \\
d_i &\geq \sum_{r \geq 1} d_{ir} && \forall i > 1 \\
d_i &= s_{i\bar{j}} - s_{i-1,\bar{j}} && \forall i > 1.
\end{aligned}$$

*Remark.* As the objective function in Definition 3.5 is minimized and the slopes $\sigma_r$ are decreasing and negative, $d_{ir} \leq d_{i,r+1}$ is induced.

*Remark.* If the number $I^*$ of non-empty punching steps is smaller than the provided upper bound $I$ (which is usually set to $N$), there are $I - I^*$ empty punching steps $i$ in which $h_{ijn} = 0$ is set. For these steps, there is no effective restriction on the $x$- and $y$-positions of the punches: Looking at eq. (3.9), $h'_{ijn}$ can be set to 1 as $h_{ijn} = 0$ is fulfilled anyway, and therefore $\overline{h}_{ijn}$ can be set to its upper bound $\overline{M}$ in eq. (3.8). Then there is no restriction on $s_{ij}$ and $v_{ij}$ imposed by eqs. (3.5) and (3.6). Consequently, it is feasible to set $s_{ij}$ such that the machine movements $d_i$ satisfy

$$d_i > \delta_R \qquad \text{for } i > I^*.$$

Therefore, it can be assured that empty punching steps do not contribute costs to the objective function.

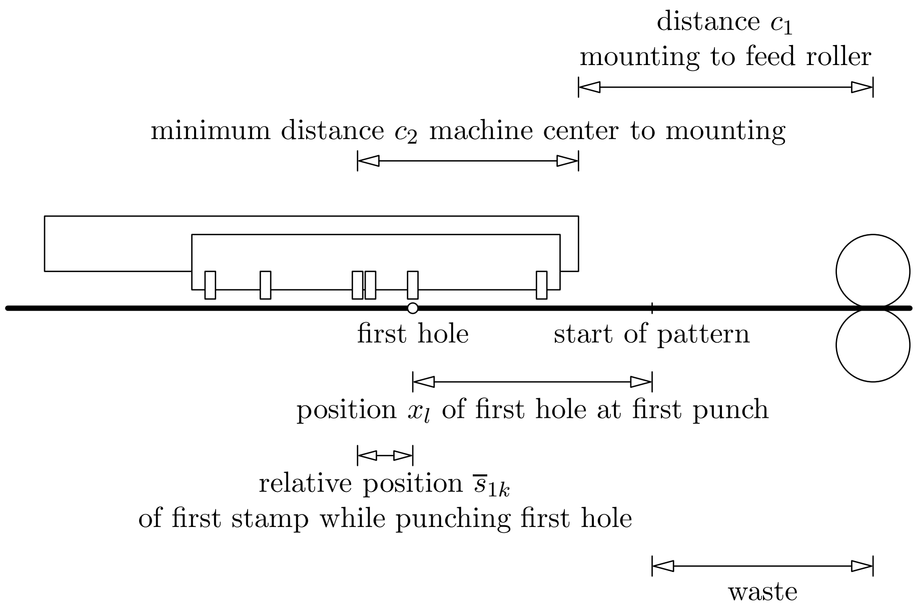

distance $c_1$
mounting to feed roller
minimum distance $c_2$ machine center to mounting
first hole
start of pattern
position $x_l$ of first hole at first punch
relative position $\overline{s}_{1k}$
of first stamp while punching first hole
waste

**Figure 3.2:** Waste resulting in the beginning of the punching process.

## Waste

Waste results in the beginning of the process, where a certain amount of metal is needed between the feed roller and the punching machine and thereby the first hole in the pattern (see Figure 2.1), even when the length of metal before the first hole as given in the pattern is shorter.

Let $c_1$ denote the distance between the mounting of the punching machine and the feed roller, and let $c_2$ denote the minimum distance between the center of the machine and the end of the mounting. Furthermore, let $x_l$ be the position of the first hole (with respect to its $x$-position) in the first punch (with respect to the sequence of punches), and $\overline{s}_{1k}$ be the relative position of the punch $k$ punching this hole (see Figure 3.2). Then the resulting waste is given by

$$w = c_1 + c_2 + \overline{s}_{1k} - x_l.$$

As $x_l$ is the sum of the absolute machine position $s_{1\bar{j}}$ and the relative position $\overline{s}_{1k}$ of punch $k$ during the first punch, this can be rewritten as

$$w = c_1 + c_2 - s_{1\bar{j}}. \tag{3.26}$$

Thus, waste only depends on the absolute position $s_{1\bar{j}}$ in the first punching step (apart from an additive constant), and can be minimized by maximizing $s_{1\bar{j}}$.

### Combining Speed and Waste

The values $c$ and $w$ of the objective functions for speed and waste, respectively, can be combined to yield a single objective function. As the absolute values of (time) costs and waste are hard to relate to each other, we relate each of them to the whole set of solutions before combining them. That is, given a set of solutions $\mathcal{S} = \{S_i\}_{1 \leq i \leq n}$ with costs $c_i$ and waste $w_i$ each, we define the overall objective function to be the convex combination

$$f_{\mathcal{S}}(c, w) = (1 - \lambda_{\mathrm{w}}) \frac{c - \min c_i}{\max\{(\max c_i - \min c_i), 1\}} + \lambda_{\mathrm{w}} \frac{w - \min w_i}{\max\{(\max w_i - \min w_i), 1\}}, \quad (3.27)$$

where $0 \leq \lambda_{\mathrm{w}} \leq 1$ is the "weight" of the optimization of waste.

This combination is especially useful "a posteriori" to find the best solution in a given set of solutions. However, this is not available in many cases and hard to deal with in theory; thus, we focus on the simpler version of only minimizing speed costs. In practice, speed plays a more crucial role than waste ($\lambda_{\mathrm{w}}$ is often chosen $< 0.1$), and waste can often be optimized using local heuristics after optimizing for speed globally (see section 6.4), as it only depends on the beginning of the process. The influence of the parameter $\lambda_{\mathrm{w}}$ is discussed in greater detail in section 7.7.

## 3.4. Practical Parameter Choices

In this section, some practical parameter choices as given by the commissioning companies regarding constraints on the punching machine are presented, as well as previous variants of the punching problem.

### Punching Machine

The punching machine primarily considered in this work consists of twelve punches indexed from 1 to 12, with $2j - 1$ denoting punches with higher $y$-positions and $2j$ denoting the corresponding lower punches for $1 \leq j \leq 6$. The double stamp is comprised by the punches 5 to 8 (see Figure 2.3).

Pairs of punches always have the same $x$-positions, that is,

$$\underline{X}_{2j-1,2j} = \overline{X}_{2j-1,2j} = 0.$$

The distances between other pairs of stamps are restricted by the width of the stamps (typically 1100 for ordinary stamps and 1800 for the double stamp) and the maximum distance from the machine center to the outer stamps (typically 14000). By the triangle inequality

$$\underline{X}_{jk} \geq \underline{X}_{jl} + \underline{X}_{lk}$$

(the minimum distance between two punches $j$ and $k$ is at least the sum of the minimum distances to another punch $l$) and

$$\overline{X}_{jk} \leq \overline{X}_{jl} + \overline{X}_{lk},$$

this results in

$$\underline{X} = \begin{pmatrix} 0^{(2)} & 1100^{(2)} & 2200^{(2)} & 2900^{(2)} & 4000^{(2)} & 5100^{(2)} \\ & 0^{(2)} & 1100^{(2)} & 1800^{(2)} & 2900^{(2)} & 4000^{(2)} \\ & & 0^{(2)} & 700^{(2)} & 1800^{(2)} & 2900^{(2)} \\ & & & 0^{(2)} & 1100^{(2)} & 2200^{(2)} \\ & & & & 0^{(2)} & 1100^{(2)} \\ & & & & & 0^{(2)} \end{pmatrix} \tag{3.28}$$

and

$$\overline{X} = \begin{pmatrix} 0^{(2)} & 12550^{(2)} & 13650^{(2)} & 14350^{(2)} & 26900^{(2)} & 28000^{(2)} \\ & 0^{(2)} & 12550^{(2)} & 13250^{(2)} & 25800^{(2)} & 26900^{(2)} \\ & & 0^{(2)} & 700^{(2)} & 13250^{(2)} & 14350^{(2)} \\ & & & 0^{(2)} & 12550^{(2)} & 13650^{(2)} \\ & & & & 0^{(2)} & 12550^{(2)} \\ & & & & & 0^{(2)} \end{pmatrix}, \tag{3.29}$$

where we use the shorthand notion

$$A^{(2)} = \begin{pmatrix} A & A \\ A & A \end{pmatrix} \tag{3.30}$$

to denote quadruples of entries in the matrices, resulting from the equivalence of punches of a single stamp with respect to their $x$-positions. As

$$s_{ij} - s_{ik} = -(s_{ik} - sij) \leq -\overline{X}_{kj},$$

the lower halves of the matrices are given by

$$\underline{X}_{jk} = -\overline{X}_{kj} \quad \text{and} \quad \overline{X}_{jk} = -\underline{X}_{kj} \qquad \text{for } j > k.$$

The maximum $x$-movement between consecutive punching steps is typically given by

$$X = 4000.$$

**Example 3.6** (Example 3.2 continued)**.** We demonstrate that the result shown in Example 3.2 complies with the constraints regarding $x$-positions and $x$-movements as given by eqs. (3.15) and (3.18).

- Considering the first punching step ($i = 1$), the $x$-positions of the punches are

$$(s_{1j})_j \quad = \quad \begin{pmatrix} 11750^{(2)} & 6050^{(2)} & 350^{(2)} & -350^{(2)} & -6750^{(2)} & -7850^{(2)} \end{pmatrix}.$$

  Note that two punches $2j-1$ and $2j$ making up a stamp always have the same $x$-positions, which is why we use the shorthand notation from eq. (3.30) again, where in the case of row vectors it is supposed to mean $a^{(2)} = (a \quad a)$.

  The distances $s_{1j} - s_{1k}$ of the punches comply with eq. (3.15) as

$$\underline{X} \le (s_{1j} - s_{1k})_{jk} = \begin{pmatrix} 0^{(2)} & 5700^{(2)} & 11400^{(2)} & 12100^{(2)} & 18500^{(2)} & 19600^{(2)} \\ -5700^{(2)} & 0^{(2)} & 5700^{(2)} & 6400^{(2)} & 12800^{(2)} & 13900^{(2)} \\ -11400^{(2)} & -5700^{(2)} & 0^{(2)} & 700^{(2)} & 7100^{(2)} & 8200^{(2)} \\ -12100^{(2)} & -6400^{(2)} & -700^{(2)} & 0^{(2)} & 6400^{(2)} & 7500^{(2)} \\ -18500^{(2)} & -12800^{(2)} & -7100^{(2)} & -6400^{(2)} & 0^{(2)} & 1100^{(2)} \\ -19600^{(2)} & -13900^{(2)} & -8200^{(2)} & -7500^{(2)} & -1100^{(2)} & 0^{(2)} \end{pmatrix} \le \overline{X}$$

  holds. For instance, the distance between punch 1 and punch 3 is

$$\underline{X}_{13} = 1100 \le s_{11} - s_{13} = 5700 \le 12550 = \overline{X}_{13}.$$

- Furthermore, the $x$-movements of the relative positions $\overline{s}_{ij}$ in steps $j = 1$ and $j = 2$ comply with eq. (3.18). By eq. (3.1) we get

$$(\overline{s}_{1j})_j = \begin{pmatrix} 12100^{(2)} & 6400^{(2)} & 700^{(2)} & 0^{(2)} & -6400^{(2)} & -7500^{(2)} \end{pmatrix}$$
$$(\overline{s}_{2j})_j = \begin{pmatrix} 11050^{(2)} & 6400^{(2)} & 400^{(2)} & 0^{(2)} & -6400^{(2)} & -7500^{(2)} \end{pmatrix}$$

  and thereby

$$-X = -4000 \le (\overline{s}_{2j} - \overline{s}_{1j})_j = \begin{pmatrix} -1050^{(2)} & 0^{(2)} & -300^{(2)} & 0^{(2)} & 0^{(2)} & 0^{(2)} \end{pmatrix} \le 4000 = X.$$

For each pair of punches, there is a minimum $y$-distance of the two punches (typically 820). Moreover, the corresponding punches of the double stamp must have the same $y$-position. Apart from that, there are no constraints on the $y$-positions relative to each other. This results in

$$\underline{Y} = \begin{pmatrix} 0 & 820 & & & & & & & & & & \\ & 0 & & & & & & & & & & \\ & & 0 & 820 & & & & & & & & \\ & & & 0 & & & & & & & & \\ & & & & 0 & 820 & 0 & & & & & \\ & & & & & 0 & & 0 & & & & \\ & & & & 0 & & 0 & 820 & & & & \\ & & & & & 0 & & 0 & & & & \\ & & & & & & & & 0 & 820 & & \\ & & & & & & & & & 0 & & \\ & & & & & & & & & & 0 & 820 \\ & & & & & & & & & & & 0 \end{pmatrix}, \tag{3.31}$$

where all empty entries mean $-\infty$. There are no upper bounds on the $y$-distances between punches except for the double stamp (being 0), so as for the lower halves of $\underline{X}$ and $\overline{X}$, $\overline{Y} = -\underline{Y}^t$ holds, that is,

$$\overline{Y} = \begin{pmatrix} 0 & & & & & & & & & & & \\ -820 & & & & & & & & & & & \\ & & 0 & & & & & & & & & \\ & & -820 & & & & & & & & & \\ & & & & 0 & & 0 & & & & & \\ & & & & -820 & 0 & & 0 & & & & \\ & & & & 0 & & 0 & & & & & \\ & & & & & 0 & -820 & 0 & & & & \\ & & & & & & & & 0 & & & \\ & & & & & & & & -820 & 0 & & \\ & & & & & & & & & & 0 & \\ & & & & & & & & & & -820 & 0 \end{pmatrix}, \tag{3.32}$$

where all empty entries mean $\infty$.

Usually, punches cannot cross the $x$-axis to an arbitrary extent, resulting in a minimum $y$-position

$$\underline{V}_{2j-1} = -500$$

for the upper punches and a maximum $y$-position

$$\overline{V}_{2j} = 500$$

for the lower punches. The other bounds can be set to $\underline{V}_{2j} = -\infty$ and $\overline{V}_{2j-1} = \infty$.

**Example 3.7** (Example 3.2 continued)**.** We demonstrate that the result shown in Example 3.2 complies with the constraints regarding $y$-positions as given by eqs. (3.16) and (3.17).

- Considering the first punching step ($i = 1$), the $y$-positions of the punches are

  $$\begin{aligned}(v_{1j})_j = (&1000, -1000, 500, -500, 500, -500,\\ &500, -500, 1000, -1000, 1000, -1000).\end{aligned}$$

  Obviously,

  $$v_{1,2j-1} - v_{1,2j} \geq 820$$

  holds for all $1 \leq j \leq 6$ and, considering the double stamp,

  $$v_{15} = v_{17} \quad \text{and} \quad v_{16} = v_{18}$$

  holds, so

  $$\underline{Y} \leq (v_{1j} - v_{1k})_{jk} \leq \overline{Y}$$

  as required by eq. (3.16) is fulfilled.
- The limits on the $y$-positions given by eq. (3.17) are kept as well, as

  $$v_{i,2j-1} \geq -500 = \underline{V}_{2j-1}$$

  and

  $$v_{i,2j} \leq 500 = \overline{V}_{2j}$$

  holds for all $1 \leq j \leq 6$.

The reference punch can be equivalently chosen to be either 7 or 8. Let us choose

$$\bar{j} = 7$$

here.

In most cases, there are no restrictions on the set of tools that can be assigned to a punch, resulting in

$$T_{jt} = 1$$

for all tools $t$ in the pattern.

## Variants

Previous versions of the punching machine had less capabilities which can also be modeled by adjusting parameters correspondingly.

**Static $y$-positions** In a previous version of the punching machine, the punches were not dynamically movable in $y$-direction at all, but had to be set to a fixed $y$-position throughout the whole batch. This can be modeled by ignoring all $y$-related parameters; for instance, by setting them to

$$y_n = \underline{Y}_{jk} = \overline{Y}_{jk} = \underline{V}_j = \overline{V}_j = 0,$$

and incorporating the actual $y_n$ into the tool specifications.

**Static $x$-positions** When the punches are not movable in $x$-direction, one can simply set

$$X = 0.$$

This was the case in the very first version of the punching machine, where the relative $x$-positions of the punches were set at the beginning and remained constant throughout a batch.

**Inter-dependent double stamp** There was a version of the machine where the punches 7 and 8 could only punch when the punches 5 and 6, respectively, also punched. While this cannot be expressed directly using the described model, pairs of holes having the same $x$-distance as the double stamp can be detected and grouped to a single hole punchable by an "artificial" double punch. This can be done for all occurring combinations of hole types with the best solution being chosen in the end.

## 3.5. Mixed-Integer Linear Programs

The step punching problem (Definition 3.4) and the speed punching problem (Definition 3.5) consist of a linear objective function and linear equality and inequality constraints with some variables (namely $u_{jt}$, $h_{ijn}$, $h'_{ijn}$, $a_{ij}$, $\overline{a}_i$) being restricted to values in $\{0, 1\}$. Thus, these problems can be formulated as so-called *mixed-integer linear programs.*

**Definition 3.8** (Mixed-integer linear program)**.** Let $A \in \mathbb{R}^{m\times n}$ be a matrix, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ vectors, $I \subseteq \{1, \dots, n\}$ a set of indices, and $x \in \mathbb{R}^n$ a vector of variables. Then the corresponding (canonical) *mixed-integer linear program* (MILP) is a problem of the form

$$\begin{aligned}
(P) \qquad & \min && c^t x && \\
& \text{subject to} && Ax = b && (3.33)\\
& && x \geq 0 && (3.34)\\
& && x_j \in \mathbb{Z} \qquad \forall j \in I. && (3.35)
\end{aligned}$$

The *solution space* of a MILP is the set of vectors $x$ satisfying the conditions given by eqs. (3.33) to (3.35).

If no variables have integrality constraints ($I = \emptyset$) the MILP is a (regular) *linear program* (LP). If all variables must be integers ($I = \{1, \ldots, n\}$) the MILP is said to be an *integer linear program* (ILP).

The *LP relaxation* of a MILP $(P)$ is the program $(P)$ with $I$ set to $\emptyset$, that is, with the integrality constraints of all variables being removed.

*Remark.* Programs involving inequality constraints of the form

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$$

such as the punching problems can be converted to canonical MILPs by introducing "slack variables" $s_i \geq 0$ and adding constraints

$$\sum_{j=1}^{n} a_{ij} x_j - s_i = b_i.$$

Variables $x_j$ unrestricted in sign can be rewritten to restricted variables $x_j^+$, $x_j^- \geq 0$ using the constraint

$$x_j = x_j^+ - x_j^-$$

as shown by Papadimitriou and Steiglitz [1982, p. 28f.].

While regular linear programs can be solved in polynomial time [Karmarkar, 1984; Khachiyan, 1979] and the simplex method [Dantzig and Thapa, 1997] solves them efficiently, solving general MILPs exactly is $\mathbb{NP}$-*hard*. This follows directly from the fact that the special case of 0-1 integer programming (deciding whether there exists a feasible solution to $Ax = b$ when all variables $x_j$ are binary) is $\mathbb{NP}$-hard already [Karp, 1972]. Interestingly, when the number $n$ of variables is fixed, integer programming is solvable in polynomial time; $n$ appears in the exponent of the time-bounding polynomial, though [Lenstra, 1983]. $\mathbb{NP}$-hardness is defined and discussed in greater detail in section 4.2.

Given the theoretical complexity of integer linear programming, it is still of interest to search for solutions "intelligently" even if that might take a long time. We shortly outline the methods of *branch-and-bound*, *cutting planes*, and their combination *branch-and-cut* here.

## Branch-and-Bound

Branch-and-bound was proposed by Land and Doig [1960] and consists of two general steps.

1. In the *branching* step, the solution space $S$ is partitioned into several subproblems $S = S_1 \cup S_2 \cup \cdots \cup S_K$. Obviously, the optimal solution in $S$ can be determined from the solutions of the subproblems through

$$z^* = \min\{c^t x \mid x \in S\} = \min_{k \in \{1, \ldots, K\}} z_k^*$$

with the individual solutions

$$z_k^* = \min\{c^t x \mid x \in S_k\}.$$

By successively branching the solution space, an enumeration tree of subproblems emerges.

2. In the *bounding* step, the enumeration tree is pruned using bounds on the optimal objective value within a branch, that is,

$$\underline{z}_k \leq z_k^* \leq \overline{z}_k.$$

Such bounds can be used in several ways.

   a) *Pruning by optimality*: If $\underline{z}_k = \overline{z}_k$, then the optimal value $z_k^* = \underline{z}_k = \overline{z}_k$ is known and there is no need to subdivide $S_k$ into smaller subsets.

   b) *Pruning by bound*: If $\underline{z}_k > \overline{z}_j$ for some $j \neq k$, then $z_k$ cannot be the optimal solution $z^*$ of the whole problem, so the branch needs not be examined further.

   c) *Pruning by infeasibility*: If $S_k = \emptyset$, there is no need to subdivide $S_k$ further, either.

In the case of (mixed) integer linear programming, branching can naturally be done on integer variables, while bounds result from LP relaxations. Given a solution $x^0$ to the LP relaxation of $(P)$, we consider a component $x_i^0$, $i \in I$, that is not integral. If there is no such component, $x^0$ is already the optimal solution of the (non-relaxed) MILP. Otherwise, we split $(P)$ into the two subproblems

$$\begin{array}{llll} (P_1) & \min & c^t x & \\ & \text{subject to} & Ax = b & \\ & & x \geq 0 & \\ & & x_j \in \mathbb{Z} & \forall j \in I \\ & & x_i \leq \left\lfloor x_i^0 \right\rfloor & \end{array}$$

and

$$\begin{array}{llll} (P_2) & \min & c^t x & \\ & \text{subject to} & Ax = b & \\ & & x \geq 0 & \\ & & x_j \in \mathbb{Z} & \forall j \in I \\ & & x_i \geq \left\lfloor x_i^0 \right\rfloor + 1. & \end{array}$$

If $x_i$ is a binary variable, the two cases boil down to $x_i = 0$ and $x_i = 1$.

In each case, solving the LP relaxation of $(P_k)$ yields a lower bound $\underline{z}_k$ on the respective objective value $z_k^*$, as the relaxed problem can not have a worse objective value than $(P_k)$ itself. Upper bounds result from feasible solutions $x$ satisfying the integrality constraints (3.35).

There are several ways to "walk" through the enumeration tree and to choose branching variables. Dakin [1965] suggests branching in a *depth-first* manner which is memory-efficient. If storage is not critical, branching from the subproblem with the lowest lower bound (*best-first* search) seems reasonable, although that might not even outperform depth-first search in parallel computations [Clausen and Perregaard, 1996]. Recent work includes applications to nonconvex quadratic programming [Burer and Vandenbussche, 2008] and adaptations for specific problems such as the maximum diversity problem [Martí and Reinelt, 2011].

## Cutting Planes

Cutting planes were first used by Dantzig, Fulkerson, and Johnson [1954] for the traveling salesman problem and generalized by Gomory [1958]. They share with branch-and-bound the idea of solving the relaxed linear program of a MILP and introducing additional constraints. The difference is the kind of linear constraint being added, though. Instead of *splitting* the problem into several branches, cutting planes slightly *modify* the problem without excluding integer feasible points. That way, the optimal integer solution remains the same.

We give a brief outline of *Gomory cuts* here, while detailed explanations can be found in the original work of Gomory [1958, 1960] or the book by Papadimitriou and Steiglitz [1982, p. 326ff.].

For simplicity, we only deal with the case of integer linear programs here, that is, all variables shall be integer. The case of $I \subsetneq \{1, \dots, n\}$ is discussed in the book by Wolsey [1998], for instance.

When solving the LP relaxation of $(P)$ with a *primal simplex algorithm* [Dantzig and Thapa, 1997], an equation in the final tableau is of the form

$$x_i + \sum_{j \notin B} \bar{a}_{ij} x_j = \bar{b}_i, \tag{3.36}$$

where $B$ is the basis and $i \in B$. Suppose a variable is not integer, that is, $x_i \notin \mathbb{Z}$. As $x \geq 0$ and the integer part fulfills $\lfloor \bar{a}_{ij} \rfloor \leq \bar{a}_{ij}$, the relation

$$\sum_{j \notin B} \lfloor \bar{a}_{ij} \rfloor x_j \leq \sum_{j \notin B} \bar{a}_{ij} x_j$$

holds. Inserting this into (3.36) yields

$$x_i + \sum_{j \notin B} \lfloor \bar{a}_{ij} \rfloor x_j \leq \bar{b}_i.$$

As $x$ is integer, the left-hand side of this equation is integer, so

$$x_i + \sum_{j \notin B} \lfloor \bar{a}_{ij} \rfloor x_j \leq \left\lfloor \bar{b}_i \right\rfloor \tag{3.37}$$

follows. Subtracting (3.37) from (3.36) yields

$$\sum_{j \notin B} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j \geq \bar{b}_i - \left\lfloor \bar{b}_i \right\rfloor .$$

This can be formulated using a slack variable $s$ as

$$-\sum_{j \notin B} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j + s = \left\lfloor \bar{b}_i \right\rfloor - \bar{b}_i .$$

Because of its construction, adding this constraint to $(P)$ does not exclude integer points. Furthermore, the tableau remains dual feasible. This is utilized in the *fractional dual algorithm* to solve ILPs by successively adding Gomory cuts and applying the dual simplex algorithm.

Gomory [1958, 1960] showed that, using a certain row selection strategy and the lexicographic version of the dual simplex algorithm, the fractional dual algorithm terminates in a finite number of steps, either finding an integer solution or reporting that there is no feasible integer solution to $(P)$.

However, convergence speed and numerical stability might still be an issue with Gomory cuts, which is why problem-specific cutting planes are often used in practice. For instance, Grötschel, Martin, and Weismantel [1996] apply cutting planes to the Steiner tree packing problem arising in very-large-scale integration (VLSI) design.

Also notable are *intersection cuts* in disjunctive programming as introduced by Balas [1979]. An overview of these methods is given by Belotti et al. [2010]. A survey of cutting plane methods in general is presented by Marchand et al. [2002].

### Branch-and-Cut

The methods of branch-and-bound and cutting planes can be combined to the so-called *branch-and-cut* method. The traveling salesman problem (TSP) was one of the first problems to be approached using branch-and-cut [Grötschel and Holland, 1991; Padberg and Rinaldi, 1991]. An overview of the topic is given by Mitchell [2002], for instance.

Branch-and-cut can be outlined as the following iterative process.

1. Maintain a list of open MILPs with corresponding lower bounds on their objective value (the *active nodes*) initialized to the given MILP $(P)$.
2. Select an active node and solve its LP relaxation.
3. If the solution is not integer, add cutting planes to the relaxation.
4. Prune the enumeration tree as described in section 3.5, deleting corresponding active nodes if their lower bounds cannot beat the feasible objective value achieved so far.
5. Partition the solution space and add corresponding active nodes.

This method has been successfully applied to a wide range of combinatorial optimization problems [Caprara and Fischetti, 1997; Jünger, Reinelt, and Thienel, 1995] and is also employed in many solvers such as *MINTO* [Nemhauser, Savelsbergh, and Sigismondi, 1994], *SCIP* [Achterberg, 2009], *GUROBI* [Gurobi Optimization, Inc. 2012], and *CPLEX* [ILOG CPLEX Division, 2007].

Despite recent progress in this area, the number of integer variables in the punching problem is still too large to be solved exactly in practice (see section 7.1), which is why we develop heuristics in chapters 5 and 6. However, we use CPLEX to compute exact solutions of very small instances of the punching problem to evaluate the quality of our heuristic results in sections 7.3 and 7.4.

CHAPTER 4

# Computational Complexity

In this chapter, we analyze the computational complexity of the step punching problem. We focus on a simplified version with a fixed machine configuration, which corresponds to setting $X = 0$. Furthermore, holes shall have distinct $x$-positions and a single hole type. Thus, we deal with the following problem.

**Definition 4.1** (Simplified punching problem)**.** The *simplified punching problem with $J$ punches* ($\mathrm{SPP}_J$) is defined as follows. Given a set of $N$ *holes* $\{x_1, \dots, x_N\}$ and the relative positions $\overline{s}_j$ of punches $j$, $1 \leq j \leq J$, determine the minimum number $I$ of *machine positions* $s_i$, $1 \leq i \leq I$, such that all holes can be punched from them, that is,

$$\bigcup_{i=1}^{I} \{s_i + \overline{s}_j \mid 1 \leq j \leq J\} \supseteq \{x_1, \dots, x_N\}. \tag{4.1}$$

*Remark.* In the previous definition, $s_i + \overline{s}_j$ corresponds to the absolute position of punch $j$ in punching step $i$. Equation (4.1) demands that the $x$-positions of all holes are covered by these punch positions.

As described in section 4.1, this problem can be formulated using bipartite graphs as well. It is related to known $\mathbb{NP}$-hard problems as described in section 4.3, which is why we give a short introduction to $\mathbb{NP}$-hardness in section 4.2. However, as we show in section 4.4 using dynamic programming, the simplified punching problem is solvable in polynomial time (in the number of holes) under the additional assumption that the number of holes in each interval of machine length is limited.

Despite the restriction to the simplified version of the punching problem, the principles discussed in this chapter should be generalizable to problems with overlapping $y$-positions of holes and several different hole types.

To analyze the asymptotic behavior of algorithms in this thesis, we use the following notation as described by Knuth [1976].

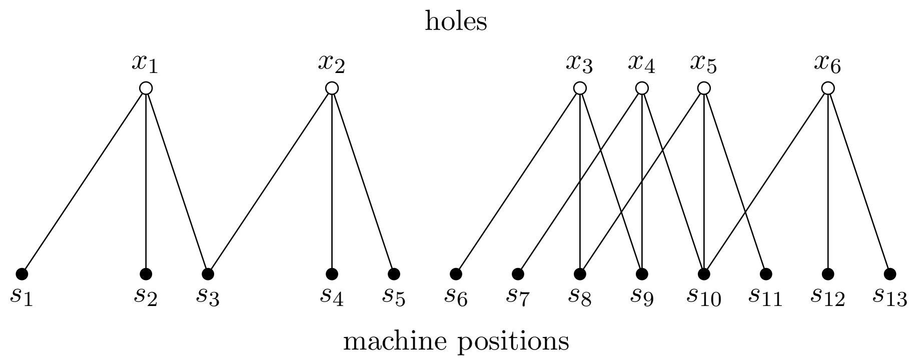


**Figure 4.1:** Bipartite graph corresponding to a punching problem $\mathrm{SPP}_3$ with three punches. Holes are at positions $x_1, \dots, x_6$, while machine positions are at $s_1, \dots, s_{13}$.

**Definition 4.2** ($\mathcal{O}$-notation)**.** Let $f : \mathbb{N} \to \mathbb{N}$ be a function. Then $\mathcal{O}(f(n))$ denotes the set of all functions $g : \mathbb{N} \to \mathbb{N}$ such that there exist positive constants $C$ and $n_0$ with

$$|g(n)| \leq Cf(n) \qquad \text{for all } n \geq n_0.$$

This definition can easily be generalized to multivariate functions. To conform with existing practice, we write

$$g(n) = \mathcal{O}(f(n))$$

when we formally mean $g(n) \in \mathcal{O}(f(n))$.

## 4.1. Bipartite Graph Formulation

The relations between holes and machine positions in $\mathrm{SPP}_J$ can be represented in a bipartite graph

$$G = (\{x_1, \dots, x_N\} \uplus \{s_1, \dots, s_I\}, E)$$

with edges

$$E = \{\{x_n, s_i\} \mid \exists 1 \leq j \leq J : x_n = s_i + \overline{s}_j\},$$

that is, there is an edge between a hole $x_n$ and a machine position $s_i$ if and only if $x_n$ can be punched from $s_i$ by any punch. An example is shown in Figure 4.1.

*Remark.* To simplify notation, we use $x$-coordinates (that is, real numbers) to identify vertices corresponding to both holes and machine positions. The two sets of vertices should still be distinct, that is, a vertex $x_n$ corresponding to a hole should be different from a vertex $s_i$ corresponding to a machine position even when their values as real numbers are the same. There are two ways to deal with this formal problem:

- We simply "de-identify" $x_n \neq s_i$ for all $n$, $i$.

- Equivalently, we could demand $\overline{s}_j \neq 0$ for all $1 \leq j \leq J$ to ensure $x_n \neq s_i$. This can be done without loss of generality, as changing all relative punch positions from $\overline{s}_j$ to $\overline{s}_j + \varepsilon$ for some $\varepsilon$ only affects the values of the resulting machine positions ($s_i = x_n - \overline{s}_j$ for some $n$, $j$), but not their number (which is the decisive quantity in $\mathrm{SPP}_J$).

However, this is not a real issue in our conclusions.

Note that the degree $\deg_G(s_i)$ of a machine position vertex $s_i$ in $G$ is the number of holes that can be punched simultaneously from that position, while the degree $\deg_G(x_n)$ of a hole vertex $x_n$ is $J$ (the number of punches) for all $n$.

### Hitting Sets

The problem of minimizing the number of machine positions in $\mathrm{SPP}_J$ corresponds to finding a minimum hitting set in $G$ by the vertices $\{s_1, \dots, s_I\}$, which is defined as follows.

**Definition 4.3** (Hitting set problem)**.** Let $G = (U \uplus V, E)$ be a bipartite graph with a distinguished set $V$ of vertices. The *hitting set problem* is to find the minimum cardinality of a subset $V' \subseteq V$ such that every vertex $u \in U$ is covered by a vertex in $V'$, that is,

$$\forall u \in U \; \exists v' \in V' : \{u, v'\} \in E.$$

In the *decision variant* of the hitting set problem, we are also given a number $k$ and ask if there exists a hitting set of cardinality not greater than $k$.

Although the hitting set problem is $\mathbb{NP}$-hard [KARP, 1972], this formulation can be used to develop heuristics for solving the general case (see chapter 5). It is particularly useful when there are only two punches. In this case, the underlying hitting set problem can be solved efficiently as follows.

### Two Punches

Considering the case of two punches ($J = 2$), they can be assumed to be placed at relative positions

$$\overline{s}_1 > 0 > \overline{s}_2 \tag{4.2}$$

without loss of generality.

Note that the vertices of $G$ can be ordered, as they all represented $x$-coordinates of either holes or machine positions.

**Proposition 4.4.** The graph $G$ induced by $\mathrm{SPP}_2$ is cycle-free.

*Proof.* To each hole vertex $x_n$ there are exactly two incident edges, namely from the machine position vertices $x_n - \overline{s}_1 < x_n$ and $x_n - \overline{s}_2 > x_n$. Conversely, to each machine position vertex $s_i$ there are at most two incident edges, namely from the hole vertices $s_i + \overline{s}_1 > s_i$ and $s_i + \overline{s}_2 < s_i$. (All inequalities hold because of (4.2).) Thus, if a vertex $v$ in $G$ is connected to two vertices $v_1 < v_2$, then

$$v_1 < v < v_2 \tag{4.3}$$

holds.

Now let $C$ be a cycle in $G$ and consider its smallest ("leftmost") vertex $v$ in $C$. As $C$ is a cycle, $v$ has to be connected to two vertices $v_1$, $v_2$ in $C$. Assume without loss of generality that $v_1 < v_2$ holds, which in turn implies (4.3), a contradiction to $v$ being the smallest vertex in $C$. ☐

**Corollary 4.5.** $G$ is a union of paths, as $G$ is cycle-free and the degree of every vertex in $G$ is not greater than 2.

*Remark.* This is not necessarily true when $J \geq 3$. As shown in Figure 4.1, there is a cycle $(x_3, s_8, x_5, s_{10}, x_4, s_9, x_3)$.

Note that the minimum hitting set in a path

$$(v_1, v_2, \ldots, v_{r-1}, v_r) \tag{4.4}$$

is simply given by $\{v_2, \ldots, v_{r-1}\}$.

**Theorem 1.** SPP$_2$ *can be solved using* $\mathcal{O}(N)$ *time and space.*

*Proof.* The graph $G$ can be constructed in $\mathcal{O}(N)$ time using $\mathcal{O}(N)$ space. A minimum hitting set in $G$ is given by the union of minimum hitting sets of the paths in $G$. To solve SPP$_2$, we only have to sum up the lengths $(r-2)$ of the minimum hitting sets $\{v_2, \ldots, v_{r-1}\}$ in individual paths of the form (4.4) in $G$, which can be done in linear time. ☐

## 4.2. $\mathbb{NP}$-Completeness

We leave an extensive definition and discussion of $\mathbb{NP}$-complete problems to the book of Garey and Johnson [1979]. In this context, we only mention a few important definitions as given by Papadimitriou and Steiglitz [1982, p. 342ff.].

**Definition 4.6** ($\mathbb{P}$)**.** A recognition problem $P$ is said to be in the class $\mathbb{P}$ if there exists an algorithm $A$ that decides $P$ in polynomial time.

**Definition 4.7** ($\mathbb{NP}$)**.** A recognition problem $P$ is said to be in the class $\mathbb{NP}$ if there exists a polynomial $p$ and an algorithm $A$ (the *certificate-checking algorithm*) such that a string $x$ is a *yes* instance of $P$ if and only if there exists a string $c(x)$ (the *certificate*) with $|c(x)| \leq p(|x|)$ and the property that $A$ with input $c(x)$ returns *yes* after at most $p(|x|)$ steps.

*Remark.* It is crucial that both the size of the certificate and the running time of the certificate-checking algorithm are polynomial in the size of the input $x$. Without loss of generality, both can be assumed to be bounded by the same polynomial $p$.

The notions of algorithms, their input, output, and running time are based on the concept of Turing machines introduced by TURING [1936]. Intuitively speaking, problems in $\mathbb{NP}$ are (potentially) *hard to solve*, but solutions are *easy to verify* given a corresponding certificate. Problems in $\mathbb{NP}$ are particularly interesting because they can be reduced to a subclass of problems. This was first approached by KARP [1972].

**Definition 4.8** (Polynomial-time reductions and transformations)**.** A problem $P_1$ *polynomially reduces* to $P_2$ if there exists a polynomial-time algorithm $A_1$ for $P_1$ that uses as a subroutine "at unit cost" an algorithm $A_2$ for $P_2$.

A problem $P_1$ *polynomially transforms* to $P_2$ if there is a polynomial-time reduction from $P_1$ to $P_2$ with just one call of the subroutine for $P_2$ at the end of the algorithm for $P_1$.

**Definition 4.9** ($\mathbb{NP}$-hardness)**.** A problem $P$ is said to be $\mathbb{NP}$-*hard* if all problems in $\mathbb{NP}$ polynomially transform to $P$.

*Remark.* An $\mathbb{NP}$-*hard* problem does neither have to be in $\mathbb{NP}$ itself (such as the halting problem introduced by TURING [1936]) nor does it have to be a recognition problem (such as the hitting set problem).

**Definition 4.10** ($\mathbb{NP}$-completeness)**.** A recognition problem $P$ is said to be $\mathbb{NP}$-*complete* if $P$ is in $\mathbb{NP}$ and $P$ is $\mathbb{NP}$-hard.

Due to polynomial-time reductions to $\mathbb{NP}$-complete problems, if any $\mathbb{NP}$-complete problem can be solved in polynomial time, all problems in $\mathbb{NP}$ can. However, the question whether $\mathbb{NP}$ equals $\mathbb{P}$ is still open [FORTNOW, 2009]. The majority of researchers believes $\mathbb{NP} \neq \mathbb{P}$, though [GASARCH, 2012], and looking for efficient algorithms to solve $\mathbb{NP}$-hard problems seems rather hopeless.

## 4.3. Related Problems and Approximations

As stated in section 4.1, the simplified punching problem can be reduced to the hitting set problem in bipartite graphs, which is $\mathbb{NP}$-hard. This is in turn equivalent to the following set cover problem.

**Definition 4.11** (Set cover problem)**.** Given a set of elements $\mathcal{U}$ (the *universe*) and a set $\mathcal{S}$ of $n$ sets $\mathcal{S} = \{S_1, \ldots, S_n\}$ with

$$\mathcal{U} = \bigcup_{i=1}^{n} S_i,$$

the *set cover problem* (SC) is to find the smallest number $k$ of sets that still cover $\mathcal{U}$, that is, $\mathcal{U} = \bigcup_{i \in K} S_i$ with a set $K \subseteq \{1, \ldots, n\}$ and $|K| = k$.

The equivalence to the hitting set problem in a bipartite graph $G = (U \uplus V, E)$ can be seen by relating

$$U = \mathcal{U}, \quad V = \mathcal{S} = \{S_1, \dots, S_n\},$$

and edges representing the inclusion of elements in sets, that is,

$$E = \{\{u, S_i\} \mid u \in U, u \in S_i\}\,.$$

The minimum set cover corresponds to the minimum hitting set by vertices in $V$.

The set cover problem can be formulated as the integer linear program (see section 3.5)

$$\begin{aligned}
(SC) \qquad & \min & & \sum_{S \in \mathcal{S}} x_S & & \\
& \text{subject to} & & \sum_{S: u \in S} x_S \geq 1 & & \forall u \in U \qquad (4.5)\\
& & & x_S \in \{0, 1\} & & \forall S \in \mathcal{S}.
\end{aligned}$$

In this program, $x_S$ reflects the decision whether set $S$ should be part of the cover. Equation (4.5) ensures that every element $u \in U$ is covered by at least one set in the cover [Vazirani, 2001, p. 108f.].

The ILP formulation $(SC)$ is particularly interesting because it yields an approximation algorithm for SC. By relaxing the integer condition of the variables $0 \leq x_S \leq 1$ in $(SC)$ and solving the corresponding linear program $(SCR)$ in polynomial time, we get a *fractional cover* of $\mathcal{U}$. Lovász [1975] proved that the *integrality gap* of this relaxation is the $n^{\text{th}}$ harmonic number defined by

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1,$$

that is, the optimal value $z$ of $(SC)$ is bound by the optimal value $z'$ of $(SCR)$ through

$$z \leq H_n z'.$$

This yields a $\log n$-approximation algorithm for SC, where the approximation ratio of an algorithm is defined as follows [Papadimitriou and Steiglitz, 1982, p. 409].

**Definition 4.12** (Approximation algorithm)**.** Let $P$ be a minimization problem with positive cost function $c$, and let $A$ be an algorithm returning a feasible solution $f_A(I)$ for each instance $I$ of $P$. Denote the optimal solution for instance $I$ by $f^*(I)$. Then $A$ is called an *$\varepsilon$-approximation algorithm* if

$$\frac{c(f_A(I) - c(f^*(I))}{c(f^*(I))} \leq \varepsilon$$

for all instances $I$ of $P$.

The fractional cover resulting from the relaxed linear program ($SCR$) can be turned into an approximate feasible cover for ($SC$) via *randomized rounding* as described by Raghavan and Tompson [1987]. Using *conditional probabilities*, this can be turned into a deterministic algorithm [Young, 1995].

However, Lovász [1975] already presented a greedy algorithm that achieves the same $H_n$ approximation ratio by choosing the set that contains the largest number of uncovered elements at each stage.

There is little hope that there is a significantly better approximation algorithms, as the following lower bounds on the approximation ratio have been proven. Lund and Yannakakis [1994] showed that set covering cannot be approximated to within a ratio of $\frac{1}{2}\log_2 n$ in polynomial time, unless $\mathbb{NP}$ has quasi-polynomial time algorithms (which is believed not to be a case, although this is a stronger assumption than $\mathbb{NP} \neq \mathbb{P}$). Feige [1998] improved this bound to $(1 - o(1))\ln n$. Raz and Safra [1997] showed a lower bound of $c \ln n$ under the weaker assumption that $\mathbb{NP} \neq \mathbb{P}$. Alon, Moshkovitz, and Safra [2006] improved the constant $c$ further.

Consequently, the greedy algorithm seems like a reasonable approach to the set covering problem, which is why we make use of it in the heuristics presented in chapter 5.

## 4.4. Dynamic Programming Solution

Even though the general hitting set problem is $\mathbb{NP}$-hard, we show that its special case of the simplified punching problem is solvable in polynomial time under an additional assumption. We already proved this for two punches (see Theorem 1). In this section, we present a solution to the general case. However, to simplify notations, we only formulate the case of three punches ($J = 3$) here. Still, the presented algorithm can easily be generalized to an arbitrary (yet constant) number $J$ of punches.

Without loss of generality, let $\overline{s}_1 = b$ (the right punch), $\overline{s}_2 = 0$ (the middle punch), and $\overline{s}_3 = -a$ (the left punch). Furthermore, we assume that the holes are sorted in ascending order, that is, $x_1 < x_2 < \cdots < x_N$. Note that there are no equal positions because the holes are given as a set according to Definition 4.1.

The additional assumption regards the number of holes in each interval of the size of the machine, $a + b$. This number shall be bounded by a constant $C$, that is,

$$|\{x_1, \ldots, x_N\} \cap [z, z + a + b]| \leq C \tag{4.6}$$

for each start $z$ of an interval of length $a + b$. This can be interpreted as a constraint on the "density" of the given hole pattern.

**Definition 4.13** (Simplified punching problem with density constraint)**.** The *simplified punching problem with $J$ punches and density $C$* ($\mathrm{SPP}_{J,C}$) is defined as $\mathrm{SPP}_J$ with the additional constraint given by (4.6).

*Remark.* When the hole positions $x_1, \ldots, x_N$ are integers, constraint (4.6) is fulfilled with $C = a + b + 1$.

We present a solution of $\mathrm{SPP}_{3,C}$ using dynamic programming. The crucial data that is constructed is the number of punching steps that are needed to punch all holes up to $n \in \{1, \dots, N\}$ and a set $S$ of holes $> n$ that are punched as well together with holes $\leq n$. Specifically, the algorithm builds up values

$$A_n^S, \quad n \in \{1, \dots, N\}, S \subseteq \{x_1, \dots, x_N\}$$

representing the minimum number of punching steps to punch holes $\{1, \dots, n\}$, and each hole $m \in S$ is punched in a punching step together with at least one hole $k \leq n$. This implies that the position of the rightmost hole in $S$ can be $x_n + a + b$ at most,

$$\max S \leq x_n + a + b. \tag{4.7}$$

Thus, all holes in $S$ lie within an interval $[x_n, x_n + a + b]$, so $S$ is actually restricted by

$$S \subseteq \{x_1, \dots, x_N\} \cap [x_n, x_n + a + b].$$

Therefore, the size of $S$ is limited by $|S| \leq C$ because of (4.6).

Additionally, we distinguish the punches that punch the "last" hole $n$ in dynamic programming. That is, we introduce variables

$$L_n^S, \quad M_n^S, \quad R_n^S$$

analogously to $A_n^S$ with the additional constraint that hole $n$ is punched by the left, middle, or right punch, respectively.

The initial conditions are

$$A_0^\emptyset = 0, \tag{4.8}$$

as an empty set of holes can be punched in 0 steps, and

$$L_0^S = M_0^S = R_0^S = \infty \quad \text{for any } S, \tag{4.9}$$

as there can be no restrictions on holes being punched by certain punches when there are no holes punched. Furthermore,

$$L_n^S = M_n^S = R_n^S = \infty \quad \text{for } \max S > x_n + a + b \tag{4.10}$$

by (4.7).

When there is no particular restriction on the punch being used to punch hole $n$, the number of needed punching steps until hole $n$ is given by the minimum over any assigned punch to hole $n$,

$$A_n^S = \min\{L_n^S, M_n^S, R_n^S\} \tag{4.11}$$

with an arbitrary set $S$ of additional holes.

Now consider $L_n^S$. When the current hole $n$ is restricted to use the left punch, the punching of $n$ cannot be performed together with another hole $< n$. When reducing the case to the punching $A_{n-1}^{S'}$ of all holes up to hole $n-1$, we only have to decide which holes in $S$ shall be punched together with a punch happening in the process of $A_{n-1}^{S'}$. We do this by enumerating all possible subsets $S' \subseteq S$. The rightmost hole that can be punched in the process of punching the holes up to hole $n-1$ is at position $x_{n-1}+a+b$, so we can restrict $S'$ to the interval

$$I_{n-1} = [x_{n-1}, x_{n-1}+a+b].$$

When punching the hole at $x_n$ with the left punch, the holes at $x_n+a$ and $x_n+a+b$ are punched as well, so we can exclude them from further restrictions. For shorter notation, let

$$X(p) = \{p-a, p, p+b\}$$

denote the set of hole positions punchable from position $p$. This results in

$$L_n^S = \min_{S' \subseteq S \cap I_{n-1}} \underbrace{1}_{\substack{\text{current step}\\ \text{punching } X(x_n+a)}} + \underbrace{A_{n-1}^{S' \setminus X(x_n+a)}}_{\text{holes punched until } n-1} + \underbrace{F(S \setminus (S' \cup X(x_n+a)))}_{\text{remaining holes}}, \quad (4.12)$$

where $F(T)$ is defined as the minimum number of punching steps for punching all holes in $T \subseteq \{x_1, \ldots, x_N\}$.

The function $F(T)$ can be computed by full enumeration of all assignments $f(t)$ from holes $t \in T$ to punches $j \in \{1,2,3\}$, that is,

$$F(T) = \min_{f:T \to \{1,2,3\}} \left| \{t - \overline{s}_{f(t)} \mid t \in T\} \right|. \quad (4.13)$$

Note that $t - \overline{s}_{f(t)}$ is the machine position when punching a hole at position $t$ with a punch $f(t)$ at relative position $\overline{s}_{f(t)}$. The value of $F(\{x_1, \ldots, x_N\})$ would be the solution to the whole problem $\mathrm{SPP}_{3,C}$ computed in exponential time. However, we only apply $F$ to sets of size at most $C$ in our algorithm, which needs constant time.

When the current hole $n$ is restricted to use the middle punch, the punching might be performed together with a previous hole at position $x_n - a$ punched by the left punch. Therefore, we consider the minimum number $L'$ of steps to punch all holes up until $x_n - a$, where the hole at position $x_n - a$ is punched by the left punch and all remaining holes from position $x_n - a$ to $x_{n-1}$ are punched as well. To simplify notation, we define

$$\mathrm{index}(z) = \begin{cases} k & \text{if } x_k = z \\ 0 & \text{otherwise} \end{cases}$$

to be the (index of the) hole at position $z$. Furthermore, let

$$\mathrm{range}(z, n) = [z, x_{n-1}] \cap \{x_1, \ldots, x_N\}$$

be the set of $x$-positions of holes before hole $n$ with $x$-position at least $z$. Then $L'$ can be expressed as

$$L' = L^{S' \cup \text{range}(x_n - a, n)}_{\text{index}(x_n - a)}.$$

With the rest being similar to $L^S_n$, this results in

$$\begin{aligned} M^S_n = \min_{S' \subseteq S \cap I_{n-1}} \min \Big\{ & 1 + A^{S' \setminus X(x_n)}_{n-1} + F(S \setminus (S' \cup X(x_n))), \\ & \underbrace{L^{S' \cup \text{range}(x_n - a, n)}_{\text{index}(x_n - a)}}_{\substack{\text{hole at } x_n - a \text{ with left punch,} \\ \text{rest punched until } x_{n-1}}} + F(S \setminus (S' \cup X(x_n))) \Big\}. \end{aligned} \tag{4.14}$$

Note that when there is no hole at position $x_n - a$ to punch the current hole $n$ with in a single step, $\text{index}(x_n - a) = 0$ and therefore $L' = \infty$ by eq. (4.9). Then, the minimum is achieved by the first of the two expressions, assuming correctly that hole $n$ has to be punched in a separate step from the holes up to $n - 1$.

Likewise,

$$\begin{aligned} R^S_n = \min_{S' \subseteq S \cap I_{n-1}} \min \Big\{ & 1 + A^{S' \setminus X(x_n - b)}_{n-1} + F(S \setminus (S' \cup X(x_n - b))), \\ & \underbrace{M^{S' \cup \text{range}(x_n - b, n)}_{\text{index}(x_n - b)}}_{\text{hole at } x_n - b \text{ with middle punch}} + F(S \setminus (S' \cup X(x_n - b))), \\ & \underbrace{L^{S' \cup \text{range}(x_n - a - b, n)}_{\text{index}(x_n - a - b)}}_{\text{hole at } x_n - a - b \text{ with left punch}} + F(S \setminus (S' \cup X(x_n - b))) \Big\} \end{aligned} \tag{4.15}$$

expresses the fact the a hole can be punched in a new step or with the right punch together with a hole punched by the left or middle punch.

The solution to $\text{SPP}_{J,C}$ is simply given by $A^{\emptyset}_N$, the minimum number of punching steps to punch all holes $\{x_1, \dots, x_N\}$ with no extra holes.

Algorithm 4.1 outlines the whole dynamic programming approach, incorporating eqs. (4.8), (4.9), (4.11), (4.12), (4.14) and (4.15). It uses recursive calls to proxy the underlying variables $L^S_n$, $M^S_n$, $R^S_n$ that are used to cache known results in order to avoid repeated computations. An implementation of the algorithm in Python is presented in appendix C.

**Proposition 4.14.** Algorithm 4.1 terminates.

*Proof.* On each invocation of L, M, R, the functions A, M, or L are potentially called with parameter $n$ reduced by at least 1. The function $F$ can obviously be computed in finite time. Calling A will potentially invoke L, M, or R with the same value of $n$, but then $n$ will be reduced by at least 1. Thus, the depth of the recursion tree (see Figure 4.2) is bounded by $2n$, and the algorithm terminates by (4.8) or (4.9) in each branch. □

**Algorithm 4.1:** Solving the simplified punching problem with density constraint using dynamic programming.

```
1:  Initialize $L_n^S$, $M_n^S$, $R_n^S$ to **null**
            for all $n \in \{1, \dots, N\}$ and all $S \subseteq \{x_1, \dots, x_N\} \cap [x_n, x_n + a + b]$.
    function A$(n, S)$
3:      if $n = 0 \wedge S = \emptyset$ then
            return 0                                                  ▷ eq. (4.8)
5:      else
            return min{L$(n,S)$, M$(n,S)$, R$(n,S)$}                  ▷ eq. (4.11)
7:  function L$(n, S)$
        if $\max S > x_n + a + b$ then
9:          return $\infty$                                           ▷ eq. (4.10)
        if $L_n^S$ = null then
11:         if $n = 0$ then
                $L_n^S \leftarrow \infty$                             ▷ eq. (4.9)
13:         else
                $L_n^S \leftarrow \min_{S' \subseteq S \cap I_{n-1}} 1 + \mathrm{A}(n-1, S' \setminus X(x_n + a)) + F(S \setminus (S' \cup X(x_n + a)))$
                                                                      ▷ eq. (4.12)
15:     return $L_n^S$
    Use eqs. (4.14) and (4.15) to define M$(n,S)$ and R$(n,S)$ analogously.
17: return A$(N, \emptyset)$
```

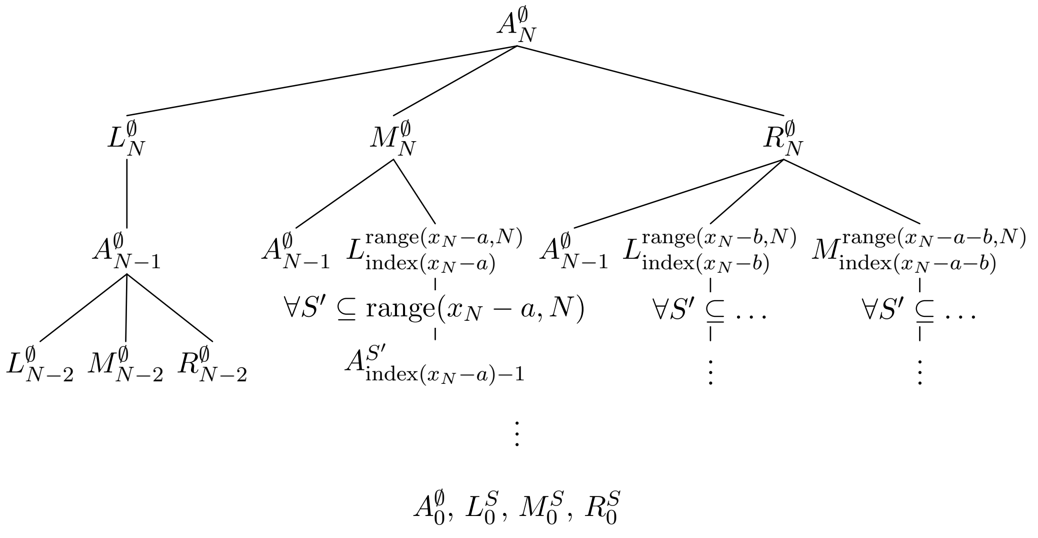

$A_N^\emptyset$
$L_N^\emptyset$
$M_N^\emptyset$
$R_N^\emptyset$
$A_{N-1}^\emptyset$
$A_{N-1}^\emptyset$
$L_{\mathrm{index}(x_N-a)}^{\mathrm{range}(x_N-a,N)}$
$A_{N-1}^\emptyset$
$L_{\mathrm{index}(x_N-b)}^{\mathrm{range}(x_N-b,N)}$
$M_{\mathrm{index}(x_N-a-b)}^{\mathrm{range}(x_N-a-b,N)}$
$\forall S' \subseteq \mathrm{range}(x_N - a, N)$
$\forall S' \subseteq \dots$
$\forall S' \subseteq \dots$
$L_{N-2}^\emptyset$ $M_{N-2}^\emptyset$ $R_{N-2}^\emptyset$
$A_{\mathrm{index}(x_N-a)-1}^{S'}$
$A_0^\emptyset$, $L_0^S$, $M_0^S$, $R_0^S$

**Figure 4.2:** Recursion tree of the dynamic program. The algorithm starts with calculating $A_N^\emptyset$ by recursive calls to the computation of $L_N^\emptyset$, $M_N^\emptyset$, and $R_N^\emptyset$. It continues down to its leaves containing values for $A_0^\emptyset$, $L_0^S$, $M_0^S$, and $R_0^S$.

**Proposition 4.15.** Algorithm 4.1 performs at most $N2^C$ writes to $L_n^S$, $M_n^S$, $R_n^S$, each (such as in lines 12 and 14).

*Proof.* No computed value is ever overwritten due to the initialization to **null** (line 1) and writes only happening when a value is **null** (line 10). Thus, the number of writes is bound by the number of distinct values of $n$ and $S$, which is at most $N2^C$, as $S$ is a subset of a set with a maximum of $C$ elements. ☐

**Theorem 2.** SPP$_{J,C}$ *can be solved in* $\mathcal{O}(N12^C C \log C)$ *time and* $\mathcal{O}(N2^C)$ *space.*

*Proof.* When a function in Algorithm 4.1 is called and no write to $L_n^S$, $M_n^S$, or $R_n^S$ occurs, only basic comparisons and calculations are performed, so the function is computed in constant time, assuming that the sets $S$ are stored in a sorted way so that the maximum can be computed efficiently.

When a write happens, at most $2^C$ subsets $S' \subseteq S$ are iterated through. For each of them, a recursive call happens, which in turn either runs in constant time or is covered by this analysis when a write happens. Furthermore, the function $F(T)$ as given by eq. (4.13) has to be computed, where $|T| \leq |S| \leq C$. In this computation, $3^{|T|} \leq 3^C$ assignments $f : T \to \{1, 2, 3\}$ from holes to punches are iterated through, and in each case a list of $|T| \leq C$ values has to sorted in $C \log C$ time to obtain the size of the corresponding set of machine positions.

Considering that there are at most $N2^C$ writes by Proposition 4.15, the total running time of Algorithm 4.1 is

$$N2^C 2^C 3^C C \log C = N12^C C \log C. \tag{4.16}$$

The space required to store $L_n^S$, $M_n^S$, and $R_n^S$ is obviously of size $\mathcal{O}(N2^C)$. ☐

**Corollary 4.16.** SPP$_{J,C}$ is in $\mathbb{P}$ for fixed $J$ and $C$.

*Remark.* The complexity of the simplified punching problem without density constraint is still an open question as well as the complexity of the original (un-simplified) punching problem.

We implemented the dynamic program in Python (see appendix C) and it performs quite fast indeed. However, there are two major limitations to the relevance of this result for solving practical problem instances.

- The simplified version does not take holes with identical $x$-positions, different $y$-positions, and different hole types into account.
- The worst-case computation time is rather long due to the factor $12^C$ in (4.16).

Therefore, we propose a heuristic approach to solve the punching problem in practice in the following chapters.

CHAPTER 5

# Algorithm for Fixed Configurations

As both the speed and the step punching problem are inefficient to solve exactly in a reasonable amount of time, we propose heuristics targeted towards finding good solutions to practical problem instances as given by the commissioning companies.

The core part of our algorithm computes a punching plan for a given *(fixed machine) configuration*, that is,

- a fixed equipment of the punches with tools and
- fixed relative $x$-positions assigned to all punches, which corresponds to disabling their dynamic $x$-movement by setting $X = 0$.

As outlined in chapter 4, the resulting punching problem is related to the hitting set problem and can be viewed from a graph-theoretic viewpoint. By considerations described in section 4.3, the problem is reasonable to approximate using greedy heuristics.

Therefore, our heuristic approach to the punching problem with fixed machine configuration consists of the following parts.

1. A set of “plausible” fixed machine configurations is determined by
   a) generating all feasible equipments, as described in section 5.1,
   b) determining distances of hole types in the punching plan (section 5.2), and
   c) mapping some of these distances on machine configurations (section 5.3).
2. Given a fixed configuration, a corresponding bipartite graph of holes and machine positions is constructed, as described in section 5.4.
3. Using heuristics described in section 5.5, a solution to the underlying hitting set problem is generated, which corresponds to minimizing the machine stops in the punching plan.

The best resulting punching plan is chosen. An overview is given in Algorithm 5.1.

When dynamic machine configurations are taken into account, additional heuristics

**Algorithm 5.1:** Overview of the algorithm to solve the punching problem.

```
1: Generate a set $\mathcal{E}$ of equipments.
   Generate a set $\mathcal{C}$ of configurations using distances occurring in the pattern.
3: for all configurations $C \in \mathcal{C}$ do
       Determine a corresponding punching plan $S_C$.
5: Choose the plan $S_C$ with minimum costs.
```

can be employed to further improve the results generated by the approach using fixed configurations. These additions are presented in chapter 6.

Throughout this chapter and the next, we will use variables described in Table 5.1.

**Table 5.1:** Common variables in algorithms.

| Variable | Explanation |
|---|---|
| $\overline{z}_n$ | the tool required for hole $n$ (so that $z_{n\overline{z}_n} = 1$) |
| $\mathcal{H}_j$ | set of holes $n$ that can be potentially punched by punch $j$ regarding restrictions on the assigned tool and $y$-position (that is, $T_{j,\overline{z}_n} = 1$ and $\underline{V}_j \le y_n \le \overline{V}_j$) |
| $\mathcal{T}_j$ | set of types $\overline{z}_n$ of holes $n \in \mathcal{H}_j$ that can be punched by punch $j$ |
| $\overline{\mathcal{T}}$ | set of all hole types occurring in the pattern |
| $C = (t_j, s_j)_j$ | a fixed machine configuration consisting of types $t_j$ and positions $s_j$ of the individual punches $j$ |
| $\mathcal{C}$ | a set of configurations $C$ |
| $\mathcal{N}$ | a subset $\mathcal{N} \subseteq \{1, \dots, N\}$ of holes |

**Example 5.1** (Example 3.1 continued)**.** In the given hole pattern shown in Figure 2.2, two hole types occur: circles (type 1) and squares (type 2). The variable $\overline{z}_n$ is set according to the types assigned to individual holes.

| $n \rightarrow$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{z}_n$ | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 |

Holes of both types have $y$-positions in the $[\underline{V}_j, \overline{V}_j]$-range for all punches, which includes the interval $[-500, 500]$ in any case (see the given positions in Example 3.1 and the machine parameter values as given by eqs. (3.31) and (3.32) in section 3.4).

Assuming that there are no additional constraints on the tools that can be assigned to the individual punches by $T_{jt}$, the set of possible types is

$$\mathcal{T}_j = \{1, 2\}$$

for all punches $j$. As each punch can punch every hole when assigned the corresponding tool, the set of potentially punchable holes is

$$\mathcal{H}_j = \{1, \ldots, N\}$$

for all punches $j$.

**Asymptotic analysis** To analyze our proposed algorithm theoretically, we examine its asymptotic runtime and memory requirements. In this analysis, $J$ is the number of punches, $N$ is the number of holes, and $T$ is the number of distinct tools (hole types). Note that

$$T \leq J \leq N.$$

The numbers $T$ and $J$ are treated as constants in our analysis, as the number of punches is fixed ($J = 12$ in practice) and there cannot be more distinct hole types than punches in a feasible pattern.

Furthermore, the maximum number $N_y$ of holes with the same $y$-position is important for the analysis of our algorithm. While $N_y$ could be as high as $N$ in theory, in practice it is very low, usually not being higher than 6.

*Remark.* Elementary operations such as comparisons, additions and multiplications are assumed to run in constant time. To simplify the analysis, primitive set operations (inclusion, deletion, and test for inclusion of a single element) are assumed to run in constant time as well; in theory, they could contribute a logarithmic factor in the worst case, whereas other implementations allow for an amortized constant analysis [Knuth, 1998].

Besides theoretical considerations, a practical evaluation of our algorithm is presented in chapter 7. Some minor algorithms that would only clutter the main part of this chapter (and the following) are specified formally in appendix A.

## 5.1. Determining Equipments

When generating configurations, all *feasible* equipments can be iterated through, as the number of distinct hole types is very limited in practice. We deem an equipment feasible if

- the punches are assigned allowed tools (as restricted by the parameter $T_{jt}$),
- the punches are only assigned types of holes that they can punch with regards to their $y$-positions (limited by the parameters $\underline{V}_j$, $\overline{V}_j$), and
- all hole types are represented in the equipment.

This results in the following procedure to determine feasible equipments (see Algorithm 5.2).

For each punch $j$, we consider the types $\overline{z}_n$ of all holes $n$ that are within the $y$-range of $j$ (that is, $\underline{V}_j \leq y_n \leq \overline{V}_j$) and that can be punched by $j$ ($T_{j\overline{z}_n} = 1$; see line 9). All such possible types $\overline{z}_n$ are stored in a set $\mathcal{T}_j$ (line 10) and all corresponding holes are included in a set $\mathcal{H}_j$ (line 11).

Punches might not be used at all in the process, especially when no holes are within their $y$-range. In this case, an arbitrary allowed type $t$ for punch $j$ (that is, $T_{jt} = 1$) is chosen (line 13).

Equipments are finally formed by picking a possible type for each punch $j$. Therefore, the Cartesian product $\bigotimes_j \mathcal{T}_j$ represents the set of possible equipments. Finally, this set is filtered to ensure that all hole types $\overline{\mathcal{T}}$ are represented in the equipment, that is, only equipments $E$ that include all types in $\overline{\mathcal{T}}$ are considered (line 14). The resulting set of feasible equipments is denoted by $\mathcal{E}$.

**Algorithm 5.2:** Equipments of punches with tools.

```
 1: function EQUIPMENTS
        for $j \leftarrow 1, J$ do
 3:         $\overline{\mathcal{T}} \leftarrow \emptyset$                    ▷ set of all hole types in the given pattern
            $\mathcal{T}_j \leftarrow \emptyset$                    ▷ set of possible types for punch $j$
 5:         $\mathcal{H}_j \leftarrow \emptyset$                    ▷ set of holes potentially punched by punch $j$
            for $n \leftarrow 1, N$ do
 7:             Set $\overline{z}_n$ such that $z_{n\overline{z}_n} = 1$.       ▷ the tool required for hole $n$
                $\overline{\mathcal{T}} \leftarrow \overline{\mathcal{T}} \cup \{\overline{z}_n\}$
 9:             if $T_{j\overline{z}_n} = 1 \wedge \underline{V}_j \le y_n \le \overline{V}_j$ then
                    $\mathcal{T}_j \leftarrow \mathcal{T}_j \cup \{\overline{z}_n\}$       ▷ include type $\overline{z}_n$ in possible types
11:                 $\mathcal{H}_j \leftarrow \mathcal{H}_j \cup \{n\}$       ▷ include hole $n$ in punchable holes
            if $\mathcal{T}_j = \emptyset$ then
13:             $\mathcal{T}_j \leftarrow \{(\text{any } t \text{ with } T_{jt} = 1)\}$       ▷ assign any allowed type
                                                                    when no holes in reach
        return $\left\{E \in \bigotimes_j \mathcal{T}_j \mid \forall t \in \overline{\mathcal{T}} : t \in E\right\}$
                                        ▷ generate all possible combinations of types and
                                filter them so that each hole type is represented at least once
```

**Example 5.2** (Example 5.1 continued)**.** As

$$\mathcal{T}_j = \overline{\mathcal{T}} = \{1, 2\} \quad \text{and} \quad \mathcal{H}_j = \{1, \ldots, N\}$$

for all punches $j$, the resulting set of equipments is the set of tuples that contain at least one 1 and at least one 2, that is,

$$\mathcal{E} = \{(t_j)_j \mid t_j \in \{1, 2\}, \exists j : t_j = 1, \exists j : t_j = 2\}.$$

## 5.2. Distances in the Pattern

Configurations where tools (hole types) occur at distances that are also present in the hole pattern are likely to enable punching of several holes in a single step, which is a crucial objective in both the step and the speed punching problem.

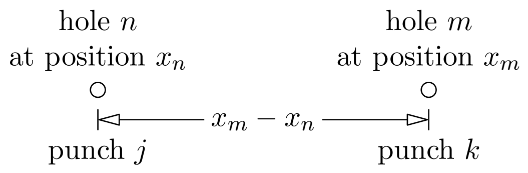


**Figure 5.1:** Distance between two holes $n$, $m$ mapped to a distance between two punches $j$, $k$.

Particularly, given two punches $j$, $k$ and respective hole types $t_j$, $t_k$, we want to find the set of distances $\mathcal{D}_{jk}$ of corresponding holes that can be punched together (see Algorithm 5.3). This can be done by stepping through all pairs of holes $n$, $m$ and taking the following constraints into account (line 5).

- Hole $n$ must have hole type $t_j$ (that is, $\overline{z}_n = t_j$) and must be punchable by punch $j$ with regards to limits on its $y$-position ($n \in \mathcal{H}_j$).
- Hole $m$ must have hole type $t_k$ ($\overline{z}_m = t_k$) and must be punchable by punch $k$ ($m \in \mathcal{H}_k$).
- The $x$-distance between holes $n$ and $m$ must be a feasible $x$-distance for the punches $j$ and $k$ ($\underline{X}_{kj} \leq x_m - x_n \leq \overline{X}_{kj}$).

An illustration of this mapping from the distance between two holes to the distance of two punches is shown in Figure 5.1. Note that we allow the distance $x_m - x_n$ to be negative when punch $k$ is placed to the left of punch $j$.

To ensure that each punch can be inserted somewhere even when there is no corresponding distance in the hole pattern, the minimum and maximum distances $\underline{X}_{jk}$, $\overline{X}_{jk}$ between two punches are always included in the set of relevant distances (line 2).

**Algorithm 5.3:** Computing the set of distances from holes of type $t_j$ to $t_k$ occurring in the hole pattern, punchable by punches $j$ and $k$, respectively.

```
1: function PATTERN-DISTANCES($j, k, t_j, t_k$)
       $\mathcal{D}_{jk} \leftarrow \{\underline{X}_{kj}, \overline{X}_{kj}\}$        ▷ include minimum and maximum distance
                                          between $j$ and $k$ regardless of given pattern
3:     for $n \leftarrow 1, N$ do
           for $m \leftarrow 1, N$ do
5:             if $\overline{z}_n = t_j \wedge \overline{z}_m = t_k \wedge n \in \mathcal{H}_j \wedge m \in \mathcal{H}_k \wedge \underline{X}_{kj} \leq x_m - x_n \leq \overline{X}_{kj}$
               then
                   $\mathcal{D}_{jk} \leftarrow \mathcal{D}_{jk} \cup \{x_m - x_n\}$
7:     return $\mathcal{D}_{jk}$
```

**Example 5.3** (Examples 2.1, 3.1, and 5.2 continued)**.** Considering the punches $j = 3$ and $k = 1$ with types $t_1 = 2$ and $t_3 = 1$, we are interested in the distances between holes punchable by them at the same time. The minimum and maximum distances of the punches are $\underline{X}_{kj} = \underline{X}_{13} = 1100$ and $\overline{X}_{kj} = \overline{X}_{13} = 12550$, respectively, as given by eqs. (3.28) and (3.29).

| | | | |
|---|---|---|---|
| hole 1 | hole 2 | hole 3 | hole 4 |
| at $x_1 = 0$ | at $x_2 = 700$ | at $x_3 = 1500$ | at $x_4 = 2500$ |
| of type $\overline{z}_1 = 3$ | of type $\overline{z}_2 = 3$ | of type $\overline{z}_3 = 2$ | of type $\overline{z}_4 = 1$ |
| ◇ | ◇ | □ | ○ |

**Figure 5.2:** Simple hole pattern.

Therefore, the punches can potentially punch together hole 5 at position $x_5 = 11750$ and hole 3 at $x_3 = 6050$, as

$$\underline{X}_{13} = 1100 \leq x_5 - x_3 = 11750 - 6050 = 5700 \leq 12550 = \overline{X}_{13}.$$

The same holds true for hole 6 and hole 4. This is visible in Figure 2.4, where these holes are punched together in the first step. Furthermore, hole 15 at $x_{15} = 72350$ and hole 11 at $x_{11} = 65950$ can be punched together by the punches 3 and 1, as

$$\underline{X}_{13} = 1100 \leq x_{15} - x_{11} = 72350 - 65950 = 6400 \leq 12550 = \overline{X}_{13}.$$

Again, the same holds for hole 16 and hole 12. There are no other combinations of holes that are within the limits on the distance between punch 1 and punch 3. Therefore,

$$\mathcal{D}_{31} = \{5700, 6400\}$$

in this case.

**Example 5.4.** In the simple hole pattern depicted in Figure 5.2, the following feasible distances occur, assuming punches 1, 2, 3 with types 1, 2, 3 that can punch the corresponding holes with regards to $y$-restrictions:

$$\begin{aligned}
\mathcal{D}_{12} &= \{-1000\} \\
\mathcal{D}_{13} &= \{-2500, -1800\} \\
\mathcal{D}_{21} &= \{1000\} \\
\mathcal{D}_{23} &= \{-1500, -800\} \\
\mathcal{D}_{31} &= \{1800, 2500\} \\
\mathcal{D}_{32} &= \{800, 1500\}.
\end{aligned}$$

## 5.3. Generating Configurations

For each equipment in the set $\mathcal{E}$ of feasible equipments of the machine (see section 5.2), several fixed configurations are constructed. We do this by considering distances between two hole types that occur in the given hole pattern and including these distances in the configuration.

To generate configurations given an equipment $E = (t_j)_j$ of punches $j$ with tools $t_j$, we consider all permutations $\sigma$ of the punches and fix the positions of the punches

**Algorithm 5.4:** Generating fixed configurations by assigning static positions to punches.

```
 1: function FILTER-CONFIGURATIONS($\mathcal{C}$, $n$)
        return set of $\leq n$ random elements of $\mathcal{C}$
 3: function CONFIGURATIONS
        $\mathcal{E} \leftarrow$ EQUIPMENTS                       ▷ get assignments and set $\overline{z}$ and $\mathcal{H}$
 5:     $\mathcal{C} \leftarrow \emptyset$                        ▷ resulting set of configurations
                                                                   (punches with types and positions)
        for all $E = (t_j)_j \in \mathcal{E}$ do
 7:         $\mathcal{C}_E \leftarrow \emptyset$           ▷ overall set of fixed punch positions for assignment
            for all $\sigma \in$ permutations$(1, J)$ do
 9:             $\hat{C} \leftarrow \emptyset$                                ▷ empty configuration
                $\hat{C}_{\sigma_1} \leftarrow (t_{\sigma_1}, 0)$           ▷ assign position 0 to punch $\sigma_1$
11:             $\mathcal{C}_{\sigma,1} \leftarrow \{\hat{C}\}$    ▷ start with configuration where punch $\sigma_1$ is at 0
                for $\tilde{k} \leftarrow 2, J$ do
13:                 $k \leftarrow \sigma_{\tilde{k}}$                                 ▷ current punch
                    $\mathcal{C}_{\sigma,\tilde{k}} \leftarrow \emptyset$          ▷ new positions including current punch
15:                 for all $\tilde{j} \leftarrow 1, \tilde{k} - 1$ do
                        $j \leftarrow \sigma_{\tilde{j}}$                ▷ punch $j$ is the “predecessor” of $k$
17:                     $\mathcal{D}_{jk} \leftarrow$ PATTERN-DISTANCES$(j, k, t_j, t_k)$
                        for all $d \in \mathcal{D}_{jk}$ do
19:                         for all $C \in \mathcal{C}_{\sigma,\tilde{k}-1}$ do
                                $(t_j, s_j) \leftarrow C_j$                  ▷ type and position
                                                                   of the predecessor $j$
                                                          in the existing configuration $C$
21:                             $C' \leftarrow C$
                                $C'_k \leftarrow (t_k, s_j + d)$      ▷ insert punch $k$ at position $s_j + d$
23:                             if IS-CONFIGURATION-FEASIBLE$(C')$ then
                                    $\mathcal{C}_{\sigma,\tilde{k}} \leftarrow \mathcal{C}_{\sigma,\tilde{k}} \cup \{C'\}$
25:                 $\mathcal{C}_{\sigma,\tilde{k}} \leftarrow$ FILTER-CONFIGURATIONS$(\mathcal{C}_{\sigma,\tilde{k}}, L_{\text{step}})$
                $\mathcal{C}_E \leftarrow \mathcal{C}_E \cup$ FILTER-CONFIGURATIONS$(\mathcal{C}_{\sigma,J}, L_{\text{permutation}})$
27:         $\mathcal{C} \leftarrow \mathcal{C} \cup$ FILTER-CONFIGURATIONS$(\mathcal{C}_E, L_{\text{equipment}})$
        $\mathcal{C} \leftarrow$ FILTER-CONFIGURATIONS$(\mathcal{C}, L_{\text{all}})$
29:     return $\{$NORMALIZE-POSITIONS$(C) \mid C \in \mathcal{C}\}$
```

after each other in the order of $\sigma$. Given a new punch $k$, we consider all already fixed punches $j$ as "predecessors." For each of them, we determine all distances from holes of type $t_j$ to holes of type $t_k$ such that these holes can be punched in a single step by the punches $j$ and $k$, respectively. Then we add punch $k$ at each of these distances (see Algorithm 5.4).

Specifically, we construct configurations $\mathcal{C}_E$ for each equipment $E = (t_j)_j \in \mathcal{E}$ as determined by Algorithm 5.2. For each permutation $\sigma$ of the punches, we construct configurations by iteratively adding punches in the given order $\sigma$. That is, we start with an empty configuration $\emptyset$ (no punches fixed yet; line 9) and place the first punch $\sigma_1$ at position 0 (line 10). Let $\hat{C}$ be the resulting configuration.

Let $\mathcal{C}_{\sigma,r}$ denote the set of already constructed configurations for permutation $\sigma$ after adding $r$ punches, where we start with $\mathcal{C}_{\sigma,1} = \{\hat{C}\}$ (line 11). Then, we step through all remaining punches $k$ in the order of the permutation $\sigma$; that is, we consider $k = \sigma_{\tilde{k}}$ for $\tilde{k} \in \{2, \dots, J\}$. We add punch $k$ to all existing configurations in $\mathcal{C}_{\sigma,\tilde{k}-1}$ at distances occurring in the given hole pattern.

To achieve this, we step through all potential predecessors $j = \sigma_{\tilde{j}}$ (for $\tilde{j} \in \{1, \dots, \tilde{k}-1\}$) of punch $k$ and consider the set of distances $\mathcal{D}_{jk}$ from holes punchable by $j$ to holes punchable by $k$ (line 17), as determined by Algorithm 5.3.

For each distance $d$ and each existing configuration $C \in \mathcal{C}_{\sigma,\tilde{k}-1}$, we construct a new configuration $C'$ by adding punch $k$ at distance $d$ from the predecessor $j$, which results in position $s_j + d$ (line 22) assuming $(t_j, s_j) = C_j$ are type and position of punch $j$ in $C$. The new configuration $C'$ is added to $\mathcal{C}_{\sigma,\tilde{k}}$ (line 24) provided that it is *feasible.*

Although punches are added sequentially at feasible distances to their predecessors, constraints on the $x$-distances to other punches might get violated during the course. Thus, we have to check whether a generated (partial) configuration is feasible with regards to the $x$-positions of the punches. This can simply be done by stepping through all pairs of punches $j$, $k$ and testing whether their $x$-distance $s_j - s_k$ is feasible, that is,

$$\underline{X}_{jk} \leq s_j - s_k \leq \overline{X}_{jk} \tag{5.1}$$

as constrained by (3.15) (see Algorithm A.1 in appendix A).

Note that neither the fact that the configuration is partial (not necessarily all punches are part of it yet) nor that it is not "centered" at the reference punch $\bar{j}$ yet impose a problem when checking differences between the positions of the (already fixed) punches.

To limit the number of generated configurations to a reasonable amount, the set of configurations is filtered randomly at several points in the algorithm, reducing it to sizes specified by parameters $L_{\text{step}}$, $L_{\text{permutation}}$, $L_{\text{equipment}}$, and $L_{\text{all}}$.

- By adding a single punch (a step in the permutation $\sigma$), no more than $L_{\text{step}}$ configurations are added (line 25).
- For each permutation $\sigma$, a maximum of $L_{\text{permutation}}$ configurations are kept (line 26).

- For each equipment $(t_j)_j$, a maximum of $L_{\text{equipment}}$ configurations are kept (line 27).
- Altogether, no more than $L_{\text{all}}$ configurations are generated (line 28).

At each of these points, the set of generated configurations is limited to a maximum number of $L_{\ldots}$ random elements (line 2). Possible choices of these parameters and their impact on the results are discussed in section 7.6.

The positions $s_j$ assigned to punches resulting from this procedure are "relative" in the sense that an arbitrary punch $\sigma_1$ is assigned position 0 and all other punches are positioned relative to $\sigma_1$. For the sake of consistency, we can simply relate them to the reference punch $\overline{j}$ by subtracting $s_{\overline{j}}$ from each of them. Specifically, given a configuration $C = (t_j, s_j)_j$, we construct a normalized configuration $C'$ by positioning punch $j$ at

$$s'_j = s_j - s_{\overline{j}} \tag{5.2}$$

(see Algorithm A.2).

**Example 5.5** (Example 5.4 continued)**.** Consider the pattern depicted in Figure 5.2 and an equipment

$$E = (1, 2, 3)$$

of a punching machine with three punches. According to Algorithm 5.4, we step through all permutations of these punches. Consider

$$\sigma = (1, 2, 3),$$

for instance. This permutation is stepped through in the algorithm.

1. Punch $\sigma_1 = 1$ is placed at position 0 in a (partial) configuration $\hat{C} = ((1, 0))$ specified by the type and position of its punches. This is included in the set

$$\mathcal{C}_{\sigma,1} = \{\hat{C}\} = \{((1, 0))\}.$$

2. Then we add the second punch $k = \sigma_{\tilde{k}} = 2$. Punch $j = \sigma_1 = 1$ is chosen as the predecessor of $k = 2$. Given $\mathcal{D}_{12} = \{-1000\}$, punch 2 is added at distance $-1000$ from punch 1 to all existing configurations $\mathcal{C}_{\sigma,\tilde{k}-1} = \mathcal{C}_{\sigma,1}$ (which is only one configuration in this case), so we get the new set of configurations

$$\mathcal{C}_{\sigma,2} = \{((1, 0), (2, -1000))\}.$$

3. Finally, we add punch $k = 3$ with predecessors 1 or 2. The relevant distances are $D_{13} = \{-2000, -2500\}$ and $D_{23} = \{-1000, -1500\}$. Starting with $\mathcal{C}_{\sigma,3} = \emptyset$, we step through the predecessors $j$, the occurring distances $\mathcal{D}_{jk}$, and the existing configurations $\mathcal{C}_{\sigma,2}$. For the single configuration $C = ((1, 0), (2, -1000))$ in there, we add punch 3 at the corresponding distance to $j$, resulting in

$$\mathcal{C}_{\sigma,3} = \{((1, 0), (2, -1000), (3, -1800)), ((1, 0), (2, -1000), (3, -2500))\}.$$

The set $\mathcal{C}_{\sigma,J} = \mathcal{C}_{\sigma,3}$ is added to $\mathcal{C}_E$, which is eventually added to the overall set $\mathcal{C}$ of configurations.

To normalize positions, let us assume the reference punch is $\overline{j} = 2$ in this case. Then $\mathcal{C}$ is be modified to

$$\{((1,1000),(2,0),(3,-800)),((1,1000),(2,0),(3,-1500))\}$$

through normalization as given by eq. (5.2).

Note that these two configurations are suited perfectly for punching the pattern depicted in Figure 5.2.

An asymptotic analysis of the proposed algorithm to generate fixed machine configurations yields the following result.

**Proposition 5.6.** The set of plausible configurations resulting from Algorithm 5.4 can be generated in $\mathcal{O}(T^J J^2 J! N^2)$ time using $\mathcal{O}(T^2 J^2 N^2)$ space.

*Proof.* The set of $\mathcal{O}(T^J)$ equipments can be determined in $\mathcal{O}(JNT + T^J)$ time (Algorithm 5.2).

For each equipment, $J!$ permutations of the punches are considered, with $J^2$ loops through pairs of punches (Algorithm 5.4). The pattern distances can be cached using $\mathcal{O}(J^2 T^2 N^2)$ time and memory (Algorithm 5.3). For each of the $\mathcal{O}(N^2)$ distances $\mathcal{D}$, at most $L_{\text{step}}$ configurations are tested for feasibility, which can be done in $\mathcal{O}(J^2)$ time. The normalization of at most $L_{\text{all}}$ configurations is finally done in $\mathcal{O}(J)$ time.

Altogether, we get a time requirement of

$$\mathcal{O}(JNT + J^2T^2N^2 + T^J J! J^2 N^2 L_{\text{step}} J^2 + L_{\text{all}} J) = \mathcal{O}(T^J J^2 J! N^2),$$

given that $L_{\text{step}}$ and $L_{\text{all}}$ are constant and assuming $J \geq 2$.

The space required is $\mathcal{O}(T^2 J^2 N^2)$ for the pattern distances. Equipments do not have to be stored but can be enumerated "on the fly." Each configuration needs $\mathcal{O}(J)$ space, so at most

$$\mathcal{O}(J \max\{L_{\text{step}}, L_{\text{permutation}}, L_{\text{equipment}}, L_{\text{all}}\}) = \mathcal{O}(J)$$

space is needed to store all configurations. □

*Remark.* In practice, the $N^2$ factor does not really emerge, because there are far less relevant distances occurring between two hole types in the pattern, especially as only distances up to certain limits $\underline{X}_{jk}$, $\overline{X}_{jk}$ are considered. For all practical purposes, the algorithm performs as if it were linear in $N$.

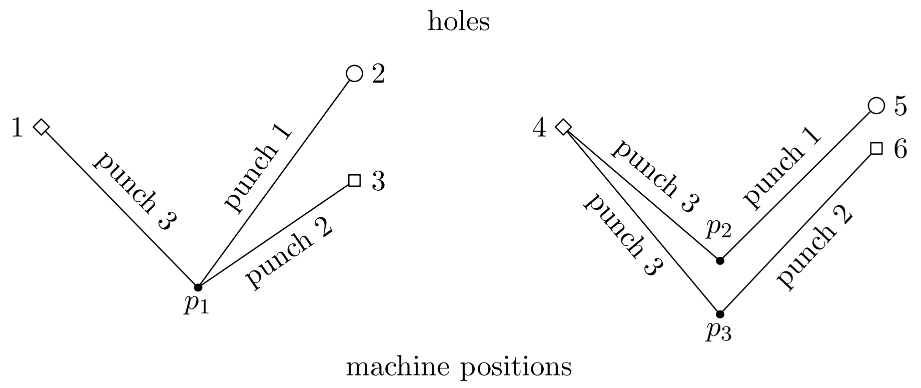


**Figure 5.3:** Machine positions resulting from a fixed machine configuration and a hole pattern with three different hole types circle (type 1), square (type 2), and rhombus (type 3). Holes 1, 2, 3 can be punched together in a single step $p_1$, while holes 5 and 6 are too close together so they have to be punched in separate steps $p_2$ and $p_3$ with the same $x$-position of the machine.

## 5.4. Constructing the Hole-Position Graph

Given a fixed configuration $C$, we try to find a punching plan that makes few machine stops. Minimizing the number of machine stops is obviously crucial to both the step punching problem (Definition 3.4) and the speed punching problem (Definition 3.5). We do this by constructing a bipartite graph of holes and machine positions as already introduced in chapter 4 and propose heuristics to solve the hitting set problem therein.

As the relative positions $\overline{s}_{ij} = \overline{s}_j$ of all punches (relative to the reference punch $\overline{j}$) are fixed through all steps $i$, there is a unique absolute machine position from which each hole $n$ can be punched by a given punch $j$ (if at all), namely $x_n - \overline{s}_j$.

However, there might be machine positions from which several holes can be punched, but not in a single step; for example, when there are two holes with the same $x$-positions and types. In that case, we speak of distinct machine positions (although their $x$-positions are identical) with different sets of punchable holes (see Figure 5.3).

Let the set of all resulting machine positions be $\mathcal{P}$. This set $\mathcal{P}$ joined with the set of holes $\mathcal{N} = \{1, \dots, N\}$ can be interpreted as the vertices of a bipartite graph $G = (\mathcal{P} \uplus \mathcal{N}, E)$, where each vertex $p \in \mathcal{P}$ is connected to another vertex $n \in \mathcal{N}$ if $n$ can be punched from $p$.

This graph $G$ is similar to the bipartite graph introduced in chapter 4 with the only differences being that in the underlying punching problem

- holes might have the same $x$-positions and
- there might be several different hole types.

As there might be several machine positions at the same $x$-position, we cannot use it to identify machine positions. Therefore, we use *objects* of type POSITION representing machine positions containing the following *attributes*.

- $p \in \mathbb{R}$: their $x$-position,
- $\mathit{punches} \subseteq \{1, \dots, J\}$: the set of active punches at this machine position,
- $n : \mathit{punches} \to \{1, \dots, N\}$: a mapping from punches to punched holes, and
- $\mathit{pos} : \{1, \dots, J\} \to \mathbb{R}$: the relative positions of the punches.

The attributes *pos* is not necessary to identify a machine position and is simply set to the static relative position of a punch in this phase of the algorithm. However, it is used at a later point when dynamic machine configurations come into play.

**Definition 5.7** (Object/attribute notation)**.** To simplify notation, we write $x.y$ to denote the *attribute* named $y$ of an *object* $x$. Objects are considered equal if and only if all their attributes are equal.

In the case of Position objects $p$, $p.p$ denotes their $x$-position, $p.\mathit{punches}$ the set of active punches at $p$, $p.n$ the mapping to punched holes, and $p.\mathit{pos}$ the relative positions of the punches. Furthermore, we write $\textsc{Position}(p, n)$ to denote the creation of an object of type Position with attributes $p$ and $n$ set to the specified values.

Note that, as $p.n$ is a function from active punches to their respective punched holes at position $p$, the image $\operatorname{Img} p.n$ of this function is the set of all punched holes in this step.

**Example 5.8.** Consider the hole pattern depicted in Figure 5.3 and three punches 1, 2, 3 that can punch hole types 1, 2, 3, respectively. Assuming that holes 2 and 3 are far enough from each other that they can be punched together by punches 2 and 3, that is,

$$\underline{Y}_{23} \leq y_2 - y_3 \leq \overline{Y}_{23}$$

as demanded by (3.16), there is a single machine position $p_1$ from which the holes 1, 2, 3 can be punched. As depicted in Figure 5.3, the attribute $p_1.n$ mapping punches to punched holes at this machine position is set to

$$p_1.n(1) = 2, \quad p_1.n(2) = 3, \quad p_1.n(3) = 1.$$

The set of active punches is

$$p_1.\mathit{punches} = \{1, 2, 3\}$$

and the set of punched holes is given by

$$\operatorname{Img} p_1.n = \{1, 2, 3\}.$$

Contrarily, holes 5 and 6 are too close to each other so that they have to be punched in separate punching steps with machine positions $p_2$ and $p_3$. The attribute $p_2.n$ is set to

$$p_2.n(1) = 5, \quad p_2.n(3) = 4,$$

while $p_3.n$ is set to

$$p_3.n(2) = 6, \quad p_3.n(3) = 4.$$

To determine the set of machine positions $\mathcal{P}$, we start with an empty set and add machine positions as we iterate through all holes $n$ and punches $j$ (see Algorithm 5.5). For each combination, we test if punch $j$ can punch hole $n$ (see line 5), that is,

- punch $j$ has the correct tool to punch hole $n$ ($z_{nt_j} = 1$) and
- hole $n$ is on a feasible $y$-position for punch $j$ ($\underline{V}_j \leq y_n \leq \overline{V}_j$).

The resulting machine position $p = s_{i\bar{j}}$ (the absolute position of the reference punch $\bar{j}$ in step $i$) from which $j$ can punch $n$ is given by $p = x_n - \overline{s}_j$ (line 6) because of

$$x_n = s_{ij} = s_{i\bar{j}} + \overline{s}_{ij} = s_{i\bar{j}} + \overline{s}_j$$

by eq. (3.1). We consider the set $P$ of existing machine positions in $\mathcal{P}$ at $x$-position $p$. If there are none (line 18), we simply add a new machine position where punch $j$ punches hole $n$.

Otherwise, we have to take restrictions on the $y$-positions of the existing punches and the new punch $j$ into account. Iterating through all existing machine positions $q \in P$, we consider the set $K$ of all punches $k$ active in $q$ that are "$y$-compatible" with the new punch $j$ punching hole $n$, that is,

$$\underline{Y}_{jk} \leq y_n - y_{q.n(k)} \leq \overline{Y}_{jk}$$

(line 9). If all the existing punches are compatible to the new punch $j$, that is, $K = q.punches$, we can simply add $j$ to $q$, provided that $j$ is not active in $q$ yet (line 10). Otherwise, we construct a new machine position at $p$ with all the compatible punches $K$ punching their respective holes and the new punch $j$ being added (line 13).

Once the machine positions $\mathcal{P}$ are determined, the graph containing the punched holes as well is simply given by

$$G = (\mathcal{P} \cup \{1, \dots, N\}, E)$$

with edges

$$E = \{\{p, n\} \mid p \in \mathcal{P}, n \in \operatorname{Img} p.n\}$$

between machine positions $p$ and holes $n$ if and only if $n$ can be punched from $p$.

## 5.5. Punching Plan by Hitting Set Heuristics

As examined in chapter 4, the problem of finding a punching plan with a minimum number of punching steps is related to the hitting set problem in the resulting graph of holes and machine positions. As stated in section 4.3, greedy algorithms perform well on the hitting set problem, which is why we propose such an approach to the punching problem with fixed machine configuration as well.

The idea is to start with an empty punching plan, to step through the holes in a given order $\sigma$, and to choose a machine position $p_n$ to punch each hole $n$ from such that the resulting costs after adding $p_n$ to the punching plan are minimal. In case of

**Algorithm 5.5:** Possible machine positions for a given fixed machine configuration.

```
1: function CONSTRUCT-POSITIONS($C = (t_j, \overline{s}_j)_j$)
       $\mathcal{P} \leftarrow \emptyset$
3:     for $n \leftarrow 1, N$ do
           for $j \leftarrow 1, J$ do
5:             if $z_{nt_j} = 1 \wedge \underline{V}_j \leq y_n \leq \overline{V}_j$ then
                   $p \leftarrow x_n - \overline{s}_j$
7:                 $P \leftarrow \{q \in \mathcal{P} \mid q.p = p\}$
                   for all $q \in P$ do
9:                     $K \leftarrow \{k \in q.punches \mid \underline{Y}_{jk} \leq y_n - y_{q.n(k)} \leq \overline{Y}_{jk}\}$
                                                  ▷ set of punches that are $y$-compatible
                                                       with the new punch $j$ punching $n$
                       if $j \notin q.punches \wedge K = q.punches$ then
                                                  ▷ all already-assigned punches are
                                                       $y$-compatible with the new punch
11:                        $q.n(j) \leftarrow n$
                           $q.punches \leftarrow q.punches \cup \{j\}$
13:                    else
                           $\overline{q} \leftarrow$ POSITION($p$, $q.n|_K$)      ▷ restrict active punches
                                                                to possible punches
15:                        $\overline{q}.n(j) \leftarrow n$
                           $\overline{q}.punches \leftarrow K \cup \{j\}$
17:                        $\mathcal{P} \leftarrow \mathcal{P} \cup \{\overline{q}\}$
                   if $P = \emptyset$ then                ▷ create a completely new position
                           when there are no existing punching steps at this position
19:                    $\overline{q} \leftarrow$ POSITION($p, j \mapsto n$)
                       $\overline{q}.punches \leftarrow \{j\}$
21:                    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\overline{q}\}$
       for all $p \in \mathcal{P}$ do
23:        p.$pos \leftarrow (\overline{s}_j)_j$     ▷ set the static machine positions (to be changed later)
       return $\mathcal{P}$
```

a tie, that is, several machine positions result in the same costs, we choose a machine position that allows a maximum number of other holes being punched as well. The latter criterion is inspired by the hitting set algorithm proposed by CHVATAL [1979].

As shown in Algorithm 5.6, vertices corresponding to holes are removed from the graph as they are punched, which is why we operate on a copy $G'$ of $G$ (line 3). By removing already-punched holes from the graph, the degree $\deg_{G'}(p)$ of a machine position vertex $p$ is reduced and equals the number of unpunched holes that can still be punched from $p$ (line 8).

**Algorithm 5.6:** Hitting set for a given position-hole graph.

```
1:  function GREEDY-PLAN($G, \sigma$)
        $S \leftarrow \emptyset$                                  ▷ set of machine stops
3:      $G' \leftarrow G$                                        ▷ remaining graph
        for $\tilde{n} \leftarrow 1, N$ do
5:          $n \leftarrow \sigma_{\tilde{n}}$
            for all $p \in \text{Neighbors}_{G'}(n)$ do
7:              $c_p \leftarrow$ PLAN-COSTS($S \cup \{p\}$)
                $d_p \leftarrow \deg_{G'}(p)$          ▷ number of remaining (non-assigned) holes
                                                         that can be punched from position $p$
9:          $\tilde{p} \leftarrow \arg\min_p(c_p, -d_p)$      ▷ choose position with minimum costs
                                                        and (in case of a tie) maximum number
                                                                 of other non-assigned holes
            $S \leftarrow S \cup \{\tilde{p}\}$
11:         for all $m \in \text{Neighbors}_{G'}(\tilde{p})$ do                ▷ punch all holes
                                              that can be punched from $\tilde{p}$ (including hole $n$)
                Remove vertex $m$ from $G'$.
13:     return $S$
```

The costs of a punching plan are determined by summing up the costs arising between any two consecutive punching steps. Therefore, the machine positions have to be sorted in ascending order with regards to their $x$-position $p$ (see Algorithm A.3.

Any reasonable cost function $c(d)$ that assigns the costs to an individual movement $d$ of the machine can be chosen (see section 3.3). A choice of $c(d) = 1$ would simply result in the minimization of the number of punching steps.

After constructing the graph as described in section 5.4, several special orderings $\sigma$ are taken into account:

1. $\hat{\sigma}$, the $x$-ordered set of holes, and
2. $\tilde{\sigma}$, the holes ordered by their degree in the graph $G$, that is, by the number of machine positions from which they can be punched.

Each ordering is used in both directions, resulting in a set $\Sigma$ of four orderings. For every ordering $\sigma$, a punching plan $S_\sigma$ is determined using the greedy approach, and the plan with the minimum costs is chosen (see Algorithm A.4).

**Proposition 5.9.** A punching plan for a given static configuration can be determined with $\mathcal{O}(JN_y^JN)$ time and space.

*Proof.* Given a static machine configuration $C$ at a fixed machine position $p$, there are not more than $N_y^J$ possible mappings from the punches to holes. This is the maximum size of the set $P$ in Algorithm 5.5. Therefore, the machine positions can be generated in $\mathcal{O}(NJN_y^J)$ time.

A hole $n$ can be punched by any of the $J$ punches, inducing $J$ different machine positions with the above limit of $N_y^J$ possible mappings from punches to holes as above. Therefore, $|\text{Neighbors}_G(n)| \leq \mathcal{O}(JN_y^J)$ holds. Furthermore, $|\text{Neighbors}_G(p)| \leq J$ holds for a position $p$, so Algorithm 5.6 can be implemented to run in $\mathcal{O}(NJN_y^J)$ time if the costs for a punching plan are updated incrementally as more holes are punched.

The graph $G$ needs $\mathcal{O}(NJN_y^J)$ space and the resulting punching plans $S_\sigma$ require $\mathcal{O}(NJ)$ space each. □

## 5.6. Overall Algorithm

The algorithms in sections 5.1 to 5.5 can be combined to yield the overall algorithm solving the punching problem with fixed machine configuration, as already outlined in Algorithm 5.1.

1. A set $\mathcal{C}$ of plausible configurations is generated.
2. For each configuration $C \in \mathcal{C}$, a punching plan $S_C$ is determined using the fixed configuration $C$.
3. A punching plan $S_{C^*}$ with minimum costs is chosen.

Algorithm 5.7, a more formal version of Algorithm 5.1, also shows the final relation of the punching plan defined by the set of POSITIONs and the actual decision variables in the punching problem.

- The equipment $u_{jt}$ follows from the types $t_j$ in the chosen configuration $C = (t_j, \overline{s}_j)_j$ (line 8).
- The relative punch positions $\overline{s}_{ij}$ in each punching step $i$ are stored in the *pos* attribute of POSITION objects (line 12).
- The absolute punch positions $s_{ij}$ can be calculated from $\overline{s}_{ij}$ and the machine positions $p$ (line 12).
- The punched holes $h_{ijn}$ follow from the images $\operatorname{Img} p.n$ of the functions $p.n$ that map active punches to their punched holes, as well as the depending variables $a_{ij}$ (line 16).
- The $y$-positions are forced for active punches as they have to match the $y$-positions $y_n$ of their punched holes (line 15). For an inactive punch $j$, we choose an arbitrary position with the following constraints (line 18).
  - The $y$-position has to be within the limits $[\underline{V}_j, \overline{V}_j]$.

**Algorithm 5.7:** Solving the punching problem with fixed machine configuration.

```
1:  function PUNCHING-PROBLEM
        $\mathcal{C} \leftarrow$ CONFIGURATIONS
3:      for all $C = (t_j, \overline{s}_j)_j \in \mathcal{C}$ do
            $S_C \leftarrow$ PUNCHING-PLAN($C$)            ▷ determine a punching plan for fixed configuration $C$
5:      $C^* \leftarrow \arg\min_{c \in \mathcal{C}}$ PLAN-COSTS($S_C$), $S^* \leftarrow S_{C^*}$
        Initialize $u_{jt}$, $h_{ijn}$, $a_{ij}$ to 0.
7:      for $j \leftarrow 1, J$ do
            $u_{jt_j} \leftarrow 1$
9:      $i \leftarrow 1$                                   ▷ number of punching step
        for all $p \in S^*$ do
11:         for $j \leftarrow 1, J$ do
                $\overline{s}_{ij} \leftarrow p.pos(j)$, $s_{ij} \leftarrow \overline{s}_{ij} + p.p$
13:             if $j \in \operatorname{Img} p.n$ then      ▷ punch $j$ is active in this step
                    $n \leftarrow p.n(j)$                  ▷ hole punched by punch $j$
15:                 $v_{ij} \leftarrow y_n$                ▷ $y$-position of the punched hole
                    $h_{ijn} \leftarrow 1$, $a_{ij} \leftarrow 1$
17:             else
                    $Y \leftarrow [\underline{V}_j, \overline{V}_j] \cap \bigcap_{j' \in \operatorname{Img} p.n} [\underline{Y}_{jj'} + y_{p.n(j')}, \overline{Y}_{jj'} + y_{p.n(j')}]$
19:                 $v_{ij} \leftarrow$ (arbitrary $y$-position $\in Y$)
            $i \leftarrow i + 1$
```

– The $y$-distances to all active punches $j' \in \operatorname{Img} p.n$ have to be feasible. The minimum distance between $j$ and $j'$ is given by $\underline{Y}_{jj'}$, the maximum is $\overline{Y}_{jj'}$, and punch $j'$ is positioned at the $y$-position $y'_{p.n(j')}$ of its punched hole. This yields the range

$$\bigcap_{j' \in \operatorname{Img} p.n} [\underline{Y}_{jj'} + y_{p.n(j')}, \overline{Y}_{jj'} + y_{p.n(j')}]$$

for punch $j$.

Summing up the asymptotic analysis of the proposed algorithms, we conclude with the following overall time and space requirements.

**Theorem 3.** *The punching problem can be solved in $\mathcal{O}(T^J J^2 J! N^2 + J N_y^J N)$ time using $\mathcal{O}(T^2 J^2 N^2 + J N_y^J N)$ space.*

*Proof.* Combining the results from Proposition 5.6 and 5.9 (as in Algorithm 5.7) yields

$$\mathcal{O}(T^J J^2 J! N^2 + L_{\text{all}}(J N_y^J N + J^4 N))$$

time and

$$\mathcal{O}(T^2 J^2 N^2 + L_{\text{all}} J N_y^J N)$$

space. The result follows from treating $L_{\text{all}}$ as constant. □

## 5.7. Implementation Details

For simplicity, some aspects in the implementation of the algorithm are not described formally in this chapter, but only mentioned briefly here.

- As pointed out in the proof of Proposition 5.9, the plan costs should not be calculated globally from scratch on each invocation of Plan-Costs, but can rather be updated locally with each added, changed, or removed punching step.
- Because holes typically appear in pairs with equal $x$-coordinates and types, a preprocessing step can group such holes together and introduce artificial combined hole types that are assigned to pairs of punches. Thereby, the number of holes as well as the number of punches that has to be dealt with is reduced by half. Of course, this only works when there are not too many asymmetrical hole types that require the punches of each stamp to be dealt with separately.
- To speed up the process of finding the best equipment of the machine with tools (see Algorithm 5.7), the hole pattern can be reduced to a few repetitions of each block of holes. After choosing a configuration $C^*$, the full pattern can be optimized.

CHAPTER 6

# Heuristics for Dynamic Configurations

After solving the punching problem for a fixed machine configuration, dynamic $x$-movements of the punches can be taken into account as well. We propose two heuristics to improve punching plans.

1. Individual punching steps are combined when possible, as described in section 6.2.
2. Machine stops are moved so that they are distributed more equally, as described in section 6.3. Similar heuristics that can be used to reduce the resulting waste are outlined briefly in section 6.4.

Some general methods needed for both heuristics are introduced in section 6.1. The heuristics are connected by repeatedly applying them after each other, as outlined in section 6.5.

## 6.1. Assignments and Positions of Punches

When improving a punching plan, we essentially reassign holes to other punches or punching steps and adjust punch position. Naturally, a central part of this is to determine possible assignments of punches to a given subset $\mathcal{N} \subseteq \{1, \dots, N\}$ of holes such that the hole types are matched by the given tools of the punches. Assuming that the implied positions of the punches are feasible as well, this allows all holes in $\mathcal{N}$ to be punched in a single punching step.

An *assignment* can be defined as an injective function

$$A : \mathcal{N} \to \{1, \dots, J\}.$$

Then the image of the function, $\operatorname{Img} A$, is the set of active punches at this step, and the pre-image $A^{-1}(j)$ of an active punch $j$ is the hole $n$ that is punched by punch $j$.

## Assignments of punches to holes

Assignments can be determined by incrementally building up a set $\mathcal{A}$ while stepping through the holes in a given subset $\mathcal{N} \subseteq \{1, \dots, N\}$ (see Algorithm 6.1). Starting with the set of an empty assignment $\mathcal{A} = \{\emptyset\}$ (line 2), we construct new sets $\mathcal{A}_n$ for each hole $n \in \mathcal{N}$ depending on the previous assignments $\mathcal{A}$. Given a hole $n$ and an existing assignment $A \in \mathcal{A}$, we check for each punch $j$ the following conditions (line 7).

- Punch $j$ is not used in $A$ yet ($j \notin \operatorname{Img} A$).
- Punch $j$ can punch hole $n$ according to its type ($\overline{z}_n = t_j$) and according to its $y$-position ($n \in \mathcal{H}_j$).

If these conditions are met, we can add $j$ punching hole $n$ to the assignment $A$ (line 8) and include it in the resulting set $\mathcal{A}_n$ (line 9), which eventually replaces $\mathcal{A}$. Finally, only feasible configurations shall be generated, which is why we filter the assignments where each punch $j$ is positioned at the $x$-position of the assigned hole $A^{-1}(j)$ using Algorithm A.1.

**Algorithm 6.1:** Assignments of punches to holes of given types.

```
1:  function HOLE-ASSIGNMENTS($\mathcal{N}, (t_j)_j$)
        $\mathcal{A} \leftarrow \{\emptyset\}$
3:      for all $n \in \mathcal{N}$ do
            $\mathcal{A}_n \leftarrow \emptyset$                              ▷ empty assignment (function)
5:          for all $A \in \mathcal{A}$ do
                for $j \leftarrow 1, J$ do
7:                  if $j \notin \operatorname{Img} A \wedge \overline{z}_n = t_j \wedge n \in \mathcal{H}_j$ then    ▷ punch $j$ is not used yet
                                                                        and can punch hole $n$
                        $A(n) \leftarrow j$          ▷ punch $j$ punching hole $n$ in existing assignment
9:                      $\mathcal{A}_n \leftarrow \mathcal{A}_n \cup \{A\}$
            $\mathcal{A} \leftarrow \mathcal{A}_n$
11:     return $\{A \in \mathcal{A} \mid \text{IS-CONFIGURATION-FEASIBLE}((t_j, x_{A^{-1}(j)})_{j \in \operatorname{Img} A})$
```

**Example 6.1.** Consider a set $\mathcal{N} = \{1, 2\}$ of two holes with types $\overline{z}_1 = 2$, $\overline{z}_2 = 1$ and an equipment $(t_j)_j = (1, 1, 2)$ of three punches, so that punches 1 and 2 can punch hole 2, while punch 3 can punch hole 1. We want to find feasible assignments $A$ of punches to holes such that the holes in $A$ can be punched in a single punching step.

Enumerating the holes $n \in \mathcal{N}$ as in Algorithm 6.1, we construct the following sets of assignments.

1. For $n = 1$, we start by setting $\mathcal{A}_1$ to $\emptyset$. Considering the existing assignment $A = \emptyset \in \mathcal{A}$ (an empty function), we test which punches $j \notin \operatorname{Img} A$ can punch hole 1, which is punch 3 in this case. Therefore, we add the function value $A(1) = 3$ to $A$ and include $A$ in $\mathcal{A}_1$, resulting in

$$\mathcal{A}_1 = \{A\} = \{1 \mapsto 3\}$$

which is then assigned to $\mathcal{A}$.

2. For $n = 2$, we consider the existing assignment $A = (1 \mapsto 2) \in \mathcal{A}$. Now, $j = 1$ and $j = 2$ can punch hole 2. In each case, we add $j$ punching hole 2 to $A$, resulting in
$$\mathcal{A} = \mathcal{A}_2 = \{(1 \mapsto 3, 2 \mapsto 1), (1 \mapsto 3, 2 \mapsto 2)\}.$$

## Relative positions of punches

While an assignment answers the question, which hole is punched by which punch, the question of where the punching machine can be positioned to do that still remains.

Specifically, we are interested in a machine position $s$ and relative punch positions $(\overline{s}_j)_j$ given

- an assignment $A$ of punches to a subset of holes;
- relative positions $(\overline{s}'_j)_{j\in P}$ in the previous punching step (before the holes in $A$ should be punched), where $P = \emptyset$ if there is no previous step;
- relative positions $(\overline{s}''_j)_{j\in N}$ in the following punching step;
- an interval $[a, b]$ of possible machine positions (restricted by the previous and following punching step, for instance); and
- a “preferred” machine position $\alpha$.

The machine position $s$ is chosen in $[a, b]$ as close to $\alpha$ as possible such that the following constraints are met.

- The absolute positions of active punches are fixed and restrict the absolute positions of all other punches through the minimum and maximum distances between punches (parameters $\underline{X}_{jk}$, $\overline{X}_{jk}$).
- The relative positions of all punches are restricted by the maximum $x$-movement after the previous punching step and before the next punching step (parameter $X$).

To compute the new relative punch positions $\overline{s}_j$ (see Algorithm 6.2), consider the set $A' = \operatorname{Img} A$ of active punches in the assignment $A$, and let $s_a = x_{A^{-1}(a)}$ be the absolute $x$-position of each active punch $a \in A'$ (line 4). Then the interval $R_j$ of possible absolute positions of each punch $j$ is restricted by $s_a$ and the constraints on the $x$-distances between punch $j$ and active punch $a$, that is,

$$R_j \subseteq \bigcap_{a\in A'} [\underline{X}_{ja} + s_a, \overline{X}_{ja} + s_a] \tag{6.1}$$

holds (line 7), which follows from

$$\underline{X}_{ja} \leq s_j - s_a \leq \overline{X}_{ja} \qquad \forall s_j \in R_j$$

as given by eq. (3.15). Furthermore, the interval $\overline{R}_j$ of possible relative positions of each punch $j$ is restricted by

$$\overline{R}_j \subseteq [\underline{X}_{j\bar{j}}, \overline{X}_{j\bar{j}}] \tag{6.2}$$

**Algorithm 6.2:** Determining positions of punches punching a set of holes in an assignment $A$ to the punches, given previous relative positions $(\overline{s}'_j)_j$ and following positions $(\overline{s}''_j)_j$ of the punches, a possible interval $[a, b]$ of machine positions and a preferred machine position $\alpha$.

```
function PUNCH-POSITIONS($A, (\overline{s}'_j)_{j \in P}, (\overline{s}''_j)_{j \in N}, [a, b], \alpha$)
    $A' \leftarrow \operatorname{Img} A$                              ▷ set of active punches
    for all $a \in A'$ do
        $s_a \leftarrow x_{A^{-1}(a)}$                  ▷ absolute $x$-position of active punch $a$
    $R \leftarrow [a, b]$                  ▷ possible range of absolute machine positions
    for $j \leftarrow 1, J$ do
        $R_j \leftarrow \bigcap_{a \in A'} [\underline{X}_{ja} + s_a, \overline{X}_{ja} + s_a]$            ▷ possible range of
                            absolute positions of punch $j$, taking into account
                     minimum and maximum $x$-distances to the active punches $a \in A'$
        $\overline{R}_j \leftarrow [\underline{X}_{j\overline{j}}, \overline{X}_{j\overline{j}}]$                      ▷ possible range of relative
                                positions of punch $j$ to the reference punch $\overline{j}$
        if $j \in P$ then
            $\overline{R}_j \leftarrow \overline{R}_j \cap [\overline{s}'_j - X, \overline{s}'_j + X]$              ▷ take maximum
                      relative $x$-movement after previous punching step into account
        if $j \in N$ then
            $\overline{R}_j \leftarrow \overline{R}_j \cap [\overline{s}''_j - X, \overline{s}''_j + X]$             ▷ take maximum
                        relative $x$-movement before next punching step into account
        $R \leftarrow R \cap (R_j - \overline{R}_j)$
    if $R = \emptyset$ then
        return $\infty, \emptyset$                      ▷ no feasible positions to punch $A$
    $s \leftarrow$ (closest point to $\alpha$ in $R$)              ▷ the chosen machine position
    for $j \leftarrow 1, J$ do
        $\overline{R}_j \leftarrow \overline{R}_j \cap (R_j - s)$            ▷ possible relative positions of punch $j$
        $\overline{s}_j \leftarrow$ (center of $\overline{R}_j$)
    return $s, (\overline{s}_j)_j$                                  ▷ machine position and
                                       resulting relative positions of punches
```

(line 8). Additionally, the difference to the previous and following relative position ($\overline{s}'_j$ and $\overline{s}''_j$, respectively) of punch $j$ (if any) must not exceed $X$, that is, the two constraints

$$\overline{R}_j \subseteq [\overline{s}'_j - X, \overline{s}'_j + X] \qquad \forall j \in P \tag{6.3}$$

$$\overline{R}_j \subseteq [\overline{s}''_j - X, \overline{s}''_j + X] \qquad \forall j \in N \tag{6.4}$$

apply (lines 10 and 12). To translate restrictions on the absolute and relative positions of individual punches $j$ to an interval $R \subseteq [a, b]$ of possible machine positions, note that

$$s_j = s + \overline{s}_j$$

(the absolute position of a punch $j$ is the machine position $s$ changed by its relative position $\overline{s}_j$) and therefore

$$R \subseteq R_j - \overline{R}_j \tag{6.5}$$

(line 13). To compute $R$, we start with $[a, b]$ and iterate through all punches $j$ to apply eqs. (6.1) to (6.5). If we end up with an empty interval $R = \emptyset$, it is not possible to find a feasible machine position to punch the assignment $A$ (line 15).

Otherwise, we choose the machine position to be in $R$ and closest to $\alpha$ (line 16). Finally, the relative positions of all punches $j$ have to be set, using the restrictions $\overline{R}_j$ and $R_j$ in combination with the machine position $s$. The interval of possible relative positions is given by

$$\overline{R}_j \cap (R_j - s).$$

Within this interval, any value can be chosen as relative position $\overline{s}_j$, so we simply choose the center (line 19) with the intention that this might leave the relative position rather flexible for further adjustment in neighboring punching steps.

**Example 6.2** (Example 6.1 continued)**.** Consider the set of two holes examined in Example 6.1 with positions $x_1 = 100$, $x_3 = 400$ and an assignment

$$A = (1 \mapsto 3, 2 \mapsto 1)$$

of holes to punches. Furthermore, let the distances between punches in this simplified punching machine with reference punch $\overline{j} = 2$ be constrained by

$$\underline{X} = \begin{pmatrix} 0 & 100 & 200 \\ -200 & 0 & 100 \\ -400 & -200 & 0 \end{pmatrix} \quad \text{and} \quad \overline{X} = \begin{pmatrix} 0 & 200 & 400 \\ -100 & 0 & 200 \\ -200 & -100 & 0 \end{pmatrix}$$

and the maximum $x$-movement of $X = 70$ between consecutive punching steps. Assume there is a punching step with relative punch positions

$$(n_j)_j = (100, 0, -100)$$

after the examined step, but no punching step before ($P = \emptyset$).

The set of active punches is $A' = \operatorname{Img} A = \{1, 3\}$ with absolute $x$-positions $s_1 = 400$ and $s_3 = 100$. For $j = 1$ and $j = 3$, the range of possible absolute positions consists of this single position, that is,

$$R_1 = \{400\} \quad \text{and} \quad R_3 = \{100\}.$$

For punch $j = 2$, the restrictions on the distances to punches 1 and 3 have to be taken into account. Inserting into (6.1), we get

$$\begin{aligned} R_2 &= [\underline{X}_{21} + s_1, \overline{X}_{21} + s_1] \cap [\underline{X}_{23} + s_3, \overline{X}_{23} + s_3] \\ &= [-200 + 400, -100 + 400] \cap [-100 + 100, 200 + 100] = [200, 300]. \end{aligned}$$

The ranges of positions relative to the reference punch 2 are

$$\begin{aligned} \overline{R}_1 &= [100, 200] \cap [30, 170] = [100, 170] \\ \overline{R}_2 &= \{0\} \\ \overline{R}_3 &= [-200, -100] \cap [-170, -30] = [-170, -100] \end{aligned}$$

by (6.2) and (6.4). Consequently, the range of all possible machine positions is

$$R = \bigcap_j (R_j - \overline{R}_j) = [230, 300] \cap [200, 300] \cap [200, 270] = [230, 270].$$

Let us assume that the punching step after the examined step is at machine position $b = 500$. There is no step before, so the interval of a-priori possible machine positions is given by $[a, b] = [-\infty, 500]$. Assuming that we want to place the machine position as far to the left as possible, we set $\alpha = -\infty$. Therefore, we choose the machine position

$$s = 230.$$

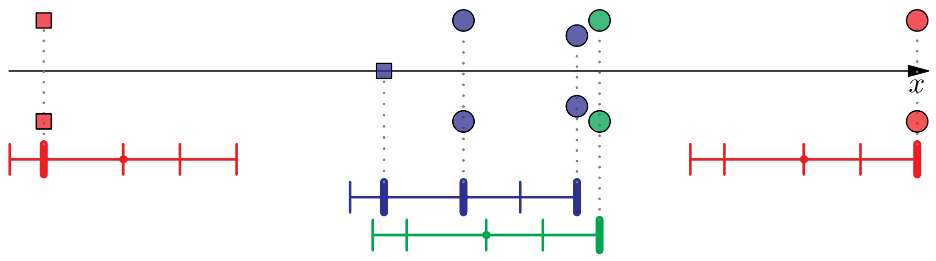
The relative punch positions are set to

$$\begin{aligned} \overline{s}_1 &= 170 \\ \overline{s}_2 &= 0 \\ \overline{s}_3 &= -130 \end{aligned}$$

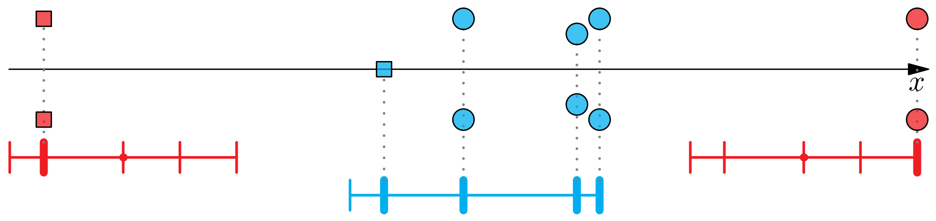
in accordance with $R_j$, $\overline{R}_j$, and $s$.

## 6.2. Combining Punching Steps

A crucial part of our heuristics to improve a punching plan is to attempt to combine pairs of punching steps into a single step. This obviously reduces the number of punching steps and creates “room” for distributing the punching steps more equally, thereby helping with both the step and the speed punching problem.



**(a)** Punching steps $p_{i-1}$, $p_i$, $p_{i+1}$, $p_{i+2}$ before combining.



**(b)** Punching steps after combining $p_i$ and $p_{i+1}$ into a single step $p$.

**Figure 6.1:** Combining two punching steps.

To combine punching steps, we iterate through all consecutive pairs $p_i$, $p_{i+1}$ of steps and try to punch all punched holes

$$\mathcal{N} = \operatorname{Img} p_i.n \cup \operatorname{Img} p_{i+1}.n$$

in one step (see Algorithm 6.3). Here, the image $\operatorname{Img} p_i.n$ of the mapping of punches to holes is the set of punched holes in step $p_i$.

All possible assignments $A \in \mathcal{A}$ of punches to the holes in $\mathcal{N}$ are considered, until the first assignment yields a feasible combination of the two punching steps (see section 6.1).

The surrounding punching steps $p_{i-1}$ (if $i > 1$) and $p_{i+2}$ (if $i < |S| - 1$) have to be taken into account as well. The relative positions of the punches in the potentially new punching step $p$ have to be chosen such that the positions in $p_{i-1}$ and $p_{i+2}$ are reachable from $p$, that is, within the limit imposed by parameter $X$.

The preferred $x$-position of the new punching step $p$ is at the center $\frac{a+b}{2}$ between $a = p_{i-1}.p$ and $b = p_{i+1}.p$ to create a close-to-equal distribution of machine positions. At the bordering punching steps, it is $-\infty$ (for the leftmost steps with $i = 1$) and $\infty$ (for the rightmost steps with $i = |S| - 1$), respectively, in order to move the step "as far outside" as possible.

The details of choosing the punch positions are encapsulated by the function PUNCH-POSITIONS defined in Algorithm 6.2. If corresponding positions can be determined, the new combined position $p$ is inserted in the punching plan instead of $p_i$ and $p_{i+1}$ (line 19). The corresponding assignment $p.n$ of holes to punches is simply given by the inverse $A^{-1}$ of the assignment $A \in \mathcal{A}$ of punches to holes.

**Algorithm 6.3:** Combining punching steps using dynamic $x$-movements.

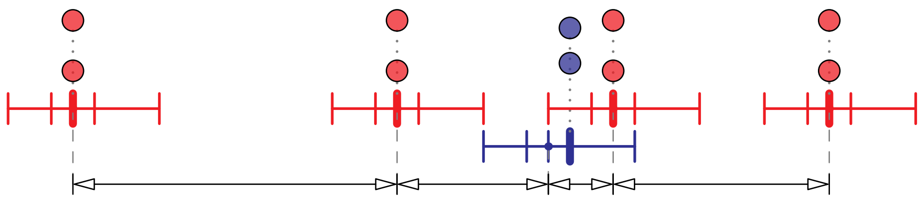
```
1:  function COMBINE-STEPS($S, (t_j)_j$)
        for all consecutive steps $p_i, p_{i+1} \in S$ do
3:          $\mathcal{N} \leftarrow \operatorname{Img} p_i.n \cup \operatorname{Img} p_{i+1}.n$        ▷ set of holes punched in step $p_i$ or $p_{i+1}$
            $\mathcal{A} \leftarrow$ HOLE-ASSIGNMENTS($\mathcal{N}, (t_j)_j$)
5:          if $i > 1$ then
                $(\overline{s}'_j)_j \leftarrow p_{i-1}.pos$, $a \leftarrow p_{i-1}.p$
7:          else
                $(\overline{s}'_j)_j \leftarrow \emptyset$, $a \leftarrow -\infty$
9:          if $i < |S| - 1$ then
                $(\overline{s}''_j)_j \leftarrow p_{i+2}.pos$, $b \leftarrow p_{i+2}.p$
11:         else
                $(\overline{s}''_j)_j \leftarrow \emptyset$, $b \leftarrow \infty$
13:         for all $A \in \mathcal{A}$ do
                $s, (\overline{s}_j)_j \leftarrow$ PUNCH-POSITIONS($A, (\overline{s}'_j)_j, (\overline{s}''_j)_j, [a, b], \frac{a+b}{2}$)
                                        ▷ determine punch positions punching $A$ at once,
                                                      with an absolute machine position
                                                  between the previous and next punch
15:             if $s < \infty$ then
                    $p \leftarrow$ POSITION($s, A^{-1}$)
17:                 $p.punches \leftarrow \operatorname{Img} A$
                    $p.pos \leftarrow (\overline{s}_j)_j$
19:                 Insert $p$ between $p_i$ and $p_{i+1}$ in $S$.
                    Delete $p_i$ and $p_{i+1}$ from $S$.
21:                 break                ▷ stop iterating through assignments $A \in \mathcal{A}$
        return $S$
```

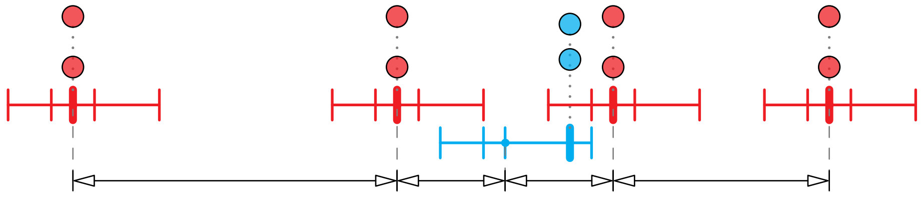
**Example 6.3** (Example 6.2 continued)**.** Assume that the holes 1 and 2 from Example 6.2 are punched in separate steps $p_1$, $p_2$. Assuming that there is a third hole punched in a step $p_3$ at position $b = p_3.p = 500$ with relative punch positions $(\overline{s}''_j)_j = (100, 0, -100)$, we set $a = -\infty$ as we want to move the combined punching step as far to the left as possible in order to increase the distance to $p_3$. In the course of Algorithm 6.3, the assignment

$$A = (1 \mapsto 3, 2 \mapsto 1)$$

is considered. Algorithm 6.2 is executed as outlined in the previous example with a desired machine position of $\alpha = \frac{a+b}{2} = -\infty$, yielding a machine position of $s = 230$ for the combined punching step $p$.

**Example 6.4.** An illustration of the *combine* heuristic using practical machine parameters as described in section 3.4 is shown in Figure 6.1. By applying Algorithm 6.3, all holes punched in the second and third punching step $p_i$, $p_{i+1}$ are punched together in a single step $p$.


**(a)** Punching steps $p_{i-2}$, $p_{i-1}$, $p_i$, $p_{i+1}$, $p_{i+2}$ before moving $p_i$.


**(b)** Punching steps after moving $p_i$ to the center between $p_{i-1}$ and $p_{i+1}$.

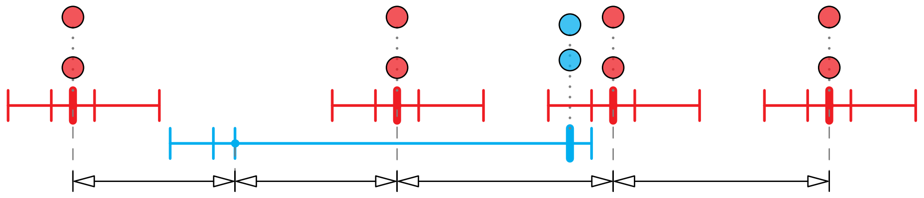

**(c)** Punching steps after moving $p_i$ to the center between $p_{i-2}$ and $p_{i-1}$.

**Figure 6.2:** Moving a punching step.

## 6.3. Moving Punching Steps

To improve the result further, we attempt to move punching steps such that they are more equally distributed, which is especially important in the speed punching problem. In other words, we try to make the machine movements between consecutive punching steps (the differences between absolute machine positions) as large as possible, while still punching the same set of holes in each step. This can potentially be achieved by using dynamic movements of the individual punches to change the absolute machine position (the absolute position $s_{i\bar{j}}$ of the reference punch $\bar{j}$) while all active punches remain at their respective absolute positions.

Specifically, we successively pick a number $L_{\text{move}}$ of punching steps $p_i$ with minimum distance to their following step $p_{i+1}$ and try to move $p_i$ so that the distance becomes bigger (see Algorithm 6.4). Usually, $L_{\text{move}}$ can be chosen as high as $N$ so that all punching steps are considered for movement. The assignment $A = p_i.n^{-1}$ of punches to holes in step $p_i$ shall remain the same. We propose three different strategies, creating a set $\mathcal{S}$ of alternative punching plans.

1. Keep $p_i$ between $p_{i-1}$ and $p_{i+1}$ and try to improve the punching plan by dynamic $x$-movements of the punches (line 7).
2. Move $p_i$ between $p_{i-2}$ and $p_{i-1}$ (line 16). In this case, the relative punch positions

in $p_{i+1}$ must be "reachable" directly from $p_{i-1}$ (as restricted by parameter $X$).

3. Analogously to 2, move $p_i$ between $p_{i+1}$ and $p_{i+2}$ (line 25). Again, the relative punch positions in $p_{i+1}$ must be reachable directly from $p_{i-1}$.

Obviously, we have to check whether a set of relative positions $(\overline{s}'_j)_j$ is reachable from another set of relative positions $(\overline{s}''_j)_j$. This can be done by simply iterating through all punches $j$ and testing whether

$$\left|\overline{s}'_j - \overline{s}''_j\right| \leq X. \tag{6.6}$$

If this is not the case for any punch $j$, the positions are not reachable from each other (see Algorithm A.5).

For each of the three strategies $r \in \{1,2,3\}$, a new feasible punching plan $S_r$ might arise, which is included in $\mathcal{S}$ then. As for the *combine* steps (see section 6.2) we use the function Punch-Positions defined in Algorithm 6.2 to determine the best punch positions given an assignment of the punches to holes. The preferred position is the center between the new surrounding steps of step $p_i$. This results in a machine position $s_r$ and relative punch positions $(\overline{s}_{rj})_j$ for each strategy $r$. A new punching plan $S_r$ is constructed by removing the step $p_i$ from $S$ and inserting a new step $p$ at machine position $s_r$ and with relative punch positions $(\overline{s}_{rj})_j$ into the plan.

The strategy $S^*$ in $\mathcal{S} \subseteq \{S, S_1, S_2, S_3\}$ resulting in the least costs is chosen to replace the existing punching plan $S$ (if better than $S$; line 28).

By these means, we iteratively replace the punching plan by a better one (with respect to the cost function). Punching steps that are attempted to being improved but cannot be moved effectively are included in a set $U$ excluded from further investigation (line 30 and 4).

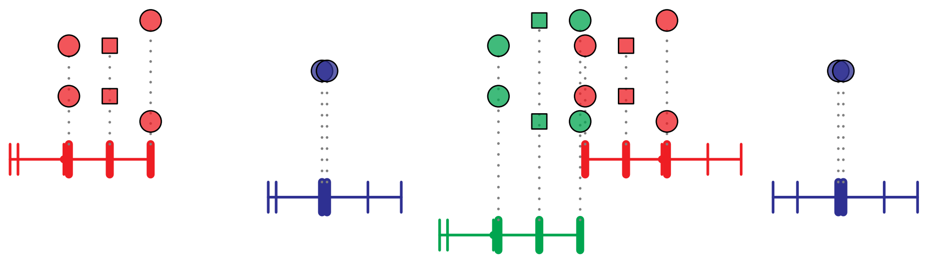
**Example 6.5.** Consider the punching plan depicted in Figure 6.2a. The punching step $p_i$ has the smallest distance to its successor $p_{i+1}$.

1. We consider moving the step to the center between $p_{i-1}$ and $p_{i+1}$, resulting in a punching plan $S_1$ (see Figure 6.2b).
2. Moving the step between $p_{i-2}$ and $p_{i-1}$ results in a punching plan $S_2$ (see Figure 6.2c). This is possible under the assumption that the relative $x$-positions of the punches in step $p_{i+1}$ are directly reachable from $p_{i-1}$, which is obviously the case here as they remain constant.
3. Moving the step between $p_{i+1}$ and $p_{i+2}$ is not possible, because the assignment $A$ of punches to holes has to remain the same, but the punches punching the two holes in $p_i$ cannot be moved to the left of the machine center, so the machine center cannot be to the right of the holes.

As depicted in Figure 6.2, the distribution of machine movements is most balanced in plan $S_2$ resulting in the least costs, so $S^* = S_2$ is chosen.

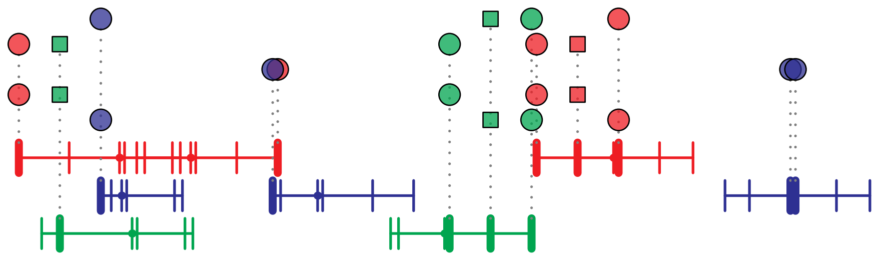
**Algorithm 6.4:** Moving punching steps to distribute machine stops more equally.

```
 1: function MOVE-STEPS($S$)
        $U \leftarrow \emptyset$        ▷ set of punching steps unsuccessfully attempted to improve
 3:     for $k \leftarrow 1, L_{\text{move}}$ do
            $p_i \leftarrow$ (step $p_i$ in $S \setminus U$ with minimum distance to next step $p_{i+1}$)
 5:         $A \leftarrow p_i.n^{-1}$        ▷ assignment of punches to holes
            $\mathcal{S} \leftarrow \{S\}$        ▷ set of alternative punching plans
 7:         $s_1, (\overline{s}_{1j})_j \leftarrow$ PUNCH-POSITIONS($A, p_{i-1}.pos, p_{i+1}.pos, p_{i-1}.p, p_{i+1}.p,$
                $\frac{p_{i-1}.p+p_{i+1}.p}{2}$)        ▷ keep current step $p_i$ between $p_{i-1}$ and $p_{i+1}$
            if $s_1 < \infty$ then        ▷ move successful
 9:             $S_1 \leftarrow S$
                Remove $p_i$ from $S_1$.
11:             $p \leftarrow$ copy $p_i$
                $p.pos \leftarrow (\overline{s}_{1j})_j$, $p.p \leftarrow s_1$
13:             Insert $p$ into $S_1$ between $p_{i-1}$ and $p_{i+1}$.
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_1\}$
15:         if IS-REACHABLE($p_{i-1}.pos, p_{i+1}.pos$) then
                $s_2, (\overline{s}_{2j})_j \leftarrow$ PUNCH-POSITIONS($A, p_{i-2}.pos, p_{i-1}.pos, p_{i-2}.p, p_{i-1}.p,$
                    $\frac{p_{i-2}.p+p_{i-1}.p}{2}$)        ▷ move current step $p_i$ before previous step $p_{i-1}$
17:             if $s_2 < \infty$ then
                    $S_2 \leftarrow S$
19:                 Remove $p_i$ from $S'$.
                    $p \leftarrow$ copy $s_i$
21:                 $p.pos \leftarrow (\overline{s}_{2j})_j$
                    $p.p \leftarrow p_2$
23:                 Insert $p$ into $S_2$ between $p_{i-2}$ and $p_{i-1}$.
                    $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_2\}$
25:             (Analogue procedure for moving $p_i$ after next step $p_{i+1}$.)
            $S^* \leftarrow \arg\min_{S' \in \mathcal{S}}$ PLAN-COSTS($S'$)
27:         if $S^* \neq S$ then
                $S \leftarrow S^*$        ▷ replace the punching plan with the improved one
29:         else
                $U \leftarrow U \cup \{p_i\}$        ▷ remember unsuccessful attempt to improve step $p_i$
31:     return $S$
```



‖first machine position: −350

**(a)** Start of punching plan without waste-minimizing heuristics.



|◁——▷| first machine position: 14350

**(b)** Start of punching plan with waste-minimizing heuristics.

**Figure 6.3:** Splitting punching steps in the beginning of the process and moving them to the right to minimize waste.

## 6.4. Reducing Waste

In addition to minimizing the costs representing the time needed for the punching problem, waste can be minimized by modifying the *move* improve step to maximize the machine position in the first punching step, as stated in eq. (3.26). We only have to set $c = \infty$ in the call to the function Punch-Positions (see Algorithm 6.2) in order to move the first punching step as far to the right as possible.

This is facilitated by the fact that maximizing machine movements is not important in the beginning of the process, as the first punches are performed while the metal is not rolling from the coil. Thus, we can safely ignore other improvement heuristics within a given length from the beginning of the hole pattern. Moreover, we can even split punching steps where several holes are punched into individual steps, as this might make it easier to move the steps to the right by reducing the constraints on active punch positions. This can be done by reassigning punches to individual holes as described in Algorithm 6.1.

**Example 6.6** (Example 2.2 continued)**.** The start of the pattern given in Figure 2.4 is depicted in Figure 6.3a without applying heuristics to minimize waste.

As shown in Figure 6.3b, when applying these heuristics, the punching steps in the beginning of the process are split up (the first two in this case), as it might be easier to move them to the right individually. All resulting five punching steps are modified so that punches further to the left punch the respective holes in order to move the punching steps to the right.

Obviously, the absolute machine position in the first punching step is increased to its maximum (14350), thereby minimizing the resulting waste.

## 6.5. Overall Improvements

In the overall improvement heuristics (*dynamic heuristics*) for a static punch plan, the *combine* and *move* heuristics are applied repeatedly after each other a number of $L_{\text{improve}}$ times, as depicted in Algorithm 6.5. This allows improvements from one phase to be utilized in the respective other.

**Algorithm 6.5:** Improving a solution.

```
1: function IMPROVE($S, (t_j)_j$)
       for $k \leftarrow 1, L_{\text{improve}}$ do
3:         $S \leftarrow$ COMBINE-STEPS($S, (t_j)_j$)
           $S \leftarrow$ MOVE-STEPS($S$)
5:     return $S$
```

The heuristics can be integrated in the algorithm for fixed machine configurations by applying them on each generated punching plan. Thus, Algorithm 5.1 is modified into Algorithm 6.6. In Algorithm 5.7, this corresponds to adding

$S_C \leftarrow \textsc{Improve}(S_C, (t_j)_j)$

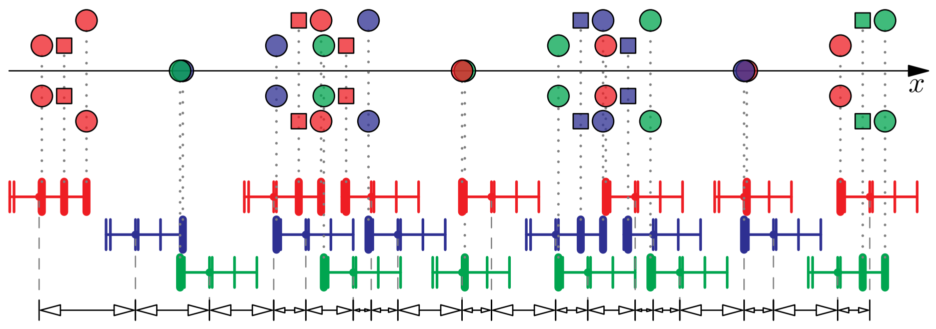
after line 4.

**Algorithm 6.6:** Overview of the proposed algorithm to solve the punching problem with heuristics utilizing dynamic machine movements.

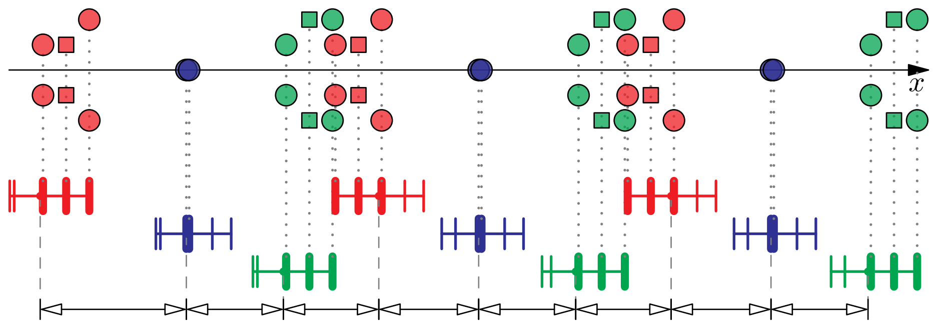
```
1: Generate a set $\mathcal{E}$ of equipments.
   Generate a set $\mathcal{C}$ of configurations using distances occurring in the pattern.
3: for all configurations $C \in \mathcal{C}$ do
       Determine a corresponding punching plan $S_C$.
5:     Improve the plan $S_C$ using dynamic machine movements.
   Choose the plan $S_C$ with minimum costs.
```

**Example 6.7** (Example 2.2 continued)**.** The punching plan presented in Example 2.2 results from applying our improvement heuristics (without waste-minimizing heuristics). A comparison of this plan to the original plan without utilizing dynamic machine movements is shown in Figure 6.4.

While the static plan (Figure 6.4a) uses 19 machine stops, the resulting dynamic plan (Figure 6.4b) only uses 9 stops. Obviously, the machine movements are distributed more equally.



**(a)** Punching plan before dynamic improvements.



**(b)** Punching plan after dynamic improvements.

**Figure 6.4:** Improving a punching plan by dynamic machine movements.

**Proposition 6.8.** The improvement of a solution using *combine* and *move* steps needs $\mathcal{O}(J^4 N)$ time.

*Proof.* In the *combine* step (Algorithm 6.3), there are at most $N$ punching steps and no more than $J^2$ possible assignments of the holes punched in two steps to the punches. For each assignment, there is an overhead of $\mathcal{O}(J)$ time (Algorithm 6.1). Determining the new punch positions (Algorithm 6.2) takes $\mathcal{O}(J^2)$ time.

In the *move* step (Algorithm 6.4), we calculate punch positions in $\mathcal{O}(J^2)$ time and determine whether punch positions are reachable from others in $\mathcal{O}(J)$ time. The plan costs can be updated locally in constant time. This is repeated $L_{\text{move}} = N$ times.

Altogether, the improvement needs

$$\mathcal{O}(NJ^2(J + J^2) + NJ^2) = \mathcal{O}(J^4 N)$$

time. □

CHAPTER 7

# Computational Results

In this chapter, we present evaluations and computational results of the proposed methods to solve the punching problem. First, we show that the mixed-integer linear program presented in chapter 3 is inefficient to use directly for practical problem instances (see section 7.1). Exact methods can only be used to optimize the punching plans for very small hole patterns with additional restrictions. This establishes the need for heuristics to solve the problem.

To evaluate the quality of our algorithm described in chapters 5 and 6, we compare its results to known optimal solutions of specific test cases. To make this comparison as meaningful as possible, a set of representative 300 test cases comprised by hole patterns occurring in practice is used, which were provided by the commissioning companies. The characteristics of these test cases are presented in section 7.2.

However, calculating optimal solutions is a hard problem which can only be solved for very small instances. This is why we split the evaluation in two parts:

1. The "core" part of the algorithm which computes the punching plan for a fixed machine configuration (as described in chapter 5) is evaluated using the test cases from practice in section 7.3.
2. The overall algorithm including the heuristics to account for dynamic $x$-movements is evaluated using randomly generated test cases that are small enough for being solved exactly in section 7.4.

To be able to calculate exact solutions within reasonable time, we disable the punches 1, 2, 5, and 6 in these two evaluation scenarios. Apart from this restriction, we use the machine parameters described in section 3.4.

To examine the impact of the dynamic heuristics utilizing dynamic $x$-movements of the punches described in chapter 6, we compare the results using fixed machine configurations from chapter 5 to the results after applying these heuristics in section 7.5.

There are many parameters to our algorithm that can be used to tune the quality of the results and its running time. Among them are

- the number of fixed machine configurations generated (see section 5.3) and
- the weight of the optimization of waste (see section 3.3).

We evaluate the influence of these choices on running time and quality of the results in sections 7.6 and 7.7, respectively.

All evaluation scenarios are summarized in Table 7.1.

**Table 7.1:** Overview of computational results, including the examined hole pattern(s) and the number $J$ of punches considered.

| Sec. | Description | Hole pattern(s) | Punches |
|---|---|---|---:|
| 7.1 | performance of exact method | 1 manually constructed | 12 |
| 7.3 | greedy heuristic for fixed machine configuration compared to exact solutions | 300 from practice with reduced repetitions | 10 |
| 7.4 | overall algorithm including dynamic heuristics compared to exact solutions | 100 random test cases | 10 |
| 7.5 | impact of dynamic heuristics | 300 from practice | 12 |
| 7.6 | influence of number of generated configurations | 300 from practice | 12 |
| 7.7 | influence of weight of waste-minimization | 300 from practice | 12 |

All computations are performed on a single core of an Intel® Core™ 2 Quad CPU (Q6600) running at 2.40 GHz with 8 GB of memory. Its operating system is Debian 6.0.6 with Linux kernel 2.6.32. We use ILOG CPLEX 11.0 [ILOG CPLEX Division, 2007] and ILOG AMPL 10.1 [Fourer, Gay, and Kernighan, 2002].

Our algorithm was implemented in C++ and compiled using gcc 4.4.5 [Stallman and the GCC Developer Community, 2008].

## 7.1. Performance of Exact Methods

To decide whether solving the mixed-integer linear program presented in chapter 3 is computationally tractable, we examine its application to a very simple hole pattern using the machine constraints stated in section 3.4.

Because of numerical instabilities with the piecewise linear objective function in the speed punching problem, we use a piecewise constant objective function as defined in

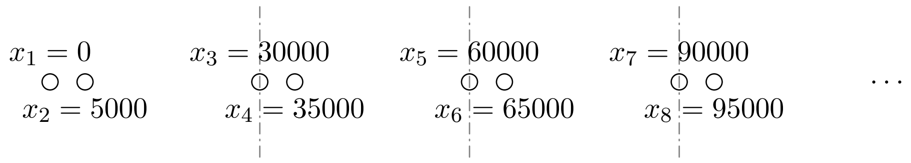


**Figure 7.1:** Simple hole pattern examined in analysis of exact method. It consist of several repetitions of a block of two holes.

eq. (3.25), where we choose

$$(\delta_r)_r = (1, 50, 100, 250, 500, 1000, 2500, 5000, 10000, 20000, 60000, 120000). \tag{7.1}$$

Therefore, zero-movements are assigned costs $\frac{1}{1} = 1$ and movements $< 50$ have costs $\frac{1}{50}$. Movements of 120000 or more are assigned costs 0.

The examined pattern consists of $R$ repetitions of a block of length 40000 two holes positioned at $x_1 = 0$ and $x_2 = 5000$ with $y$-coordinates $y_1 = y_2 = 0$ and the same hole type, resulting in $N = 2R$ holes (see Figure 7.1).

This pattern can simply be punched with a punching plan $S^*$ of $R$ steps using punch 1 for even and punch 3 for odd holes, for instance. The assigned costs

$$c(S^*) = \frac{R-1}{\delta_{11}}$$

are optimal, as all $R-1$ movements are 30000 and thereby larger than $\delta_{10} = 20000$; at the same time, there cannot be a punching plan with a movement of $\delta_{11} = 60000$ or more, as this would create a “gap” of $\geq \delta_{11} - \overline{X}_{1,12} = 32000$ where no hole can be punched; however, in every interval of this length there are at least two holes in the pattern.

Because the maximal machine length spanning from punch 1 to punch 12 is $\overline{X}_{1,12} = 28000$ as given by (3.29), not more than two holes can be punched at the same time, so the optimal number of punching steps is

$$I^* = R = \frac{N}{2}.$$

Our algorithm finds an optimal solution for a number of repetitions $R$ up to 100 within a fraction of a second.

Contrarily, this is not the case when using CPLEX to solve the corresponding mixed-integer linear program, which is implemented in the AMPL modeling language and presented in appendix B.1.

There are several parameters to CPLEX that can accelerate finding a solution.

- As mentioned in section 3.2, the number $I^*$ of optimal punching steps is not known a priori. A safe estimate to limit the number $I$ of considered steps is the number $N$ of holes. To examine the influence of the “tightness” of $I$, we also try setting $I$ to the optimal number $I^* = \frac{N}{2}$ of steps.

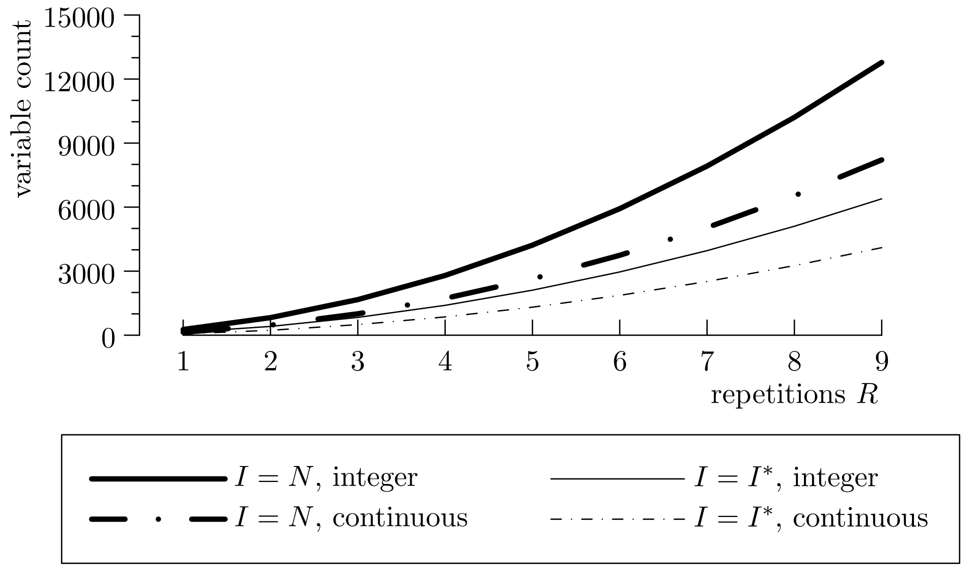


**Figure 7.2:** Number of integer and continuous variables in the resulting mixed-integer linear program, depending on the a-priori bound $I$ on the number of machine stops.

- An allowed relative tolerance $\varepsilon$ for optimizing integer variables (*MIP gap*) can be specified. Given an integer solution $S$ and a non-integer solution $\overline{S}$ resulting in a lower bound $c(\overline{S})$ on the objective value $c(S^*)$ of the optimal integer solution $S^*$, CPLEX stops its optimization process when

$$c(S) - c(\overline{S}) \leq \varepsilon\,(1 + c(S))\,.$$

We examine settings of $\varepsilon \in \{0.1, 0.5, 1\}$. Setting $\varepsilon = 1$ corresponds to neglecting the objective function and only trying to find a feasible integer solution.

When solving the model directly, the number of variables is very high, even with tight bounds on the number of steps (see Figure 7.2). While there are 85 integer variables (including binary variables) for one repetition of the pattern ($R = 1$), there are already 1913 integer variables for $R = 5$. For a practical instance size of $R = 60$, the problem has an intractably high number of 260688 integer variables. When an "uninformed" bound of $I = N$ on the number of resulting punching steps is chosen, the number of variables is twice as high, as can be expected.

The high number of variables is also reflected in the computation time to solve the program (see Figure 7.3). Even with a tight bound of $I = I^*$ and a MIP gap of $\varepsilon = 0.5$, only instances up to $R = 3$ repetitions of the two holes can be solved within 30 minutes.

Apparently, it is already hard to find *any* integer-feasible solution to the program. Even with a MIP gap of $\varepsilon = 1$, instances with more than 7 repetitions cannot be solved efficiently. The choices of $I$ and the MIP gap do not seem to have a significant impact on the tractability of solving the program.

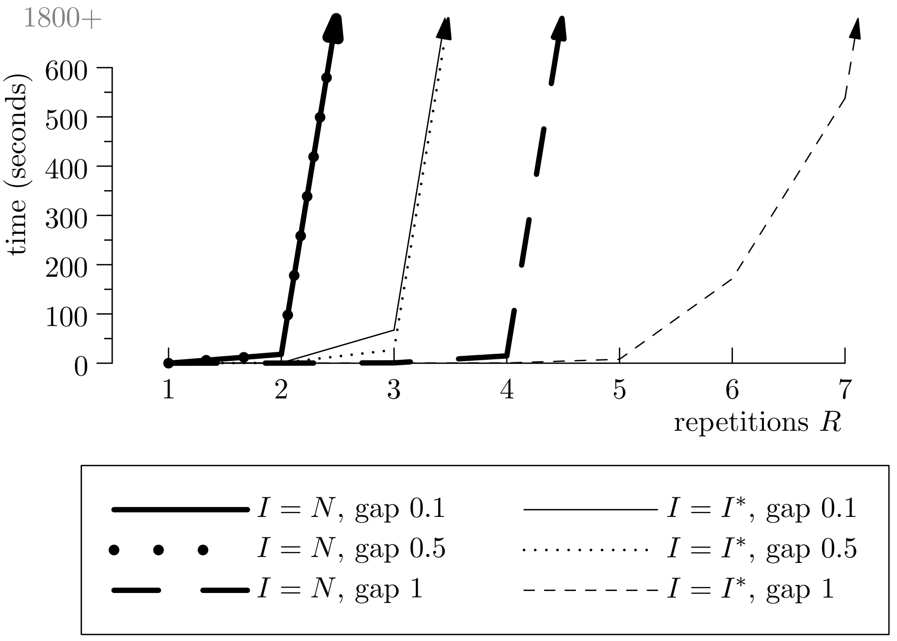


**Figure 7.3:** Computation time to solve the resulting mixed-integer linear program, depending on the a-priori bound $I$ on the number of machine stops and the MIP gap. Computation times longer than 30 minutes are depicted by upward arrows.

We also look at the relaxed version of the mixed-integer linear program where all variables are allowed to be continuous to examine the *integrality gap* of the model. This relaxation allows the variables $h'_{ijn}$ to assume values slightly lower than 1. These variables are used to ensure that a punch is at the correct position and is equipped with the correct tool when punching a hole through (3.8) and the bounds on the related variables $\overline{h}_{ijn}$, $h_{ijn}$ through eqs. (3.5) to (3.7) and (3.9). However, when $0 < h'_{ijn} < 1$, the constraint (3.8) can be fulfilled while $\overline{h}_{ijn}$ is not 0. At the same time, $h_{ijn}$ can be set to $1 - h'_{ijn} > 0$ in accordance with (3.9), allowing "partial punching" of holes even though the used punch is not feasible for the punched hole. As a consequence, active punches can be positioned at almost arbitrary positions, resulting in sufficiently high machine movements to yield costs of 0. Therefore, this model of the punching problem has a very high integrality gap, which might be a reason for the long computation times to solve it. Finding special cutting planes as described in section 3.5 could help with this problem.

As finding exact solutions using CPLEX—even with high optimality gaps and regardless of the chosen bound $I$ on the number of punching steps—is almost impossible for simple problem instances such as the analyzed pattern depicted in Figure 7.1, we conclude that only a heuristic approach can reliably generate solutions to practical problem instances within reasonable time.

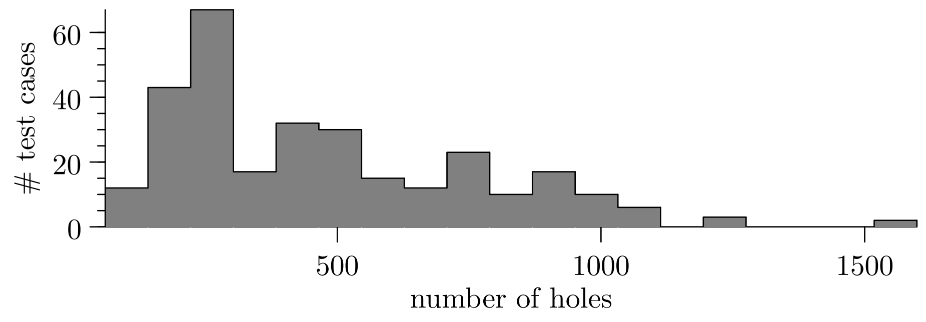


**Figure 7.4:** Histogram of the number of holes in the 300 test cases.

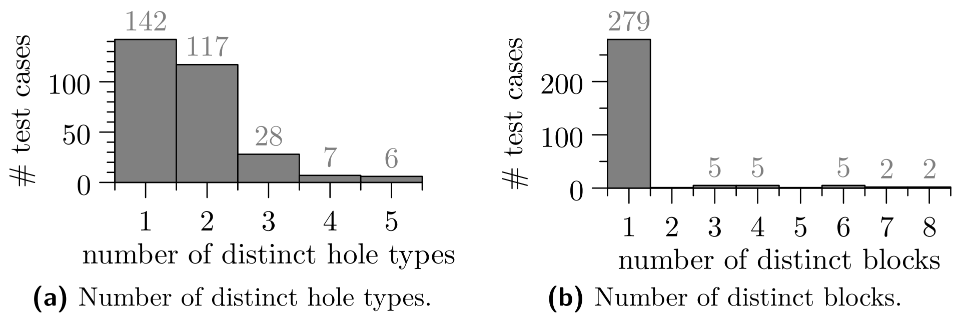


**Figure 7.5:** Histograms of the number of distinct hole types and the number of distinct blocks in the 300 test cases.

## 7.2. Test Cases

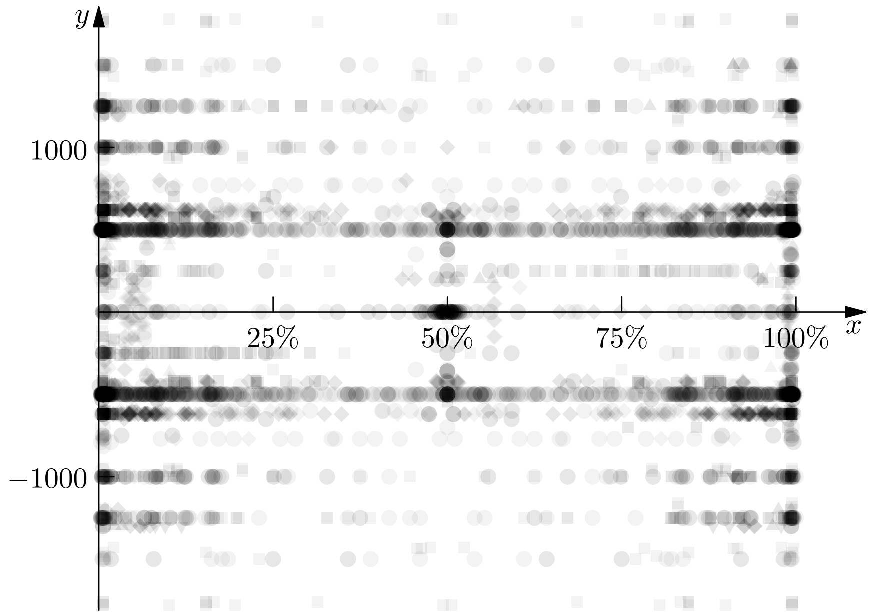
A representative sample of 300 test cases was provided by the commissioning companies to design and test our algorithm with data from practice. In this section, a short characterization of these test cases is given.

The number of holes in each pattern is in the order of a few hundreds up to a thousand, with a few patterns having 1200 or even 1600 holes (see Figure 7.4).

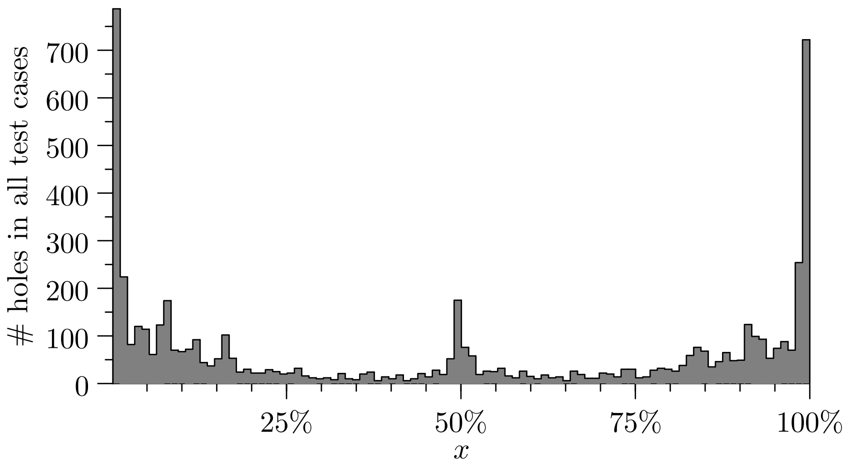
In most test cases (142 out of 300), there is only one type of holes. Two distinct hole types are common as well (117 test cases), whereas a few test cases have three, four, or five different hole types (see Figure 7.5a).

All patterns consist of several repetitions of one or more blocks of holes. In the majority of cases (279), there is a single block repeated up to 60 times. In the rest of the cases, up to eight distinct blocks appear (see Figure 7.5b).

To give a clearer picture of how these blocks of holes look like, Figure 7.6a shows the blocks of all 300 patterns combined into one plot by "overlaying" them. Apparently, holes are concentrated at the beginning, end, and the middle of each block. This becomes even more obvious when looking at the histogram of $x$-positions (relative to the block the length) in Figure 7.6b. A few patterns contain holes that are not placed symmetrically with respect to the $x$-axis (see Figures D.2 and D.3, for instance).



**(a)** All blocks in the 300 test cases plotted together. The $x$-axis is scaled relatively to the length of each block. Every hole is plotted with an opacity of 10%, so darker areas depict regions with more holes. Different shapes correspond to different hole types.



**(b)** Histogram of the $x$-positions of the holes (relative to the block length) in the distinct blocks of all 300 test cases.

**Figure 7.6:** Distribution of holes in each block of the 300 test cases.

## 7.3. Fixed Machine Configurations

To evaluate the algorithm dealing with fixed machine configurations (as described in chapter 5), we solve the underlying hitting set problem, which can be described as follows. Given a set of possible machine positions covering a set of holes each, find a subset of machine positions that cover all holes such that the resulting costs are minimum.

This is similar to the simplified punching problem (Definition 4.1) discussed in chapter 4. However, we want to be able to assign arbitrary costs to each movement of the machine (a difference between machine positions) instead of only minimizing the number of machine stops.

Formally, the problem can be expressed as the following linear integer program. Given are binary parameters $P_{in}$ specifying whether hole $n \in \{1, \ldots, N\}$ can be punched from machine position $i \in \{1, \ldots, I\}$ and costs $c_{ij}$ resulting between machine positions $i$ and $j$.

Binary decision variables $p_{ij} \in \{0, 1\}$, $i < j$, express whether position $i$ is chosen and is followed by position $j$, thereby creating a "connected path" through machine positions. Additionally, special start and end variables $\hat{p}_i \in \{0, 1\}$ and $\tilde{p}_i \in \{0, 1\}$ are needed to describe where the path starts and ends.

The objective is to minimize the costs of the path, that is,

$$\min \sum_{i,j:i<j} c_{ij} p_{ij}. \tag{7.2}$$

The following constraints ensure that a valid path through the positions is chosen. There has to be exactly one start, that is,

$$\sum_{i} \hat{p}_i = 1, \tag{7.3}$$

and one end, that is,

$$\sum_{i} \tilde{p}_i = 1. \tag{7.4}$$

The number of incoming and outgoing edges has to be the same in each vertex, that is,

$$\hat{p}_i + \sum_{k:k<i} p_{ki} = \sum_{j:i<j} p_{ij} + \tilde{p}_i \qquad \forall i. \tag{7.5}$$

Note that in an optimal solution, this sum is either 1 (if machine position $i$ is used) or 0 otherwise. Because the objective function is increased when a vertex is visited more than once in a path, values of (7.5) higher than 1 do not occur in an optimal solution.

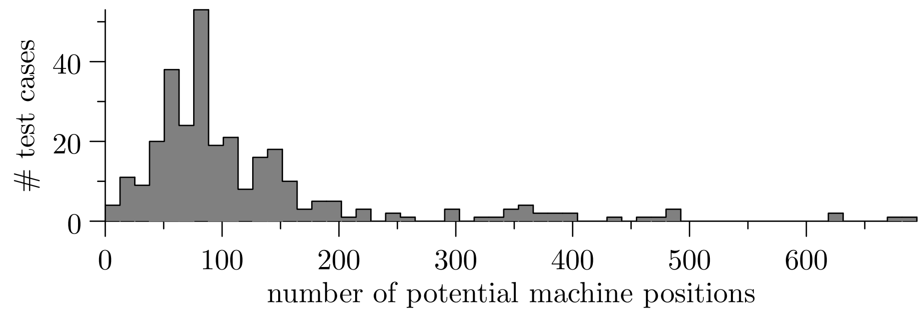


**Figure 7.7:** Histogram of the number of potential machine positions for the 300 test cases.

Finally, we express that every hole has to be punches (covered) by a visited position through the constraint

$$\sum_i P_{in}\Big(\sum_{j:i<j} p_{ij} + \tilde{p}_i\Big) \geq 1 \qquad \forall n. \tag{7.6}$$

The integer linear program

$$\begin{array}{llll} (P) & \min & \sum_{i,j:i<j} c_{ij}p_{ij} & \\ & \text{subject to} & \text{eqs. (7.2) to (7.6)} & \\ & & p_{ij} \in \{0,1\} & \forall i,j \\ & & \tilde{p}_i, \hat{p}_i \in \{0,1\} & \forall i \end{array}$$

can be implemented in the AMPL modeling language and can be solved exactly using CPLEX. See appendix B.2 for the full AMPL code of this model.

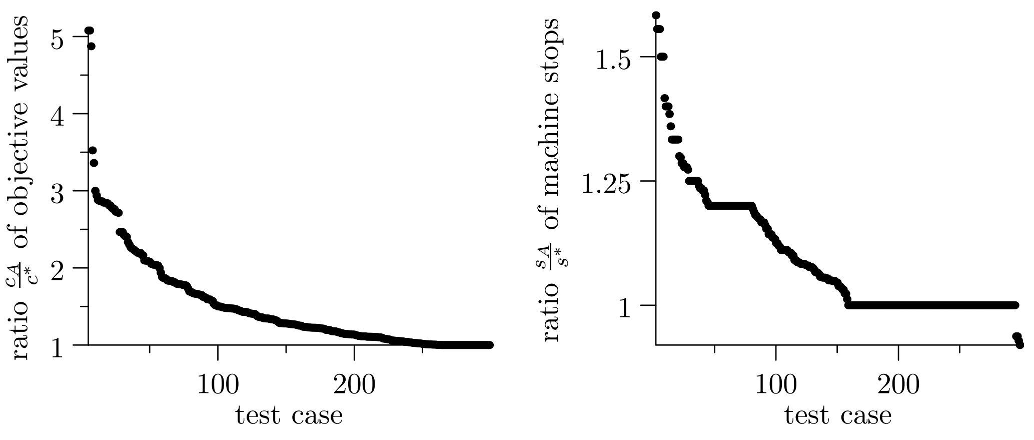
To be able to solve $(P)$ exactly in reasonable time, we have to reduce the size of the test cases. By reducing the number of repetitions of the blocks in each pattern from typical values of 60 to a maximum of 3, we retain the basic structure of the holes in the pattern, so the solutions and objective values are still representative.

We solve each problem using the algorithm described in chapter 5 and consider the fixed machine configuration that leads to the best solution. Then, the corresponding machine positions are constructed using Algorithm 5.5. The numbers of generated potential machine positions are shown in Figure 7.7. For most test cases, around a hundred machine positions are generated.

The costs of a movement $d$ between two machine positions $p_1$ and $p_2 > p_1$ are chosen to be

$$c(d) = \frac{1}{d} = \frac{1}{p_2 - p_1}$$

as described in section 3.3, eq. (3.23).



**(a)** Ratio $\frac{c_A}{c^*}$ of the resulting objective value $c_A$ and the determined optimal objective value $c^*$.

**(b)** Ratio $\frac{s_A}{s^*}$ of the resulting number of machine stops $s_A$ and the number of machine stops $s^*$ in the optimal solution.

**Figure 7.8:** Results using fixed machine configurations on 300 test cases from practice. Ratios are sorted in descending order.

Machine movements of 0 (resulting from two consecutive identical machine positions) have to be dealt with specially. They are assigned significantly high costs (greater than the number of holes $N$ in the pattern) so that the number of zero-movements is apparent from the overall costs of a punching plan.

To evaluate our algorithm $A$, we compare the resulting objective value $c_A$ to the optimal objective value $c^*$ obtained by CPLEX. The ratio

$$\frac{c_A}{c^*}$$

determines how "bad" our results are compared to the optimal solution.

The results are shown in Figure 7.8a. For 243 problem instances out of the 300 (81%), the ratio of the resulting objective value and the calculated optimal objective value is below 2.0. The majority of results (64.7%) are even within a 150% bound of the optimal value.

In most cases, our algorithm produces the same number of zero-movements as in the optimal solution. Only for three instances, our algorithm results in zero-movements where the optimal solutions has none. Consequently, the ratio of the objective values is incomparably high, which is why these instances are not depicted in Figure 7.8a.

Because this is only a part of the overall algorithm, the resulting costs are not the only important criterion in this step. Equally important is the resulting number $s_A$ of machine stops, and it turns out that our algorithm performs very well in this regard when compared to the number $s^*$ of stops in the optimal solution.

The ratio $\frac{s_A}{s^*}$ of the number of machine stops in our result and in the optimal solution for each problem instance is shown in Figure 7.8b. Only in two cases this ratio is above 1.5. For almost half of the problems (145), the number of machine stops is the same as in the optimal solution, and in three cases it is even lower.

A central aspect of our algorithm is execution speed. Although in most cases, solving the integer program using CPLEX is quite fast, there are cases in which it takes very long. For two problem instances, CPLEX did even not solve the program within several hours, so it is not possible to compare our results to the optimal values in these cases. The two patterns are similar to the one depicted in D.5 where many holes have equal $x$-positions, differing only in their $y$-positions.

Adding more holes up to practical instance sizes would increase the running time of CPLEX even more, thereby rendering the integer programming approach impractical.

In contrast to that, our algorithm finds solutions within seconds of time for every problem instance. While the problems were reduced in size for this analysis in order to find optimal solutions, the execution times of our algorithm are still representative because the time required is polynomial in the number of holes (see Theorem 3).

## 7.4. Dynamic Configurations

As the practical problems are too large for being solved exactly even after drastically reducing the number of blocks (see section 7.3), we generate 100 *random* test cases to evaluate our overall algorithm including the heuristics utilizing dynamic $x$-movements of the punches. This has the additional benefit of not only evaluating the algorithm against a given set of problem instances but examining its "average" behaviour as well.

The test cases are generated by placing $N = 6$ holes with

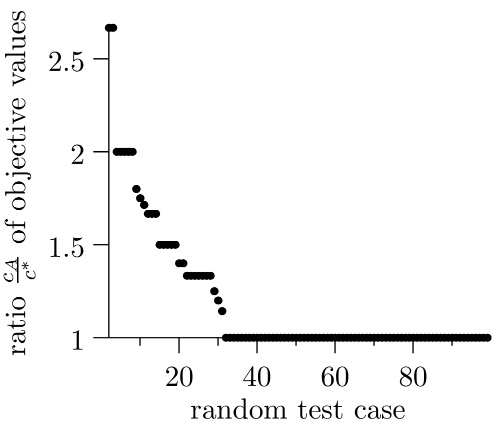
- $x$-coordinates randomly distributed in the range $[0, 100000]$,
- $y$-coordinates drawn randomly from $\{500, 1000\}$, and
- hole types drawn randomly from $\{14, 18\}$,

which are typical situations in practice as well.

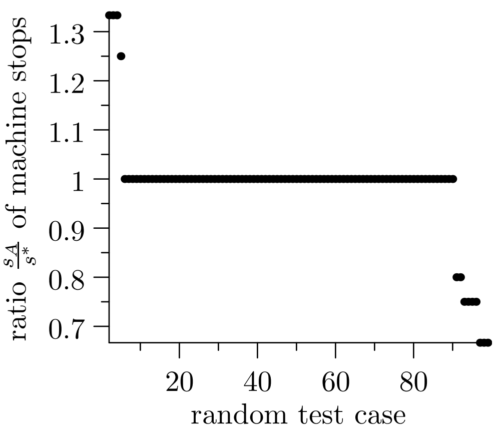
The choice of $N = 6$ is motivated by the results in section 7.3 as patterns with six holes are solvable within reasonable time, while larger patterns seem rather impossible to solve. As in section 7.3, exact solutions are computed using the AMPL model given in appendix B.1 and CPLEX. The same objective function as specified by eq. (7.1) is used.

The results of the comparison are shown in Figure 7.9. For 85 out of 100 problems, our solution $c_A$ is within a 150% bound of the optimal solution $c^*$. In 68 cases it is even equally good (see Figure 7.9a).

The results regarding the number of machine stops are even better. In 94 cases, this number $s_A$ is the same as $s^*$ in the optimal solution or even lower. Even in the worst case, $s_A$ is not more than four thirds of $s^*$ (see Figure 7.9b).



**(a)** Ratio $\frac{c_A}{c^*}$ of the resulting objective value $c_A$ and the determined optimal objective value $c^*$.



**(b)** Ratio $\frac{s_A}{s^*}$ of the resulting number of machine stops $s_A$ and the number of machine stops $s^*$ in the optimal solution.

**Figure 7.9:** Results using dynamic machine configurations on 100 random test cases. Ratios are sorted in descending order.

## 7.5. Impact of Dynamic Heuristics

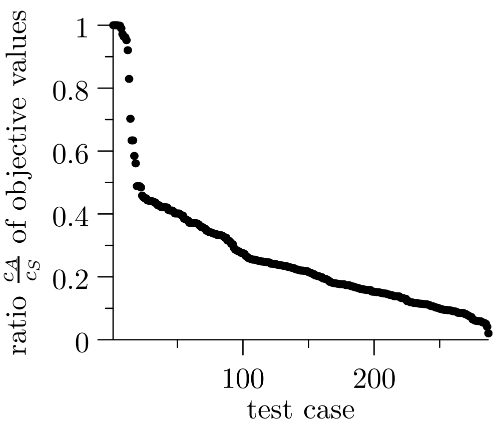
To evaluate the heuristics utilizing dynamic $x$-movements of the punches described in chapter 6, we compare the resulting punching plans after applying these *dynamic heuristics* to the "static plans" from chapter 5. Again, we use the set of 300 test cases from practice.

Given the resulting objective value $c_A$ of the overall algorithm (including dynamic heuristics) and the objective value $c_S$ of the corresponding static punching plan, we examine the ratio $\frac{c_A}{c_S}$ (see Figure 7.10a). For almost all test cases (282), the costs $c_A$ after applying dynamic heuristics is 50% or less of the costs before. For the majority (219) it is even less than a third. Only a few test cases are resistant to dynamic heuristics, mostly because the hole patterns are rather simple and the static solution is already sufficiently good.

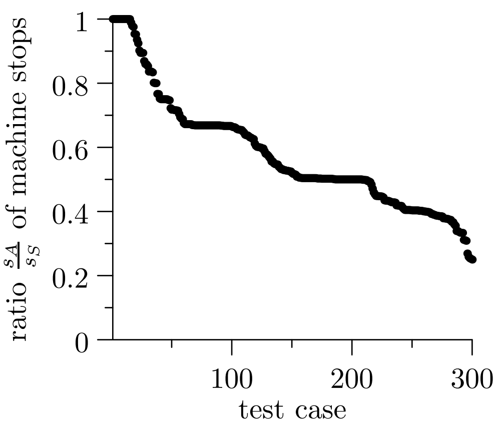
A similar picture emerges for the number of machine stops. We examine the ratio $\frac{s_A}{s_S}$ of the resulting number of machine stops $s_A$ with dynamic heuristics and the number of machine stops $c_S$ without them (see Figure 7.10b). For almost half of the test cases (143), the number of machine stops is reduced by half, which is mostly due to the *combine* heuristic from section 6.2.

Nevertheless, the impact of dynamic heuristics on the running time of the algorithm are moderate. We compare the times spent in the following four *phases* of the algorithm:

1. the generation of configurations, as described in section 5.1 and 5.3;



(a) Ratio $\frac{c_A}{c_S}$ of the resulting objective value $c_A$ with dynamic heuristics and the objective value $c_S$ without them.



(b) Ratio $\frac{s_A}{s_S}$ of the resulting number of machine stops $s_A$ with dynamic heuristics and the number of machine stops $s_S$ without them.

**Figure 7.10:** Impact of dynamic heuristics on 300 test cases from practice. Ratios are sorted in descending order.

2. solving the underlying hitting set problem using a greedy algorithm, as described in section 5.5;
3. applying the *combine* heuristic from section 6.2;
4. applying the *move* heuristic from section 6.3.

As the sole average of execution times over all test cases might not include all underlying information, we use *box plots* [McGill, Tukey, and Larsen, 1978] to visualize results in this and the following sections. Box plots include the following information.

- A box represents the data points within the first and third quartile.
- A line inside the box represents the median.
- A cross depicts the average of the data.
- So-called "whiskers" extend to the last data point within a 1.5-multiple of the inner quartile range (IQR, the difference between third and first quartile).
- Outliers are shown as additional dots, if applicable.

As depicted in Figure 7.11, dynamic heuristics account for approximately half of the running time of the overall algorithm on average. A positive feature of the *combine* and *move* heuristics is that their time behavior does not highly depend on the structure of the hole pattern, but mostly on the number of considered punching steps. This is reflected in the fact that there are less outliers regarding computation time compared to the generation of configurations and the greedy phase, which can be seen as the averages are closer to the median.

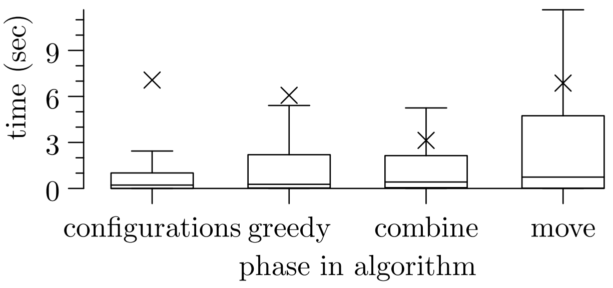


**Figure 7.11:** Running time of the different phases in the algorithm among 300 test cases from practice. The phases are the generation of *configurations*, the *greedy* algorithm to solve the underlying hitting set problem, and the *combine* and *move* dynamic heuristics. Box plots are used to depict the quartiles and averages of data (outliers not shown).

## 7.6. Number of Generated Configurations

As our algorithm generates a set of fixed machine configurations and calculates a punching plan for each of them, the results strongly depend on the number of generated configurations. Recall from section 5.3 that there are several points in the generation of configurations at which this number is reduced from what is theoretically available to what is practicably tractable, configurable by individual parameters.

- By adding a single punch to a configuration, no more than $L_{\text{step}}$ configurations are added.
- For each permutation of punches, a maximum of $L_{\text{permutation}}$ configurations are kept.
- For each equipment $(t_j)_j$, a maximum of $L_{\text{equipment}}$ configurations are kept
- Altogether, no more than $L_{\text{all}}$ configurations are generated.

We examine a set of choices $l$ as given in Table 7.2 and their influence on the set of 300 test cases from practice.

For each of these settings, we evaluate the resulting costs, machine stops, and execution time of our algorithm (see Figure 7.12). As in section 7.3, we use the objective function

$$c(d) = \frac{1}{d}$$

as costs for each movement $d$ of the machine (see section 3.3).

The results show that for many test cases, a small number of configurations is already enough to yield good results. As shown in Figure 7.12a, the median of the costs decreases only slightly when the number of generated configurations is raised from 100 to 10000. At the same time, the average costs decrease from around 50 to 35.

Special attention has to be given to results with infinite costs, resulting from zero-movements of the machine (two consecutive stops at the same position). As they

**Table 7.2:** Examined choices regarding the number of generated configurations.

| $l$ | $L_{\text{step}}$ | $L_{\text{permutation}}$ | $L_{\text{equipment}}$ | $L_{\text{all}}$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 10 |
| 3 | 1 | 1 | 1 | 50 |
| 4 | 1 | 1 | 1 | 100 |
| 5 | 1 | 1 | 4 | 2000 |
| 6 | 2 | 4 | 8 | 2000 |
| 7 | 2 | 5 | 10 | 5000 |
| 8 | 1 | 1 | 4 | 10000 |
| 9 | 2 | 4 | 8 | 10000 |
| 10 | 10 | 20 | 40 | 10000 |

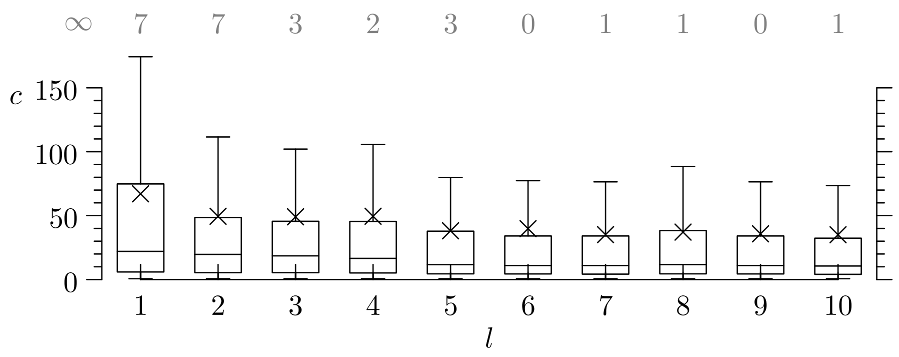
would distort the visualization of results, they are depicted separately in Figure 7.12a. However, zero-movements do not occur often and without significantly correlating to the number of generated configurations.

Another measure of the quality of results is the number of machine stops in a punching plan (see Figure 7.12b). As for the costs, they are not affected significantly by the number of generated configurations in many cases. However, for some “hard” cases, generating more configurations is crucial, as the reduction of top outliers for larger settings $l$ in Figures 7.12a and 7.12b suggests.

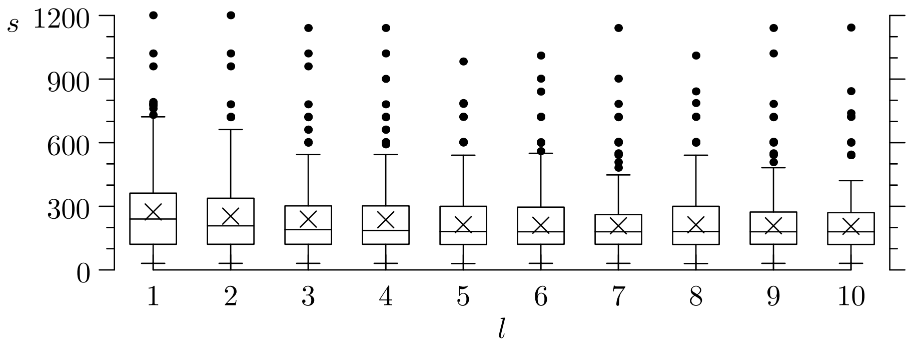
As can be expected, the execution time is proportional to the overall number $L_{\text{all}}$ of generated configurations. Their distribution among test cases is very skewed, though. While it is not much higher than a minute for 75% of the test cases even for 10000 configurations with the setting $l = 10$, there are a few test cases that take a very long time to solve and that increase the average over all test cases even above the third quartile (see Figure 7.12c).

Therefore, the maximum execution time is particularly interesting, because it should be within reasonable limits even for the worst case. As shown in Figure 7.12c, it ranges from 250 seconds for $l = 1$ up to almost one hour for $l = 7$. The time limit for each problem instance in practice is around ten minutes.

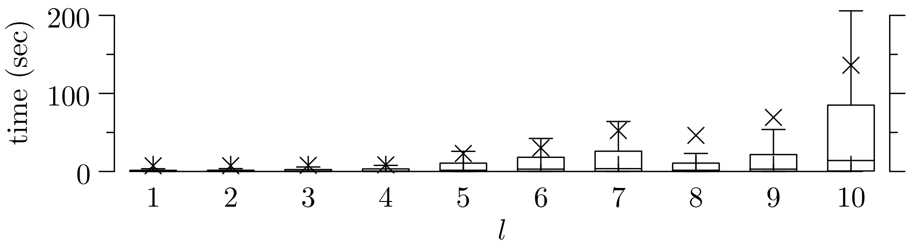
Informed by these results, a reasonable tradeoff between the quality of results and execution time has to be made. Considering the limits on computation time in practice, we choose the setting $l = 5$ as the default setting in our implementation delivered to the commissioning companies.



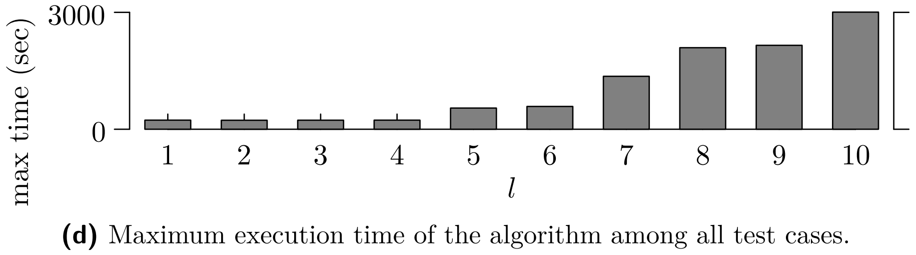
**(a)** Costs (outliers not shown). Infinite costs are excluded from the analysis but their number of occurrences is printed on top.



**(b)** Number of machine stops.



**(c)** Execution time of the algorithm (outliers not shown).



**(d)** Maximum execution time of the algorithm among all test cases.

**Figure 7.12:** Results and execution time depending on the setting $l$ of generated configurations. Box plots are used to depict the quartiles, averages, and outliers of the data.

## 7.7. Weight of Waste Minimization

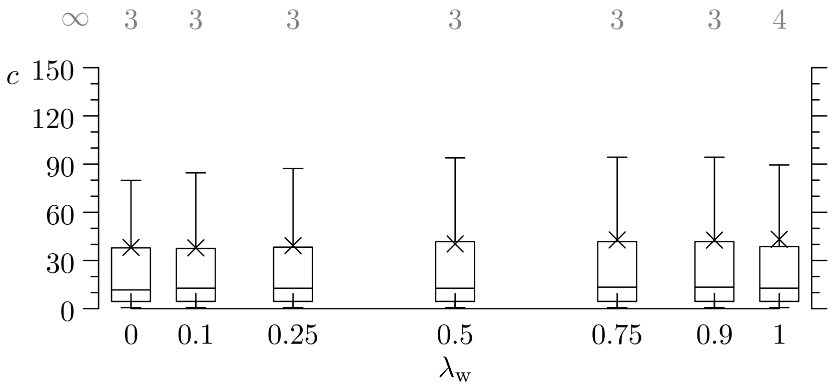
A critical parameter to our algorithm is the “weight” $\lambda_\mathrm{w}$ of the optimization of waste opposed to the cost function $c$ that measures the time needed for the punching process, as introduced in section 3.3 by eq. (3.27). It mainly influences two parts of the algorithm.

1. The equipment of the machine is chosen rather towards minimizing the waste in the beginning for larger values of $\lambda_\mathrm{w}$ (see Algorithm 5.1).
2. For values $\lambda_\mathrm{w} > 0$, the waste-decreasing movement of the first punching step as described in section 6.4 is enabled.

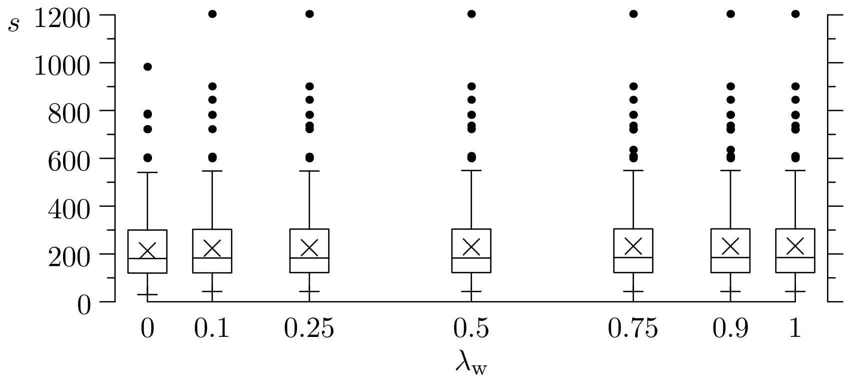
As shown in Figure 7.13, the waste-decreasing movement heuristic has the larger influence. Even for very small values of $\lambda_\mathrm{w} > 0$, waste is reduced significantly (see Figure 7.13c). Increasing $\lambda_\mathrm{w}$ further only has a small influence on the resulting waste. The biggest change besides $\lambda_\mathrm{w}$ being 0 or not comes with $\lambda_\mathrm{w}$ being set greater than 0.5, when waste has higher priority than the cost function so the machine configuration is chosen primarily upon its influence on the start of the process.

The reason that waste does not disappear even for $\lambda_\mathrm{w} = 1$ is that even with this setting, the configuration needed to eliminate waste (which requires the leftmost punch punching the first hole) might not be among the configurations considered in our algorithm. However, this only happens in very rare cases.

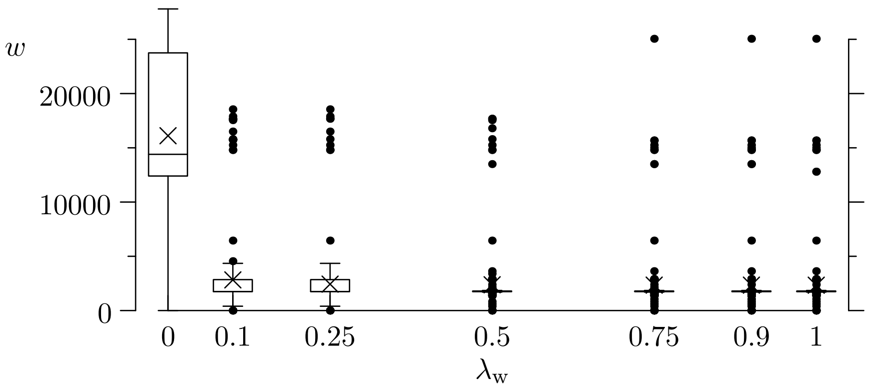
The influence of $\lambda_\mathrm{w}$ on the speed of the punching process is rather small. Only in a few edge cases, the number of machine stops increases slightly (see Figure 7.13b), while the cost function remains almost unchanged (see Figure 7.13a). Likewise, changing $\lambda_\mathrm{w}$ has no influence on the execution time of the algorithm. Therefore, setting $\lambda_\mathrm{w}$ to a small value around 0.1 seems reasonable in practice.



**(a)** Costs (outliers not shown). Infinite costs are excluded from the analysis but their number of occurrences is printed on top.



**(b)** Number of machine stops.



**(c)** Waste. Notable are the "jumps" at $\lambda_w = 0$ and $\lambda_w = 0.5$.

**Figure 7.13:** Results depending on the weight $\lambda_w$ of waste optimization. Box plots are used to depict the quartiles, averages, and outliers of the data.

CHAPTER 8

# Concluding Remarks

In this thesis, we introduce a novel problem arising in industry that has not been studied before. We develop a corresponding mathematical model and study its complexity. We show that a simplified variant with additional constraints is solvable in polynomial time using a graph-theoretic approach and dynamic programming. However, due to the simplification and an inefficient worst-case behavior, this does not help with practical problem instances. Consequently, we present a heuristic algorithm to calculate solutions efficiently. Our computational results suggest that the generated solutions are close to optimal solutions. Particularly, good punching plans are determined in short time for practical problem instances.

**Limitations** Still, there are some limitations to the proposed algorithm in this thesis.

- While we present a mixed-integer program (see chapter 3), a dynamic programming approach (see section 4.4), and heuristics (see chapters 5 and 6) to solve the punching problem and show that only the latter is useful in practice, we do not attempt a "hybrid" approach, that is, solving some partial problems using integer programs while still relying on heuristics. A modified and extended dynamic programming approach could be examined as well.
- Extensive refinements of the mixed-integer programs and the corresponding solver CPLEX is beyond the scope of this thesis. For instance, finding appropriate cutting planes (see section 3.5) could significantly speed up the process of finding an optimal or close-to-optimal solution.
- The precise speed of the examined punching machine and its actual relation to the movements between consecutive punching steps is not known. The objective functions presented in section 3.3 and used throughout this thesis arose through feedback from the commissioning companies regarding the visual appearance of punching plans without measuring the resulting speed of the process directly.

**Future work** While the problem can be considered solved from a practical point of view, there are several theoretical questions that would still be interesting to investigate in the future.

- The computational complexity of the general punching problem is not proven. Although a polynomial reduction from the $\mathbb{NP}$-complete hitting set problem seems reasonable, this is still an open problem.
- While approximation bounds can probably not be given for the general case, there might be special classes of instances where quality guarantees for our proposed algorithm can be made. It would be interesting to search for criteria on the hole pattern that allow for this.
- The heuristics to deal with dynamic movements of the punches are iteratively applied to individual positions in order to improve the punching plan. It would be interesting to investigate whether these improvements "converge" to a final punching plan and under what circumstances this is optimal.

However, even with these theoretical questions still open, an implementation of the proposed algorithm in C++ is successfully used by the commissioning companies in various manufactories all over the world.

APPENDIX A

# Additional Algorithms

In this appendix, some algorithms described in chapters 5 and 6 are defined formally.

Algorithm A.1 tests whether a given configuration is feasible with regards to the $x$-distances between punches, as described by eq. (5.1) in section 5.3.

**Algorithm A.1:** Testing whether a given (partial) fixed configuration $C$ is feasible.

```
function IS-CONFIGURATION-FEASIBLE($C$)
    for $j \leftarrow 1, J$ do
        $(t_j, s_j) \leftarrow C_j$
        for $k \leftarrow j+1, J$ do
            $(t_k, s_k) \leftarrow C_k$
            if not $\underline{X}_{jk} \leq s_j - s_k \leq \overline{X}_{jk}$ then
                return false
    return true
```

Algorithm A.2 normalizes punch positions, as described by eq. (5.2).

**Algorithm A.2:** Normalizing relative punch positions in a configuration $C$ so that the reference punch $\overline{j}$ has position 0.

```
function NORMALIZE-POSITIONS($C$)
    $(t_{\overline{j}}, s_{\overline{j}}) \leftarrow C_{\overline{j}}$      ▷ tool and relative position of the reference punch $\overline{j}$
                                                         in the given configuration $C$
    $C' \leftarrow \emptyset$                                ▷ empty resulting configuration
    for $j \leftarrow 1, J$ do
        $(t_j, s_j) \leftarrow C_j$                  ▷ tool and relative position of punch $j$
        $C'_j \leftarrow (t_j, s_j - s_{\overline{j}})$       ▷ add punch $j$ at position $s_j - \overline{s}_{\overline{j}}$
                                                         to the resulting configuration
    return $C'$
```

Algorithm A.3 shows how to compute the costs of a given punching plan, as mentioned in section 5.5.

**Algorithm A.3:** Costs of a punching plan $S$.

```
1: function PLAN-COSTS($S$)                              ▷ $S$ is a multiset of POSITIONS
       $S \leftarrow [s.p \mid s \in S]$
3:     Sort $S$ in ascending order.
       return $\sum_{i=1}^{|S|-1} c(S_{i+1} - S_i)$
```

Algorithm A.4 determines a punching plan for a given fixed machine configuration $C$, as described in section 5.5.

**Algorithm A.4:** Punching plan for a given configuration.

```
1:  function PUNCHING-PLAN($C$)
        $\mathcal{P} \leftarrow$ CONSTRUCT-POSITIONS($C$)
3:      $E \leftarrow \{\{p, n\} \mid p \in \mathcal{P}, n \in \{1, \ldots, N\}, n \in \operatorname{Img} p.n\}$
        $G \leftarrow (\mathcal{P} \cup \{1, \ldots, N\}, E)$
5:      $\hat{\sigma} \leftarrow (1, \ldots, N)$
        $\tilde{\sigma} \leftarrow$ (ordering of holes with ascending degree in G)
7:      $\Sigma \leftarrow \{\hat{\sigma}, \tilde{\sigma}, \text{Reversed}(\hat{\sigma}), \text{Reversed}(\tilde{\sigma})\}$
        for all $\sigma \in \Sigma$ do
9:          $S_\sigma \leftarrow$ GREEDY-PLAN($G, \sigma$)
        $\sigma^* \leftarrow \arg\min_\sigma$ PLAN-COSTS($S_\sigma$)
11:     return $S_{\sigma^*}$
```

Algorithm A.5 tests whether two sets of relative punch positions are reachable from each other, as constrained by eq. (6.6) in section 6.3.

**Algorithm A.5:** Testing whether the relative punch positions $\overline{s}''_j$ are reachable from the positions $\overline{s}'_j$.

```
1: function IS-REACHABLE($(\overline{s}'_j)_j, (\overline{s}''_j)_j$)
       for $j \leftarrow 1, J$ do
3:         if $\left|\overline{s}'_j - \overline{s}''_j\right| < X$ then
               return false
5:     return true
```

APPENDIX B

# Mathematical Models in AMPL

In this appendix, the mathematical models are formalized in the AMPL modeling language.

## B.1. Full Model

The “full” model is described in chapter 3 and implements the punching problem with piecewise constant cost function as defined in (3.25). It is evaluated in section 7.1 and used in section 7.4 to compare our algorithm to exact results.

```
param N_count;  # number of holes
param J_count;  # number of punches
param T_count;  # number of tools
param I_bound;  # bound on number of machine stops

set N := 1..N_count;     # set of holes
set J := 1..J_count;     # set of punches
set T := 1..T_count;     # set of tools
set I := 1..I_bound;     # set of machine stops

param x{n in N};     # x-positions of holes
param y{n in N};     # y-positions of holes
param z{n in N, t in T};     # tools of holes

param T_{j in J, t in T};     # feasible tools for punches
param X;                      # maximum x-movement after each step
param X_lower{j in J, k in J};  # minimum x-distances between
    punches
param X_upper{j in J, k in J};  # maximum x-distances between
    punches
```

```
param Y_lower{j in J, k in J};  # minimum y-distances between
    punches
param Y_upper{j in J, k in J};  # maximum y-distances between
    punches
param V_lower{j in J};  # minimum y-positions of punches
param V_upper{j in J};  # maximum y-positions of punches
param j_ref;            # reference punch

var s{i in I, j in J};      # absolute punch positions in each
    step
var s_rel{i in I, j in J};  # relative punch positions
var v{i in I, j in J};      # y-positions of punches
var u{j in J, t in T} binary;   # tools assigned to punches
var h{i in I, j in J, n in N} binary;   # punch j punching hole n
    in step i
var h_neg{i in I, j in J, n in N} integer;  # auxiliary
var h_if{i in I, j in J, n in N} binary;    # auxiliary
var a{i in I, j in J} binary;   # punch j active in step i
var a_any{i in I} binary;       # any punch active in step i

param R;                # number of linearization segments
param delta{r in 0..R}; # linearization steps of cost function

param M := (max{n in N} x[n]) + 1000 + delta[R] * I_bound;  #
    upper bound on h_neg[i,j,n];

var d{i in 2..I_bound}; # machine displacements
var d_interval{i in 2..I_bound, r in 0..R} binary;  #
    linearization interval

param costs_interval{r in 0..R} integer
= delta[R] / delta[r];  # costs in interval

minimize costs:
sum{i in 2..I_bound} sum{r in 0..R} d_interval[i,r] *
    costs_interval[r];

# distance intervals
subject to distance_interval{i in 2..I_bound, r in 0..R}:
d[i] >= delta[r] - d_interval[i,r] * delta[r];

# distance definition
subject to distances{i in 2..I_bound}:
d[i] = s[i,j_ref] - s[i-1,j_ref];

# punch on valid x-position when punching
subject to x_pos_1{i in I, j in J, n in N}:
h_neg[i,j,n] >= x[n] - s[i,j];
subject to x_pos_2{i in I, j in J, n in N}:
```

```
h_neg[i,j,n] >= s[i,j] - x[n];

# punch on valid y-position when punching
subject to y_pos_1{i in I, j in J, n in N}:
h_neg[i,j,n] >= y[n] - v[i,j];
subject to y_pos_2{i in I, j in J, n in N}:
h_neg[i,j,n] >= v[i,j] - y[n];

# punch has correct tool when punching
subject to tools_1{i in I, j in J, n in N, t in T}:
h_neg[i,j,n] >= z[n,t] - u[j,t];
subject to tools_2{i in I, j in J, n in N, t in T}:
h_neg[i,j,n] >= u[j,t] - z[n,t];

# coupling of auxiliary "if-then" variables
subject to if_1{i in I, j in J, n in N}:
h_neg[i,j,n] <= M * h_if[i,j,n];      # => h_if = 1 if h_neg > 0
subject to if_2{i in I, j in J, n in N}:
h[i,j,n] <= 1 - h_if[i,j,n];          # => h = 0 if h_neg > 0
                                      # => h = 1 only if h_neg = 0

# punch active when punching any hole
subject to step{i in I, j in J, n in N}:
a[i,j] >= h[i,j,n];

# punch not active when not punching any hole
subject to step_max{i in I, j in J}:
a[i,j] <= sum{n in N} h[i,j,n];

# every hole punched once
subject to holes_punched{n in N}:
sum{i in I, j in J} h[i,j,n] = 1;

# punches are equipped with feasible tools
subject to punches_feasible_tools{j in J, t in T}:
u[j,t] <= T_[j,t];

# punches have exactly one tool each
subject to punches_one_tool_each{j in J}:
sum{t in T} u[j,t] = 1;

# x-distances between punches are feasible
subject to x_distances{i in I, j in J, k in J}:
X_lower[j,k] <= s[i,j] - s[i,k] <= X_upper[j,k];

# y-distances between punches are feasible
subject to y_distances{i in I, j in J, k in J}:
Y_lower[j,k] <= v[i,j] - v[i,k] <= Y_upper[j,k];
```

```
# y-positions of punches are feasible
subject to y_feasible{i in I, j in J}:
V_lower[j] <= v[i,j] <= V_upper[j];

# relative x-movements of punches are feasible
subject to x_movements_1{i in 2..I_bound, j in J}:
s_rel[i,j] - s_rel[i-1,j] <= X;
subject to x_movements_2{i in 2..I_bound, j in J}:
s_rel[i-1,j] - s_rel[i,j] <= X;

# relation between absolute and relative punch positions
subject to abs_rel_positions{i in I, j in J}:
s_rel[i,j] = s[i,j] - s[i,j_ref];

# no empty punching step
subject to no_empty_step{i in 2..I_bound}:
a_any[i] <= a_any[i-1];

# punching step active when any punch active
subject to any_punch_min{i in I, j in J}:
a_any[i] >= a[i,j];
subject to any_punch_max{i in I}:
a_any[i] <= sum{j in J} a[i,j];
```

## B.2. Model for Fixed Machine Configurations

The model for a fixed machine configuration is described in section 7.3 and is used to evaluate the greedy hitting set heuristics described in section 5.5.

```
param N_count;       # number of holes
param I_count;       # number of possible machine positions

set N = 1..N_count; # set of holes
set I = 1..I_count; # set of possible machine positions

param P{i in I, n in N} binary; # hole n can be punched from
    machine position i
param c{i in I, j in I} >= 0;   # costs between positions i and j

var p{i in I, j in I: i < j} binary;    # positions i and j are
    used
var p_first{i in I} binary;      # position i is the first position
var p_last{i in I} binary;       # position i is the last position

minimize costs:
sum{i in I, j in I: i < j} c[i,j] * p[i,j];

```

```
# every hole is punched
subject to every_hole_punched{n in N}:
sum{i in I} (P[i,n] * (sum{j in I: i < j} p[i,j] + p_last[i])) >=
    1;

# incoming = outgoing edges along path
subject to path {i in I}:
p_first[i] + sum{j in I: j < i} p[j,i] = sum{j in I: i < j} p[i,j]
     + p_last[i];

# path has exactly one start
subject to path_start:
sum{i in I} p_first[i] = 1;

# path has exactly one end
subject to path_end:
sum{i in I} p_last[i] = 1;
```

APPENDIX C

# Dynamic Program in Python

In this appendix, an implementation of the dynamic program proposed in section 4.4 in Python is presented.

```python
from itertools import chain, combinations, product

inf = float('inf')

def subsets(S):
    """
    Iterator through the subsets of S.

    Arguments:
        S: iterable container of elements.
    Returns:
        iterator of subsets of S.
    """
    S = list(S)
    return (set(s) for s in chain.from_iterable(combinations(S, r)
        for r in range(len(S)+1)))

def solve(x, a, b):
    """
    Solve the simplified punching problem.

    Arguments:
        x: positions of holes.
        a: relative position of left punch.
        b: relative position of right punch.
    Returns:
        minimum number of punching steps to punch all holes.
    """
    x = sorted(x)
    s_ = [-a, 0, b]
    _L, _M, _R = {}, {}, {}
    _index = dict((z, n) for n, z in enumerate(x))

    def index(z):
        """
        Index of a hole position z in x.
        """
        return _index.get(z, -1)

    def holes_range(z, n):
        """
        Range of hole positions >= z and before hole n.
        """
        z_index = index(z)
        if z_index < 0:
            return set()
        return set(x[z_index:n])

    def F(T):
        """
        Minimum number of punching steps to punch T, computed by
            full enumeration.
        """
        return min(len(set(t - s_[f[t_index]] for t_index, t in
           enumerate(T)))
            for f in product([0, 1, 2], repeat=len(T)))

    def A(n, S):
        """
        Minimum steps to punch all holes until n and holes in S.
        """
        if n == -1 and not S:
            return 0
        return min([L(n, S), M(n, S), R(n, S)])

    def L(n, S):
        """
        Minimum steps when punching hole n with the left punch.
        """
        if S and max(S) > x[n] + a + b:
            return inf
        key = (n, frozenset(S))
        if key not in _L:
            if n == -1:
                _L[key] = inf
            else:
                punched = set([x[n], x[n] + a, x[n] + a + b])
                _L[key] = min(1 + A(n-1, S_ - punched) + F(S - S_
                   - punched)
                    for S_ in subsets(s for s in S if n == 0 or s
                       <= x[n-1] + a + b))
        return _L[key]

    def M(n, S):
        """
        Minimum steps when punching hole n with the middle punch.
        """
        if S and max(S) > x[n] + a + b:
            return inf
        key = (n, frozenset(S))
        if key not in _M:
            if n == -1:
                _M[key] = inf
            else:
                punched = set([x[n] - a, x[n], x[n] + b])
                _M[key] = min(min([1 + A(n-1, S_ - punched) + F(S
                   - S_ - punched),
                    L(index(x[n] - a), S_ | holes_range(x[n] - a,
                       n)) + F(S - S_ - punched)
                ]) for S_ in subsets(s for s in S if n == 0 or s
                   <= x[n-1] + a + b))
        return _M[key]

    def R(n, S):
        """
        Minimum steps when punching hole n with the right punch.
        """
        if S and max(S) > x[n] + a + b:
            return inf
        key = (n, frozenset(S))
        if key not in _R:
            if n == -1:
                _R[key] = inf
            else:
                punched = set([x[n] - a - b, x[n] - b, x[n]])
                _R[key] = min(min([1 + A(n-1, S_ - punched) + F(S
                   - S_ - punched),
                    M(index(x[n] - b), S_ | holes_range(x[n] - b,
                       n)) + F(S - S_ - punched),
                    L(index(x[n] - a - b), S_ | holes_range(x[n] -
                        a - b, n)) + F(S - S_ - punched)
                ]) for S_ in subsets(s for s in S if n == 0 or s
                   <= x[n-1] + a + b))
        return _R[key]

    return A(len(x) - 1, set())
```

APPENDIX D

# Selected Solutions

In this appendix, we present a selection of the test cases described in section 7.2 and the solution generated by our algorithm described in chapters 5 and 6 using the machine parameters as given in section 3.4. The number of repetitions of each block of holes in the given patterns is reduced so that they fit on a page.

In Figures D.1 to D.5, different colors denote different punching steps. The pattern of holes is shown at the top. Depicted below are the positions of the punches in each step, where active punches are drawn bold and are connected to their respective punched holes by dotted lines. At the bottom the machine positions are shown again with the distances between consecutive steps.

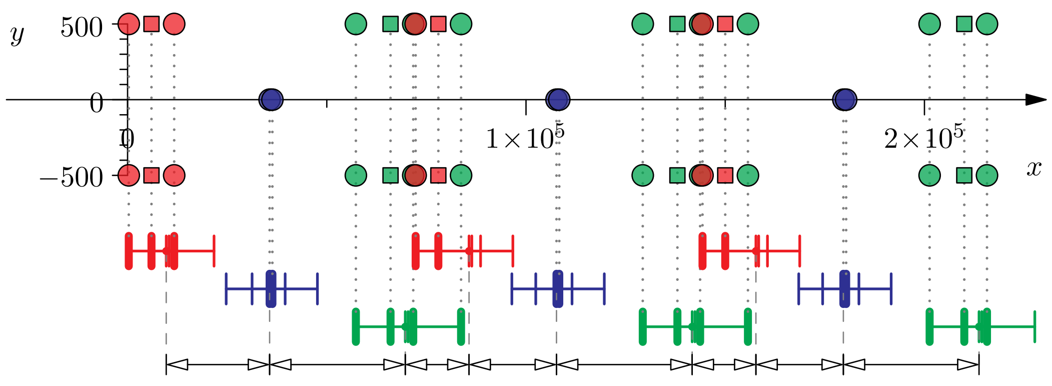


**Figure D.1:** A simple pattern.

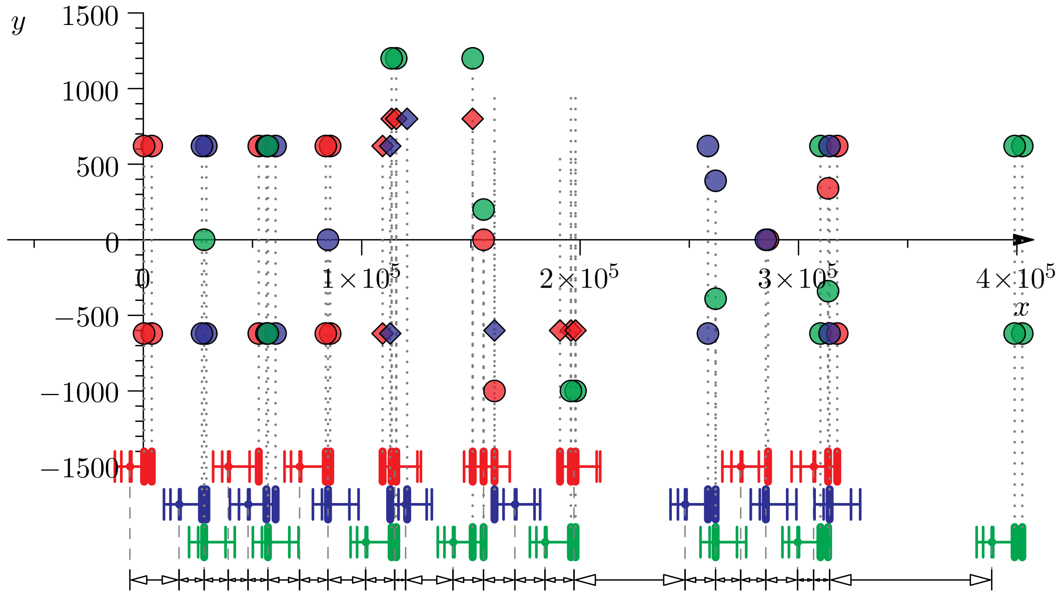


**Figure D.2:** Pattern containing asymmetrical holes with respect to the $x$-axis.

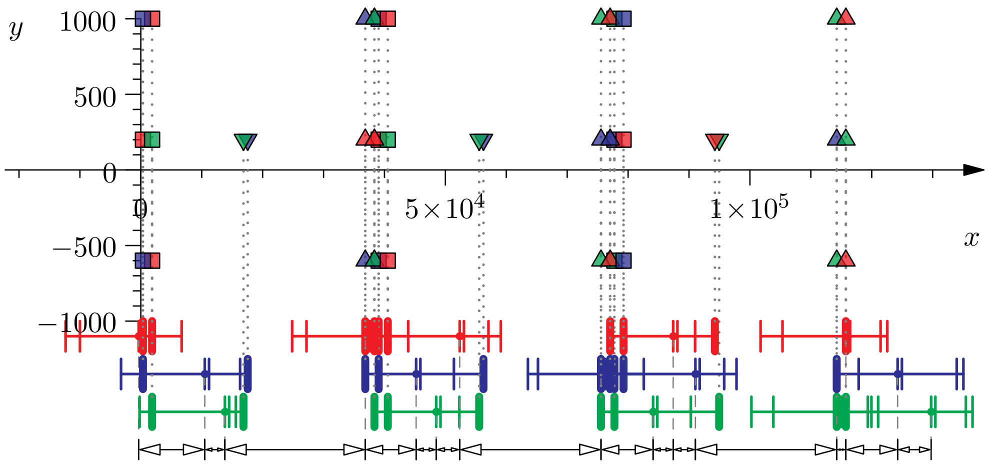


**Figure D.3:** Pattern with symmetry axis above the $x$-axis.

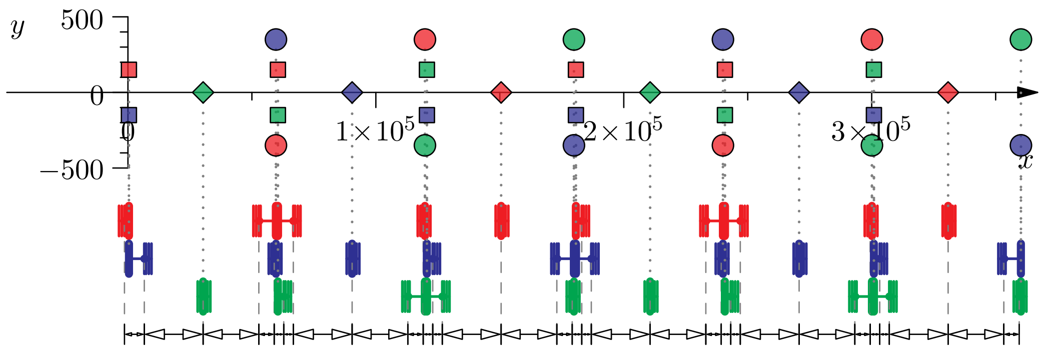


**Figure D.4:** Pattern with three different hole types containing pairs of holes close to each other in $y$-direction.

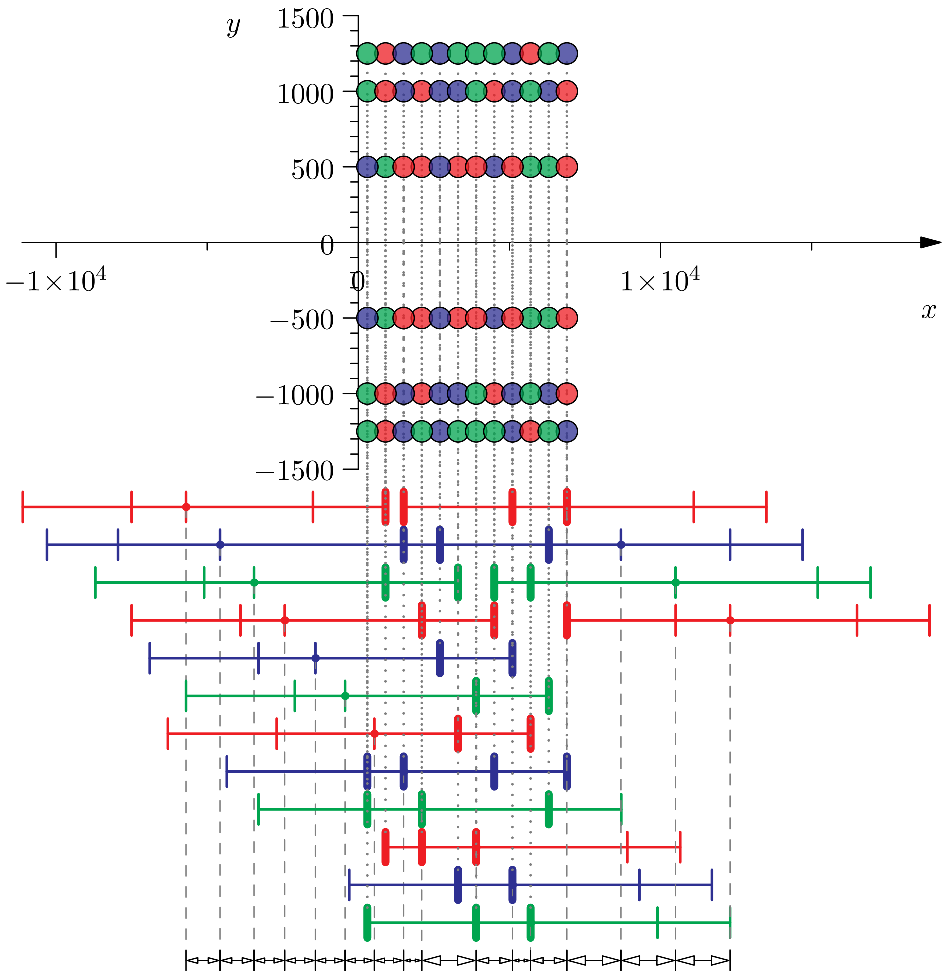


**Figure D.5:** Very dense pattern.

# Bibliography

Achterberg, T. (2009). “SCIP: solving constraint integer programs”. *Mathematical Programming Computation* **1**.1, pp. 1–41.

Alon, N., D. Moshkovitz, and S. Safra (2006). “Algorithmic construction of sets for k-restrictions”. *ACM Transactions on Algorithms* **2**.2, pp. 153–177.

Arrow, K. J., S. Karlin, and H. Scarf (1958). *Studies in the Mathematical Theory of Inventory and Production.* Stanford Mathematical Studies in the Social Sciences. Stanford, CA, USA: Stanford University Press.

Balas, E. (1979). “Disjunctive Programming”. In: *Discrete Optimization II.* Ed. by P. L. Hammer, E. L. Johnson, and B. H. Korte. Vol. 5. Annals of Discrete Mathematics. Elsevier, pp. 3–51.

Belotti, P., L. Liberti, A. Lodi, G. Nannicini, and A. Tramontani (2010). “Disjunctive Inequalities: Applications And Extensions”. In: *Wiley Encyclopedia of Operations Research and Management Science.* Ed. by J. J. Cochran. John Wiley & Sons.

Burer, S. and D. Vandenbussche (2008). “A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations”. *Mathematical Programming* **113** (2), pp. 259–282.

Caprara, A. and M. Fischetti (1997). “Branch-and-Cut Algorithms”. In: *Annotated Bibliographies in Combinatorial Optimization.* John Wiley & Sons, pp. 45–63.

Carvalho, J. M. V. de (1998). “Exact Solution of Cutting Stock Problems Using Column Generation and Branch-and-Bound”. *International Transactions in Operational Research* **5**.1, pp. 35–44.

Chvatal, V. (1979). “A greedy heuristic for the set-covering problem”. *Mathematics of Operations Research* **4**.3, pp. 233–235.

Clausen, J. and M. Perregaard (1996). “On the Best Search Strategy in Parallel Branch-and-Bound – Best-First-Search vs. Lazy Depth-First-Search.” *Annals of Operations Research* **11**, pp. 1–17.

Dakin, R. J. (1965). “A tree-search algorithm for mixed integer programming problems”. *The Computer Journal* **8**.3, pp. 250–255.

Dantzig, G. B., D. R. Fulkerson, and S. Johnson (1954). “Solution of a large-scale traveling-salesman problem”. *Operations Research* **2**, pp. 393–410.

Dantzig, G. B. and J. H. Ramser (1959). “The Truck Dispatching Problem”. *Management Science* **6**.1, pp. 80–91.

Dantzig, G. B. and M. N. Thapa (1997). *Linear Programming: 1: Introduction.* Springer Series in Operations Research. Springer.

Donoghue, W. F. (1969). *Distributions and Fourier transforms.* Vol. 32. Pure and Applied Mathematics. Academic Press.

Feige, U. (1998). “A threshold of ln n for approximating set cover”. *Journal of the ACM* **45**.4, pp. 634–652.

Fortnow, L. (2009). “The status of the P versus NP problem”. *Communications of the ACM* **52**.9, pp. 78–86.

Fourer, R., D. M. Gay, and B. W. Kernighan (2002). *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co.

Gasarch, W. I. (2012). “Guest Column: The Second P =? NP Poll”. *SIGACT News* **43**.2, pp. 53–77.

Gilmore, P. C. and R. E. Gomory (1961). “A Linear Programming Approach to the Cutting-Stock Problem”. *Operations Research* **9**.6, pp. 849–859.

Gilmore, P. C. and R. E. Gomory (1963). “A Linear Programming Approach to the Cutting Stock Problem—Part II”. *Operations Research* **11**.6, pp. 863–888.

Glover, F. and C. McMillan (1986). “The general employee scheduling problem: an integration of MS and AI”. *Computers & Operations Research* **13**.5, pp. 563–573.

Glover, F. and G. A. Kochenberger, eds. (2003). *Handbook of Metaheuristics.* International Series in Operations Research & Management Science. Norwell, MA, USA: Kluwer.

Gomory, R. E. (1958). “Outline of an algorithm for integer solutions to linear programs”. *Bulletin of the American Mathematical Monthly* 64, pp. 275–278.

Gomory, R. E. (1960). “An algorithm for the mixed integer problem”. In: *Research Memorandum, RM-2597, The RAND Corp.* Santa Monica, CA, USA.

Goulimis, C. (1990). “Optimal solutions for the cutting stock problem”. *European Journal of Operational Research* **44**.2, pp. 197–208.

Grötschel, M. and O. Holland (1991). “Solution of large-scale symmetric travelling salesman problems”. *Mathematical Programming, Series A* **51**.2, pp. 141–202.

Grötschel, M., A. Martin, and R. Weismantel (1996). “Packing Steiner trees: a cutting plane algorithm and computational results”. *Mathematical Programming* **72**.2, pp. 125–145.

Gurobi Optimization, Inc. (2012). *Gurobi Optimizer Reference Manual Version 5.0.* Houston, TX, USA.

Hillier, F. S. and G. J. Lieberman (2010). *Introduction to Operations Research.* 9th ed. McGraw-Hill Higher Education.

ILOG CPLEX Division (2007). *CPLEX 11.0 User’s Manual.* Incline Village, NV, USA.

Jünger, M., G. Reinelt, and S. Thienel (1995). “Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization”. *Combinatorial Optimization.* DIMACS Series in Discrete Mathematics and Computer Science **20**.

Karmarkar, N. (1984). “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing.* STOC ’84. New York, NY, USA: ACM, pp. 302–311.

Karp, R. (1972). “Reducibility among combinatorial problems”. In: *Complexity of Computer Computations.* Ed. by R. Miller and J. Thatcher. New York, NY, USA: Plenum Press, pp. 85–103.

Khachiyan, L. G. (1979). “A polynomial algorithm in linear programming”. *Doklady Akademii Nauk SSSR* 244, pp. 1093–1096.

Knuth, D. E. (1976). “Big Omicron and big Omega and big Theta”. *SIGACT News* **8**.2, pp. 18–24.

Knuth, D. E. (1998). *Sorting and Searching.* Vol. 3. The Art of Computer Programming. Reading, MA, USA: Addison-Wesley, pp. 513–558.

Krumke, S., S. Saliba, T. Vredeveld, and S. Westphal (2008). “Approximation algorithms for a vehicle routing problem”. *Mathematical Methods of Operations Research* **68**.2, pp. 333–359.

Land, A. H. and A. G. Doig (1960). “An Automatic Method of Solving Discrete Programming Problems”. *Econometrica* **28**.3, pp. 497–520.

Laporte, G. (1992). “The Vehicle Routing Problem: An overview of exact and approximate algorithms”. *European Journal of Operational Research* **59**.3, pp. 345–358.

Lenstra, H. W. (1983). “Integer Programming in a Fixed Number of Variables”. *Mathematics of Operations Research* **8**, pp. 538–548.

Li, J. and R. R. Rhinehart (1998). “Heuristic random optimization”. *Computers & Chemical Engineering* **22**.3, pp. 427–444.

Lovász, L. (1975). “On the ratio of optimal integral and fractional covers”. *Discrete Mathematics* **13**.4, pp. 383–390.

Lund, C. and M. Yannakakis (1994). “On the hardness of approximating minimization problems”. *Journal of the ACM* **41**.5, pp. 960–981.

Marchand, H., A. Martin, R. Weismantel, and L. Wolsey (2002). “Cutting planes in integer and mixed integer programming”. *Discrete Applied Mathematics* **123**.1-3, pp. 397–446.

Martí, R. and G. Reinelt (2011). “Branch-and-Bound”. In: *The Linear Ordering Problem.* Vol. 175. Applied Mathematical Sciences. Springer Berlin Heidelberg, pp. 85–94.

McGill, R., J. W. Tukey, and W. A. Larsen (1978). “Variations of Box Plots”. *The American Statistician* **32**.1, pp. 12–16.

Mitchell, J. E. (2002). “Branch-and-cut algorithms for combinatorial optimization problems”. In: *Handbook of Applied Optimization.* Oxford, GB: Oxford University Press, pp. 65–77.

Nemhauser, G. L., M. W. Savelsbergh, and G. C. Sigismondi (1994). “MINTO, a Mixed INTeger Optimizer”. *Operations Research Letters* **15**.1, pp. 47–58.

Padberg, M. and G. Rinaldi (1991). “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems”. *SIAM Review* **33**.1, pp. 60–100.

Papadimitriou, C. H. and K. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Pardalos, P. and V. Korotkikh (2003). *Optimization and Industry: New Frontiers.* Vol. 78. Applied Optimization. Springer.

Raghavan, P. and C. D. Tompson (1987). “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”. *Combinatorica* **7**.4, pp. 365–374.

Raz, R. and S. Safra (1997). “A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP”. In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* STOC ’97. New York, NY, USA: ACM, pp. 475–484.

Smith, J. C. and Z. C. Taşkin (2008). “A Tutorial Guide to Mixed-Integer Programming Models and Solution Techniques”. In: *Optimization in Medicine and Biology.* Ed. by G. J. Lim and E. K. Lee. Boca Raton, FL, USA: Taylor and Francis, Auerbach Publications.

Stallman, R. M. and the GCC Developer Community (2008). *Using the GNU Compiler Collection.* Boston, MA, USA.

Turing, A. (1936). “On Computable Numbers, with an Application to the Entscheidungsproblem”. *Proceedings of the London Mathematical Society* **42**, pp. 230–265.

Vazirani, V. V. (2001). *Approximation Algorithms.* New York, NY, USA: Springer.

Wolsey, L. A. (1998). *Integer Programming.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons.

Young, N. E. (1995). “Randomized rounding without solving the linear program”. In: *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms.* SODA ’95. Philadelphia, PA, USA: SIAM, pp. 170–178.