TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

MASTERARBEIT

Reconstructing Borders of Manually Torn Paper Sheets Using Integer Linear Programming

ausgeführt am

Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von

Univ. Prof. Dipl.-Ing. Dr. Günther Raidl
und
Univ. Ass. Mag. Dipl.-Ing. Matthias Prandtstetter

durch

Bakk.techn. Peter Schüller
Veltlinerstraße 3/5/3
A-2353, Guntramsdorf

Oktober 2007

Abstract

This thesis discusses the destruction of sheets of paper by manual tearing and evaluates two approaches for reconstruction of such pages by exact methods based on Integer Linear Programming. These methods operate solely on the shape of torn pieces of paper and reconstruct the borders of the paper sheets.

For evaluation and tests within this problem domain, a tearing model of human paper tearing as well as a C++ library and an XML data interchange format were developed. The tearing simulation takes into regard shear effects which distort paper in a way that—opposed to cutting—the shapes of adjacent pieces will not prefectly fit together anymore.

The ILP formulations were evaluated with extensive empirical tests using the CPLEX solver software. The performance and accuracy of several objective functions was compared, in particular functions using the CPLEX specific *IloAbs* absolute value calculation feature and formulations not using this feature. Further evaluation was performed on the very performance-relevant CPLEX parameters *MIPEmphasis* and on the CPLEX specific *lazy constraints.*

The developed library and tools were very useful for the evaluation in this problem domain and can be reused for further studies. The ILP models are fast and yield good accuracy results on single page instance sizes which makes them well suitable for usage in hybrid algorithms.

Zusammenfassung

Diese Diplomarbeit behandelt die Zerstörung von Papierdokumenten durch manuelles Zerreißen und evaluiert zwei Verfahren zur Rekonstruktion solcher Dokumente basierend auf Methoden der ganzzahligen linearen Programmierung (ILP). Die vorgestellten Verfahren operieren ausschließlich auf der Form der Schnipsel und rekonstruieren die Ränder der zerstörten Dokumente.

Für die Evaluierung wurde ein Modell für manuelles Zerreißen und eine C++ Bibliothek mit einem XML Datenformat entwickelt. Die Simulation des Zerreißprozesses berücksichtigt auch Schereffekte, die Papier beim Reißen—im Gegensatz zum Schneiden—plastisch so verformen, dass die Umrisse benachbarte Stücke nicht mehr exakt zusammenpassen.

Die ILP Formulierungen wurden durch umfangreiche empirische Tests mit der CPLEX Software untersucht. Die Performanz und Ergebnisqualität von mehreren Zielfunktionen wurde verglichen, insbesondere solcher die von der CPLEX-spezifischen *IloAbs*-Absolutwertsberechnung Gebrauch machen und solcher die dies nicht tun. Weiters wurde der sehr performancekritische *MIPEmphasis* Parameter und das *Lazy Constraint* Feature von CPLEX evaluiert.

Die entwickelte Bibliothek und die Programmwerkzeuge haben sich in den Tests bewährt und eignen sich für weiterführende Untersuchungen innerhalb dieser Domäne. Die ILP Formulierungen zeigen gute Performanz und gute Ergebnissgenauigkeit für Instanzen bestehend aus einer Papierseite und eignen sich deshalb für die Verwendung in Hybridalgorithmen.

Contents

1 Preface

This thesis is the result of one and a half years of challenging and rewarding work in very different parts of mathematics and information technology. Here I want to give a short account of the creation and distribute my thanks to all the people who supported me.

The Making Of

In the beginning there was an idea for how to reassemble manually torn paper. The first challenge was to model the creation of problem instances which was based on some empirical experiments I did with work and university colleagues (they found this very amusing). Then I started designing a library to support my further experiments and in parallel thought about how to recreate the pages once I had a framework for solving. After the tools for modelling the whole problem and the solver were working in principle I had to improve the performance of my approach because solving took ages, even for two or three pages. Some experiments later and after throwing quite a lot of brains at specific ILP constraints I had an improved formulation which worked better and could start to design methodical empirical tests which were more than mere experiments on a handful of problem instances. Because of the big amount of instances I created in this process, several bugs in lower layers, especially in the simulation of paper tearing and in geometric libraries, surfaced and had to be fixed. As other people at the institute started to work on the same problem and wanted to use my library, I did a rewrite of the whole code to create cleaner C++ and XML interfaces and to complete the source code documentation. After that the empirical tests started in earnest. At first I tried to solve everything to the optimum or nearly there but after the first test ran for one week and seemed to require another three weeks to finish I decided to test within a shorter reasonable time bound and evaluate the quality of the results as far as they could be optimised even if they were not optimal or far from optimal. The empirical tests had to be repeated once due to bugs in the tearing simulator tool and once due to bugs in the solver tool and took about two weeks in the final version. As the analysed data from the evaluation process would require more pages than the thesis itself I decided not to waste too many trees and to present the results in plots only. The final challenge was to create these plots and I am grateful that several people suggested to use gnuplot and that I decided to use it.

Acknowledgements

First I want to thank the Algorithms and Data Structures group, especially Günther Raidl and Matthias Prandtstetter who expertly guided me through the process of writing this thesis. Many thanks also go to Florian Pflug because during eating a marvellous pizza we discussed a complexity problem of a formulation in this thesis and he had a very good idea which led us to the efficient approach for special constraints described on page 20. For teaching me many software engineering skills I needed for the development of the library and tools in this thesis I want to thank Philipp Tomsich. Very special thanks go

to my girlfriend Edda because she always encouraged me to have no bad conscience to work on my thesis instead of spending my time together with her. I also want to thank my parents—my mother Monika because she taught me to always keep an eye on the big picture of things and never to lose contact to the rest of the world if I was very deep into something technologically—and my father Werner who gave me the fascination of technology before I even went to school.

My final thanks go to the whole open source community, most notably the LaTeX, bibtex, gnuplot, gcc, valgrind, gnumeric, OpenOffice, python, openbox, mozilla, debian and last but not least the Linux Kernel projects which were all directly involved in the creation of this thesis.

2 Introduction

In 1989, the Stasi (main security and intelligence organisation of the German Democratic Republic) destructed thousands of secret documents by manually tearing them into pieces. This resulted in about 17000 bags each containing about 2000 torn document pages. Nowadays, the confidential information contained in this documents is still of great interest. The reconstruction of manually torn papers has similarities to solving jigsaw puzzles, giving the problem the name "Puzzle". This thesis describes a tearing simulation, a C++ library to deal with instances of this problem and two reconstruction approaches.

Manual reconstruction of these torn documents is estimated to take about 100 years, so it is necessary to devise means of reconstruction which rely on computational power to speed up the reconstruction efforts. One can think of several possibilities for solving the Puzzle problem, one approach already taken in [9] is to create an image database of all pieces and do a pairwise pattern matching. After each successful match, the pieces are glued together and the resulting bigger piece is put back into the pool instead of the original parts.

The approach taken in this thesis disregards the image content of the paper snippets and operates on the geometric shape only. Additionally no pairwise matching is done but the whole input data is considered at once and a solution is created in one calculation pass. This is a strategy completely opposite to pattern matching while it is not incompatible to this approach, creating the possibility to combine the two approaches into hybrid algorithms using both complementary approaches to solve instances with a performance superior to each of the single approaches.

Thesis Organisation

Section 3 describes the problem in detail, the next section shows two integer linear programming formulations for solving parts of the Puzzle problem. Section 5 gives a description of the software tools created for evaluating the integer linear program from Section 4 and the two following sections describe the software library SnipLib which was created to implement the tools and how the library was used to implement the tools. Section 8 describes the setup and the results of the empirical tests and the final section contains the conclusions from the tests.

3 Problem Description

In the Puzzle problem—in the following called PP, we are given a set of snippets (small pieces of paper), which properly combined form one or more sheets of document pages. The goal is to find exactly this combination, such that all original sheets of paper are reconstructed. The given snippets might come from shredders for destructing documents or from manually torn documents. Of course, both methods of destructing paper have their specific properties and lead to different challenges in the reassembling process. This thesis will only focus on manually torn documents.

Before anything is destructed, there are several sheets of paper. Most of these pages are rectangles which means that their borders consist of four sheet edges. Figure 1 illustrates a torn sheet of paper. As already mentioned we denote by snippets the given pieces of paper produced by tearing or shredding those sheets of paper. Each snippet has at least three edges, which can be classified into outer snippet edges and inner snippet edges. Inner snippet edges were created by the tearing or shredding process whereas outer snippet edges were part of a sheet edge. This means that after reconstructing the original document, inner snippet edges are "inside" of the sheet. Furthermore we classify inner snippets—bounded by inner snippet edges only—and outer snippets, bounded by at least one outer snippet edge. Corner snippets are a special case of outer snippets because they are bounded by two outer snippet edges which enclose a common angle. Since most of the sheets are rectangular, this enclosed angle will be a right angle but this need not always be the case. By tears we denote those abstract objects which separate two or more snippets by producing inner snippet edges.

Assumptions

The following assumptions are made for the sake of simplicity:

- All sheets which shall be reconstructed have the same paper format which is known as "A4". From this assumption follows the assumption that the size of the sheets to be reconstructed is known in advance.

- For each sheet there exist four corner snippets. As long as this is not true the solver adds small corner snippets to the instance to make the model feasible.

- Corner snippets have exactly two outer edges which need not always be the case in practice, where it could happen that a sheet of paper is torn exactly through the corner or that two corners are part of one snippet. The first case is simplified by adding a new corner snippet with very short edge lengths which replaces the "lost" corner snippet, the second case is averted by the tearing strategy introduced in this thesis which creates four corner snippets as soon as more than one tear is done (see Section 5.1.2).

- Snippets which are not corner snippets contain may only contain one outer edge. This is ensured by the tearing model in this thesis like the previous assumption.

Instance Description

An instance of PP consists of a list of snippets. A snippet is a polygon with coordinates measured in millimetres where each edge is marked either as an inner edge or as an outer edge. Snippets containing no outer edges are called inner snippets and not regarded in the ILP formulations in this thesis. Snippets containing exactly two outer edges are called corner snippets and will often be handled as special cases. References to "next" or "previous" edges or vertices always refer to the next or previous edges or vertices counterclockwise along outer edges of sheets or snippets.

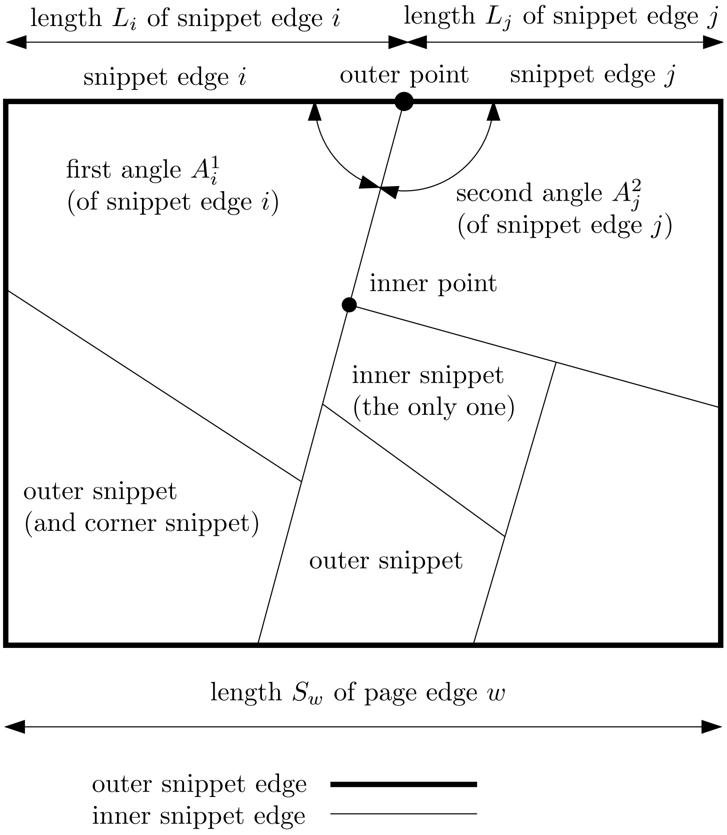


Figure 1: Visualisation of a torn document page.

Figure 1 visualises a torn sheet of paper and names several properties of this sheet. The symbols of the mathematical formulation of the instance are described in the following text and in Table 1. The sheets which are to be reconstructed are numbered from one to m, the sheet edges from one to k. The numbering of the sheet edges is defined to start at the top edge and to be done counterclockwise. In addition to the set of all sheet edge indices $\mathcal{K}$, two additional sets $\mathcal{K}^L$ and $\mathcal{K}^S$ are introduced to represent the indices of all long and short sheet edge indices, respectively. The snippet edges are numbered from one to n, the corner snippets from one to u. Similarly $\mathcal{N}$ and $\mathcal{U}$ denote the sets of all snippet edge indices and the set of all corner snippet indices. For the management of corners, additional constants C_v^1 and C_v^2 are introduced for all $v \in \mathcal{U}$. These constants

Symbol	Explanation
m	number of sheets to be reconstructed
$\mathcal{M}$	set of sheet indices
k	number of sheet edges
$\mathcal{K}$	set of sheet edge indices
$\mathcal{K}^L$	set of all long sheet edge indices
$\mathcal{K}^S$	set of all short sheet edge indices
n	number of outer snippet edges
$\mathcal{N}$	set of outer snippet edge indices
u	number of corner snippets
$\mathcal{U}$	set of corner snippet indices
C_v^1	index of the edge directly before the corner point on corner snippet v. This edge always contains a first angle and never a second angle.
C_v^2	index of the edge directly after the corner point on corner snippet v. This edge always contains a second angle and never a first angle.
$\mathcal{C}^1$	set of all first corner snippet edges: $\mathcal{C}^1 = \{C_v^1 : v \in \mathcal{U}\}$
$\mathcal{C}^2$	set of all second corner snippet edges: $\mathcal{C}^2 = \{C_v^2 : v \in \mathcal{U}\}$
S_w	length of sheet edge with index w.
L_i	length of outer snippet edge with index i.
A_i^1	first angle of outer snippet edge with index i.
A_i^2	second angle of outer snippet edge with index i.

Table 1: Symbols used in the mathematical formulation of PP instances.

contain the first and second edge indices of all corner snippets. Furthermore the sets $\mathcal{C}^1$ and $\mathcal{C}^2$ contain all first and second edge indices of all corner snippets, respectively. The lengths of the sheet edges are named S_w for all $w \in \mathcal{K}$. The lengths of the outer snippet edges are named L_i for all $i \in \mathcal{N}$. The first angles A_i^1 of outer snippet edges are defined for all snippet edges not starting in a corner (for all indices $i \notin \mathcal{C}^2$) and the second angles A_i^2 of outer snippet edges are defined for all snippet edges not ending in a corner (for all indices $i \notin \mathcal{C}^1$).

Special challenges of manual paper tearing

Manual rupture of paper leads to the special challenge of shear effects which cause a tear to have a width greater than zero. Shear effects occur because the fibres of the paper are not all torn at the same position as would be the case had the paper been cut and not torn. Therefore tears are not straight lines, they are ragged and irregular. In addition, shear effects increase the area of paper snippets and a correctly reassembled sheet will contain many overlapping regions located along the tears due to shear effects and the total area of a reassembled pages' snippets will be significantly greater than the area of the original page.

Another challenge where shear effects are involved is that the snippets have to be scanned and subjected to a shape detection process to get a digital representation of the shape. This shape detection process has to create polygons from image data and has to calculate for each edge whether it is a ragged (inner) edge or a straight (outer) edge. The need for a decision of “raggedness” means that there most likely will be a threshold for the minimum polygon segment length. This further means that some short edges will not be detected at all but incorporated into adjacent longer edges, or that short edges will be detected with the wrong edge type (inner versus outer edges).

The above two paragraphs state the two main reasons which make the reconstruction of manually torn sheets of paper a hard problem. This thesis will take into regard both of these implications of shear effects.

4 Integer Linear Programming Formulations

A Integer Linear Program (ILP) is a mathematical formulation of a constraint satisfaction problem which is often combined with an optimisation problem. In this thesis we will always use a special case of an ILP called linear mixed-integer program (MIP) which is defined as:

maximise

$$cx + hy$$

subject to

$$Ax + Gy \leq b$$

$$x \in \mathbb{Z}^n$$

$$y \in \mathbb{R}^p$$

An instance of the problem is specified by the tuple $\langle c, h, A, G, b \rangle$ with c an n-vector, h a p-vector, A an m x n matrix, G an m x p matrix and b an m-vector. This problem is called mixed because of the presence of both integer and continuous (real) variables [12]. A solution $\langle x, y \rangle$ is called feasible if $Ax + Gy \leq b$, infeasible otherwise. It may be the case that there exists no feasible configuration for a given problem instance. A solution is called optimal if it is a feasible solution and if the objective function $cx + hy$ is greater than all objective functions of all other feasible solutions of this problem instance. To solve a minimisation problem it is only necessary to change the sign of the objective function and solve the maximisation problem of the resulting instance. It may be the case that a MIP instance has feasible solutions but no optimal solution—in this case the problem is called unbounded.

The following sections describe two ILP formulations for solving instances of PP. Both formulations assemble outer snippets only, the first formulation is called Lilp and uses the lengths of outer snippet edges, the second formulation—called LAilp—additionally includes the angles between each pair of adjacent outer snippets.

The software used for evaluating the performance of Lilp and LAilp uses the CPLEX solver by ILOG, Inc. version 10.0 with its C++ API. Constraints which might be redundant are simply added to the model as CPLEX will eliminate them in the presolving process. This way it is up to the solver software to discard the more costly constraints and to keep the less costly ones. Some parts of the ILP formulations use the *IloAbs* absolute value calculation feature which is specific to CPLEX and will therefore only be solvable with CPLEX.

4.1 Lilp Formulation

This formulation assigns the outer snippet edges to sheet edges in a way that the sum of the snippet edge lengths fits the lengths of the sheet edges optimally. Inner snippets are not regarded.

The following binary variables are introduced to represent assignments of snippet edges to sheet edges:

$$x_{i,w} = \begin{cases} 1 & \text{if snippet edge } i \text{ is part of sheet edge } w \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$
$$\forall i \in \mathcal{N}, w \in \mathcal{K}$$

The output of a PP instance solved using Lilp is a function which maps each outer snippet edge in the input to a page edge of a certain output page. The number of output pages is defined by the amount of corner snippets in the input. Output pages and page edges are solely defined by the snippet edges assigned to them.

4.1.1 Constraints

The following constraints are defined in Lilp:

- Every snippet edge is part of exactly one sheet edge:

$$\sum_{w \in \mathcal{K}} x_{i,w} = 1 \qquad \forall i \in \mathcal{N} \qquad (2)$$

- Every sheet edge has at least two snippet edges assigned[1]:

$$\sum_{i \in \mathcal{N}} x_{i,w} \geq 2 \qquad \forall w \in \mathcal{K} \qquad (3)$$

- Corners always connect short and long sheet edges. This is formulated in the following constraints:
  Of the two edges of any given corner snippet, exactly one edge has to be assigned to a *long* sheet edge:

$$\sum_{w \in \mathcal{K}^L} \left(x_{C_v^1,w} + x_{C_v^2,w} \right) = 1 \qquad \forall v \in \mathcal{U} \qquad (4)$$

[1] This makes use of the assumption that there are four corner snippets for each torn document page.

Of the two edges of any given corner snippet, exactly one edge has to be assigned to a *short* sheet edge.

$$\sum_{w \in \mathcal{K}^S} \left(x_{C_v^1,w} + x_{C_v^2,w} \right) = 1 \qquad \forall v \in \mathcal{U} \tag{5}$$

- If one corner snippet edge is part of a sheet edge, the other edge of this corner snippet has to be part of the following paper edge on the same sheet:

$$\begin{aligned} x_{C_v^1,w} &= x_{C_v^2,w_{next}} \\ w_{next} &= w - (w \bmod 4) + ((w+1) \bmod 4) \end{aligned} \qquad \forall w \in \mathcal{K} \tag{6}$$

- Every paper edge has to contain exactly one corner snippet's first edge and one corner snippet's second edge[2]:

$$\sum_{i \in \mathcal{C}^1} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{7}$$

$$\sum_{i \in \mathcal{C}^2} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{8}$$

4.1.2 Objective Function

All integer solutions of the polyhedron defined by constraints (2)-(8) are valid solutions to the reconstruction problem. Our goal is to find the solution which corresponds closest to the original document page. Therefore we define an objective function which minimises the difference between the length of a sheet edge and the sum of all outer snippet edges which are assigned to that sheet edge. There are several approaches henceforth called "delta modes" which implement this goal as an integer linear program.

`helpvar` An auxiliary variable Δ_w^{help} models the difference between the length of the sheet edge and the sum of the snippet edges (12). This length is limited by the constant Δ_{Max} which will be explained at the end of this section. A non negative variable Δ_w^L which is put into the objective (9) is constrained to be greater than the absolute value of the auxiliary variable with constraints (10) and (11):

$$minimise \sum_{w \in \mathcal{K}} \Delta_w^L \tag{9}$$

$$\Delta_w^L > \Delta_w^{help} \qquad \forall w \in \mathcal{K} \tag{10}$$

$$\Delta_w^L > -\Delta_w^{help} \qquad \forall w \in \mathcal{K} \tag{11}$$

$$\Delta_w^{help} = S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{12}$$

$$0 \leq \Delta_w^L \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{13}$$

[2] This makes use of the assumption that there are four corner snippets for each torn document page.

`nohelpvar` The absolute value calculation can also be done without an auxiliary variable. In this case there are two constraints (15) and (16) instead of (10), (11) and (12).

$$minimise \sum_{w \in \mathcal{K}} \Delta_w^L \tag{14}$$

$$\Delta_w^L > S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{15}$$

$$\Delta_w^L > -S_w + \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{16}$$

$$0 \leq \Delta_w^L \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{17}$$

`twovars` It is also possible to put the absolute value into the objective with a single constraint (19) and two non negative variables $\Delta_w^{L_{pos}}$ for positive length differences and $\Delta_w^{L_{neg}}$ for negative length differences on each sheet edge. This method allows to weight and limit positive differences differently from negative ones[3]:

$$minimise \sum_{w \in \mathcal{K}} \Delta_w^{L_{pos}} + \Delta_w^{L_{neg}} \tag{18}$$

$$S_w + \Delta_w^{L_{pos}} = \Delta_w^{L_{neg}} + \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{19}$$

$$0 \leq \Delta_w^{L_{pos}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{20}$$

$$0 \leq \Delta_w^{L_{neg}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{21}$$

`posdif` Tearing paper should only make the sum of the snippet edge lengths greater than the real sheet edge length due to shear effects which increase edge lengths. Shear effects occur at each tear and therefore should dominate scanning problems which can decrease edge lengths by not recognising edges. This delta mode therefore only allows a positive length difference by omitting the $\Delta_w^{L_{neg}}$ variables from the previous formulation:

$$minimise \sum_{w \in \mathcal{K}} \Delta_w^L \tag{22}$$

$$S_w + \Delta_w^L = \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{23}$$

$$0 \leq \Delta_w^L \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{24}$$

`iloabs` The natural way to formulate the optimisation is an absolute value function. The CPLEX optimiser provides a method called `IloAbs` to create absolute value

[3] This possibility was not evaluated for reasons of scope but it could improve the practical performance of the model.

functions in the ILP model[4]. This delta mode uses a help variable Δ_k^L to store the absolute value obtained by `IloAbs` in (26) and puts the helper variables into the objective function:

$$minimise \sum \Delta_w^L \tag{25}$$

$$\Delta_w^L = \text{IloAbs}\left(S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w}\right) \qquad \forall w \in \mathcal{K} \tag{26}$$

$$0 \leq \Delta_w^L \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{27}$$

`iloabsdirect` The same CPLEX functionality can be used to put the absolute value expression directly into the objective. This removes the possibility to limit the value because it removes the helper variables but it could allow CPLEX to process more efficiently:

$$minimise \text{ IloAbs}\left(S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w}\right) \qquad \forall w \in \mathcal{K} \tag{28}$$

`limitpagemax` This formulation is different from the formations so far because it does not minimise the sum of all length differences in all pages but it minimises the greatest length difference per page, taking no care of how small other differences on the same page are. Δ_q^L refers to the largest length difference on page q:

$$minimise \sum_{q \in \mathcal{M}} \Delta_q^L \tag{29}$$

$$\Delta_q^L > \quad S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \{4q, 4q+1, 4q+2, 4q+3\}, \forall q \in \mathcal{M} \tag{30}$$

$$\Delta_q^L > -S_w + \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \{4q, 4q+1, 4q+2, 4q+3\}, \forall q \in \mathcal{M} \tag{31}$$

$$0 \leq \Delta_q^L \leq \Delta_{Max} \qquad \forall q \in \mathcal{M} \tag{32}$$

`limitglobalmax` In this formulation we do not minimise the largest length difference on each page but the largest length difference on all page edges of all pages. Therefore only one Δ^L suffices and the objective function consists of one variable:

$$minimise \ \Delta^L \tag{33}$$

$$\Delta^L > \quad S_w - \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{34}$$

$$\Delta^L > -S_w + \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{35}$$

[4] Internally the `IloAbs` feature of CPLEX is implemented using additional auxiliary decision variables and special constraints called “Special Ordered Sets” which enforce that at most two adjacent binary variables in each set are one, the other variables have to be zero.

$$0 \leq \Delta^L \leq \Delta_{Max} \tag{36}$$

Most of the delta modes share the common property that they contain Δ_{Max} which is an upper bound on the maximum length difference for one page edge. The calculation of these limits is done as follows:

- An auxiliary constant "tear width limit" $\Delta_{tearWidth}$ is defined from observations. This limit indicates the maximum expected width of a tear which is created by shear effects and is typically a length between five and ten millimetres.
- The maximum allowed length difference Δ_{Max} is calculated from the tear width limit and the maximum number of tears along a sheet edge, which is assumed to be five. This assumption comes from the observation that in real world instances most times there are only three to four tears along one sheet edge. The tearing simulation in the empirical tests will also take into regard this real world facts. As each of at most five tears can be $\Delta_{tearWidth}$ wide and this applies to both snippets of the tear:

$$\Delta_{Max} = 2 \cdot 5 \cdot \Delta_{tearWidth} = 10 \cdot \Delta_{tearWidth} \tag{37}$$

4.1.3 Complete LILP Formulation

The following is a summary of LILP with the `twovars` delta mode:

minimise

$$\sum_{w \in \mathcal{K}} \Delta_w^{L_{pos}} + \sum_{w \in \mathcal{K}} \Delta_w^{L_{neg}} \tag{38}$$

subject to

$$S_w + \Delta_w^{L_{pos}} = \Delta_w^{L_{neg}} + \sum_{i \in \mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{39}$$

$$\sum_{w \in \mathcal{K}} x_{i,w} = 1 \qquad \forall i \in \mathcal{N} \tag{40}$$

$$\sum_{i \in \mathcal{N}} x_{i,w} \geq 2 \qquad \forall w \in \mathcal{K} \tag{41}$$

$$\sum_{w \in \mathcal{K}^L} \left(x_{C_v^1,w} + x_{C_v^2,w} \right) = 1 \qquad \forall v \in \mathcal{U} \tag{42}$$

$$\sum_{w \in \mathcal{K}^S} \left(x_{C_v^1,w} + x_{C_v^2,w} \right) = 1 \qquad \forall v \in \mathcal{U} \tag{43}$$

$$\sum_{i \in \mathcal{C}^1} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{44}$$

$$\sum_{i \in \mathcal{C}^2} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{45}$$

$$\begin{aligned} &x_{C_v^1,w} = x_{C_v^2,w_{next}} \\ &w_{next} = w - (w \bmod 4) + ((w+1) \bmod 4) \end{aligned} \qquad \forall v \in \mathcal{U}, \forall w \in \mathcal{K} \tag{46}$$

$$0 \leq \Delta_w^{L_{pos}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{47}$$

$$0 \leq \Delta_w^{L_{neg}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{48}$$

4.2 LAilp Formulation

This formulation is an extension of Lilp, it uses the whole Lilp model and extends the constraints and the objective function by defining an order of assigned snippets around each page such that a circle of snippets arises. The optimisation goal here is to find the order of snippets where the angles of adjacent snippets (compare Figure 1) add up to 180 degrees as good as possible. To achieve this, additional binary variables are introduced:

$$y_{i,j} = \begin{cases} 1 & \text{if outer snippet edge } i \text{ is followed by outer snippet edge } j \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{N}, j \in \mathcal{N} \tag{49}$$

The output of a PP instance solved using LAilp is the output from Lilp and a function which maps each outer snippet edge in the input to another outer snippet edge in the input. This cycle of outer snippet edges is defined for each page and defines the order of outer snippets around each document page.

4.2.1 Constraints

The following constraints apply in addition to the Lilp constraints:

- No snippet edge follows itself:

$$y_{i,i} = 0 \qquad \forall i \in \mathcal{N} \tag{50}$$

- Every snippet edge follows exactly one snippet edge and each snippet edge is followed by exactly one snippet edge:

$$\sum_{j \in \mathcal{N}} y_{i,j} = 1 \qquad \forall i \in \mathcal{N} \tag{51}$$

$$\sum_{i \in \mathcal{N}} y_{i,j} = 1 \qquad \forall j \in \mathcal{N} \tag{52}$$

- At each corner snippet, the second snippet edge is the only one following the first one and the first snippet edge does not follow the second one:

$$y_{C_v^1,j} = \begin{cases} 1 & \text{if } j = C_v^2 \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in \mathcal{U}, \forall j \in \mathcal{N} \tag{53}$$

$$y_{C_v^2,C_v^1} = 0 \qquad \forall v \in \mathcal{U} \tag{54}$$

- If a snippet edge j follows another snippet edge i these two snippet edges must be assigned to the same sheet edge unless i is the first edge of a corner snippet or j is the second edge of a corner snippet:

$$y_{i,j} = 1 \Rightarrow x_{i,w} = x_{j,w} \qquad \forall w \in \mathcal{K}, \forall i \in \left(\mathcal{N} \setminus \mathcal{C}^1\right), \forall j \in \left(\mathcal{N} \setminus \mathcal{C}^2\right) \tag{55}$$

  The implementation of these constraints is described in Section 4.2.3.

- In each sheet there must be exactly one circle built from the follow-relations induced by the variables $y_{i,j}$ which have the value one. The implementation of these constraints is described in Section 4.2.4.

4.2.2 Objective Function

The objective function of LILP has to be extended by the additional goal of minimising all differences between 180 degrees and the sum of the angles between two snippet edges i and j where $y_{i,j}$ is one. This goal is multiplied with the constant factor W_{LAILP}. This allows weigthing the LAILP objective relative to the LILP objective in the combined ILP. As in LILP there are different ways of expressing this LAILP goal:

`direct` This delta mode precalculates the absolute values of the angle differences of all valid (i,j) combinations and directly puts them into the objective function. This requires no additional constraints:

$$Objective_{\text{LAILP}} = Objective_{\text{LILP}} + W_{\text{LAILP}} \sum_{\substack{i \in \left(\mathcal{N} \setminus \mathcal{C}^1\right) \\ j \in \left(\mathcal{N} \setminus \mathcal{C}^2\right)}} \left|\pi - A_i^2 - A_j^1\right| \cdot y_{i,j} \tag{56}$$

`limitglobalmax` This method works like the equally named delta mode of LILP: one new variable Δ^A is introduced, added to the objective and constrained to be greater than or equal to the absolute value of any angle difference:

$$Objective_{\text{LAILP}} = Objective_{\text{LILP}} + W_{\text{LAILP}} \Delta^A \tag{57}$$

$$\Delta^A \leq \left|\pi - A_i^2 - A_j^1\right| \cdot y_{i,j} \qquad \forall i \in \left(\mathcal{N} \setminus \mathcal{C}^1\right), \forall j \in \left(\mathcal{N} \setminus \mathcal{C}^2\right) \tag{58}$$

Preliminary tests showed that `direct` creates far better results than `limitglobalmax` so the latter delta mode was not evaluated during extensive tests in Section 8. The value of W_{LAILP} was set to the value 15 which means that 15 degrees angle difference correspond to about 4 millimetres of length difference[5]. As the model created satisfying results with this setting the impact of changing this value was not evaluated in this thesis but it could be the subject of future work.

4.2.3 Implementation Approaches for Follow Constraints

The following constraint was already introduced in Section 4.2.1: If a snippet edge j follows another snippet edge i these two snippet edges must be assigned to the same sheet edge unless i is the first edge of a corner snippet or j is the second edge of a corner snippet:

$$y_{i,j} = 1 \Rightarrow x_{i,w} = x_{j,w} \qquad \forall w \in \mathcal{K}, \forall i \in \left(\mathcal{N} \setminus \mathcal{C}^1\right), \forall j \in \left(\mathcal{N} \setminus \mathcal{C}^2\right) \tag{59}$$

In the following, three approaches for implementation are shown in the order they were developed, the most efficient approach is shown last.

Natural approach

The straight-forward method to implement this is given by the following formula:

$$y_{i,j} - 1 \leq x_{i,w} - x_{j,w} \leq 1 - y_{i,j} \qquad \forall w \in \mathcal{K}, \forall i \in \left(\mathcal{N} \setminus \mathcal{C}^1\right), \forall j \in \left(\mathcal{N} \setminus \mathcal{C}^2\right) \tag{60}$$

This is easier viewed using two cases for values of $y_{i,j}$:

$$y_{i,j} = 0 \Rightarrow -1 \leq x_{i,w} - x_{j,w} \leq 1 \qquad \rightarrow \qquad \text{no restriction on } x_{i,w} \text{ and } x_{j,w} \tag{61}$$

$$y_{i,j} = 1 \Rightarrow \quad 0 \leq x_{i,w} - x_{j,w} \leq 0 \qquad \rightarrow \qquad x_{i,w} = x_{j,w} \tag{62}$$

This exactly realises the conditions in (59). The approach given in (60) needs $O\left(n^2 \cdot k\right)$ constraints and preliminary tests showed that adding all these constraints to the model causes the solver to be very slow.

Prime numbers approach

This approach tries to reduce the amount of constraints necessary for each $y_{i,j}$. The matrix of combinations of x_{i,w_1} and x_{j,w_2} for one $y_{i,j}$ can be visualised as a table where each row represents a x_{i,w_1} for $w_1 \in \mathcal{K}$ and each column represents a x_{j,w_2} for $w_2 \in \mathcal{K}$ (Figure 2). For a given $y_{i,j}$ exactly one of the x_{i,w_1} and one of the x_{j,w_2} decision variables is set to one according to the LILP constraints. Therefore only one cell of this table can be "selected" by the x_{i,w_1} and x_{j,w_2}. If $y_{i,j}$ is one, a cell is "OK" iff $w_1 = w_2$. Figure 2 shows such a table for $w_1, w_2 \in \{1, \ldots, 6\}$ and all allowed and forbidden cells.

[5] This is the `--laaweight` command line parameter of the `solver` described in Secction 5.3.

		x_{j,w_2}					
		$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	$x_{j,5}$	$x_{j,6}$
x_{i,w_1}	$x_{i,1}$	OK	☒	☒	☒	☒	☒
	$x_{i,2}$	☒	OK	☒	☒	☒	☒
	$x_{i,3}$	☒	☒	OK	☒	☒	☒
	$x_{i,4}$	☒	☒	☒	OK	☒	☒
	$x_{i,5}$	☒	☒	☒	☒	OK	☒
	$x_{i,6}$	☒	☒	☒	☒	☒	OK

Figure 2: Constraint table for $y_{i,j} = 1$ and $w_1, w_2 \in \{1, \ldots, 6\}$

The following constraints ensure that *more than half* of all invalid combinations of values for x_{i,w_1} and x_{j,w_2} are rejected:

$$\begin{aligned} & x_{i,1} + x_{i,3} + x_{i,5} + \ldots + x_{j,2} + x_{j,4} + x_{j,6} + \ldots \leq 2 - y_{i,j} \\ & x_{i,2} + x_{i,4} + x_{i,6} + \ldots + x_{j,1} + x_{j,3} + x_{j,5} + \ldots \leq 2 - y_{i,j} \\ & \qquad \forall i \in \left(\mathcal{N} \setminus C^1\right) \quad \forall j \in \left(\mathcal{N} \setminus C^2\right) \end{aligned} \tag{63}$$

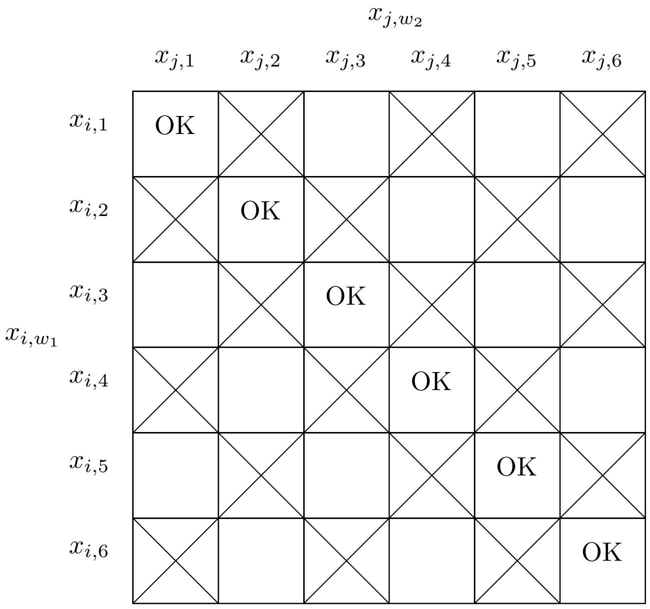
Or in $\sum$ notation:

$$\begin{aligned} & \sum_{w_1 \in \mathcal{K}^L} x_{i,w_1} + \sum_{w_2 \in \mathcal{K}^S} x_{j,w_2} \leq 2 - y_{i,j} \\ & \sum_{w_1 \in \mathcal{K}^S} x_{i,w_1} + \sum_{w_2 \in \mathcal{K}^L} x_{j,w_2} \leq 2 - y_{i,j} \\ & \qquad \forall i \in \left(\mathcal{N} \setminus C^1\right) \quad \forall j \in \left(\mathcal{N} \setminus C^2\right) \end{aligned} \tag{64}$$

Figure 3 shows the cells forbidden by the constraints above. These constraints forbid more than half of the invalid $\langle x_{i,w_1}, x_{j,w_2} \rangle$ combinations using $\mathcal{O}\left(n^2\right)$ constraints while the formulation in (60) needs $O\left(n^2 \cdot k\right)$ constraints to forbid all invalid combinations.

Analogously to (63) it is possible to formulate constraints using the modulus of three:

$$\begin{aligned} & \sum_{w_1 \in \{w : w \in \mathcal{K}, w \bmod 3 = k_1\}} x_{i,w_1} + \sum_{w_2 \in \{w : w \in \mathcal{K}, w \bmod 3 = k_2\}} x_{j,w_2} \leq 2 - y_{i,j} \\ & \qquad \forall i \in (\mathcal{N} \setminus C^1) \quad \forall j \in (\mathcal{N} \setminus C^2) \\ & \qquad \forall (k_1, k_2) \in \{(0,1), (0,2), (1,0), (1,2), (2,0), (2,1)\} \end{aligned} \tag{65}$$



	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	$x_{j,5}$	$x_{j,6}$
$x_{i,1}$	OK	☒		☒		☒
$x_{i,2}$	☒	OK	☒		☒	
$x_{i,3}$		☒	OK	☒		☒
$x_{i,4}$	☒		☒	OK	☒	
$x_{i,5}$		☒		☒	OK	☒
$x_{i,6}$	☒		☒		☒	OK

Figure 3: Cells forbidden by modulus two constraints.

Figure 4 shows the cells forbidden by the modulus three constraints above (six constraints per $y_{i,j}$):

The general formulation of the modulus constraints can be done with prime numbers as follows:

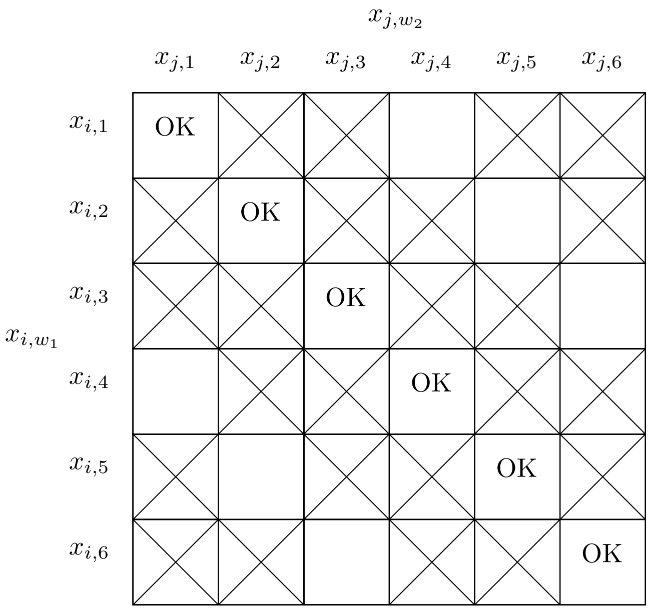
$$\sum_{w_1 \in \{w : w \in \mathcal{K}, w_1 \bmod p = k_1\}} x_{i,w_1} + \sum_{w_2 \in \{w : w \in \mathcal{K}, w_2 \bmod p = k_2\}} x_{j,w_2} \leq 2 - y_{i,j}$$

$$\forall i \in (\mathcal{N} \setminus C^1) \quad \forall j \in (\mathcal{N} \setminus C^2)$$
$$\forall (k_1, k_2) : (k_1, k_2) \in \{0, 1, \ldots, p-1\}^2, k_2 \neq k_1 \qquad (66)$$
$$\forall \text{ primes } p \text{ up to } p_{Limit} \text{ starting at } 2$$

This creates $p(p-1)$ constraints for each prime p, the product of all primes lesser than or equal to p is the maximum value for the number of sheet edges k such that all of the invalid combinations are rejected. Table 2 shows the amount of necessary constraints and primes needed to generate constraints covering several maximum values[6] of k. The table shows that for some maximum values of k this new model needs far more constraints than the previous model. If k is at least 36 the new model always needs less constraints, the greater the numbers the better this model performs compared to the previous model[7]. Although this approach needs fewer constraints than the natural approach preliminary tests revealed that it does not perform significantly better.

[6] For generating constraints, only the maximum values which are an integer multiple of 4 are interesting, because this maximum equals the number of sheet edges $\mathcal{K}$ and each sheet has 4 edges.

[7] $k = 1 \ldots 36$ means that the solver solves a problem with 9 sheets of paper.



		x_{j,w_2}					
		$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	$x_{j,5}$	$x_{j,6}$
x_{i,w_1}	$x_{i,1}$	OK	☒	☒		☒	☒
	$x_{i,2}$	☒	OK	☒	☒		☒
	$x_{i,3}$	☒	☒	OK	☒	☒	
	$x_{i,4}$		☒	☒	OK	☒	☒
	$x_{i,5}$	☒		☒	☒	OK	☒
	$x_{i,6}$	☒	☒		☒	☒	OK

Figure 4: Cells forbidden by modulus three constraints.

		Constraints per $y_{i,j}$	
Primes	Maximum for k	Prime numbers approach	Natural model
2	2	2	2*2 = 4
2, 3	3	2+2*3 = 8	3*2 = 6
2, 3	4	2+2*3 = 8	4*2 = 8
2, 3	5	2+2*3 = 8	5*2 = 10
2, 3	2*3 = 6	2+2*3 = 8	6*2 = 12
2, 3, 5	7	2+2*3+4*5 = 28	7*2 = 14
...	...	...	...
2, 3, 5	14	2+2*3+4*5 = 28	14*2 = 28
2, 3, 5	15	2+2*3+4*5 = 28	15*2 = 30
...	...	...	...
2, 3, 5	2*3*5 = 30	2+2*3+4*5 = 28	30*2 = 60
2, 3, 5, 7	31	2+2*3+4*5+6*7 = 70	31*2 = 62
...	...	...	...
2, 3, 5, 7	35	2+2*3+4*5+6*7 = 70	35*2 = 70
...	...	...	...
2, 3, 5, 7	2*3*5*7 = 210	2+2*3+4*5+6*7 = 70	210*2 = 420
2, 3, 5, 7, 11	211	2+2*3+4*5+6*7+10*11 = 180	211*2 = 422
2, 3, 5, 7, 11	2*3*5*7*11 = 2310	2+2*3+4*5+6*7+10*11 = 180	2310*2 = 4620

Table 2: Prime numbers approach: amounts of primes and constraints.

Efficient approach

This approach further reduces the number of constraints necessary for one $y_{i,j}$. This is possible by reformulating the previous constraints so that the number of constraints is independent of the total number of sheets to be reconstructed. The following constraint is created for each $y_{i,j}$, where the absolute value is realised by the `IloAbs` functionality:

$$\begin{aligned} &|1 \cdot x_{i,1} + 2 \cdot x_{i,2} + \cdots + k \cdot x_{i,k} \\ &-1 \cdot x_{j,1} - 2 \cdot x_{j,2} - \cdots - k \cdot x_{j,k}| \leq k \cdot (1 - y_{i,j}) \\ &\qquad \forall i \in \left(\mathcal{N} \setminus C^1\right) \quad \forall j \in \left(\mathcal{N} \setminus C^2\right) \end{aligned} \tag{67}$$

Or in $\sum$ notation:

$$\left| \sum_{w_1 \in \mathcal{K}} w_1 \cdot x_{i,w_1} - \sum_{w_2 \in \mathcal{K}} w_2 \cdot x_{j,w_2} \right| \leq k \cdot (1 - y_{i,j}) \qquad \forall i \in \left(\mathcal{N} \setminus C^1\right) \quad \forall j \in \left(\mathcal{N} \setminus C^2\right) \tag{68}$$

Exactly one of the x_{i,w_1} is non-zero. The same is true for the x_{j,w_2} in the second sum. This means that the left part of the inequality can never exceed $k - 1$. If $y_{i,j}$ is zero, the right part of the inequality is k and the $x_{i,w}$ in the left side are not constrained. In case $y_{i,j}$ is one, the right part becomes zero and so must be the left part. The left part of the inequality becomes zero exactly if x_{i,w_1} and x_{j,w_2} are one with $w_1 = w_2$ which means that snippet edge i and j are assigned to the same sheet edge $w_1 = w_2$. Therefore these constraints imply all constraints from (59) using only $O\left(n^2\right)$ instead of $O\left(n^2 \cdot k\right)$ constraints in (60) .

In preliminary tests the natural and the prime numbers approaches did not yield any feasible result for instances containing more than two pages within an hour. The efficient approach allows to get solutions for more than four pages within that time, so only this approach was further evaluated in the empirical tests in Section 8.

4.2.4 Implementation Approaches for Anti-Circle Constraints

As already mentioned in Section 4.2.1, there must be exactly one circle of follow relations per sheet given by the variables $y_{i,j}$. The implementation of this assertion is described in this section. To prevent all circles of length one and two, the following constraints are introduced:

$$y_{i,i} = 0 \qquad \forall i \in \mathcal{N} \tag{69}$$

$$y_{i,j} + y_{j,i} \leq 1 \qquad \forall i \in \mathcal{N}, j \in (\mathcal{N} \setminus \{i\}) \tag{70}$$

To prevent all circles of length q, $O\left(n^q\right)$ constraints are needed. It is important, though, not to forbid any valid circles. A circle is valid if it is the only one, even if this circle has length 8 which means that the circle consists of corner snippet edges only. A circle therefore is invalid only if there is another longer circle on the same sheet and elimination of such invalid circles can only be done on demand during the solving process. It is necessary to check for bad circles whenever an integer feasible solution is found and then forbid unwanted circles by adding new constraints into the model. The following strategies are used to implement this anti-circle constraints efficiently:

- Circles of length one and two can be prevented with $O(n)$ and $O(n^2)$ constraints, respectively. Since these are not too many constraints they are simply added.
- Constraints preventing circles of length three are added to the model if specified in a configuration parameter[8], some of them are added as "lazy constraints" which are only evaluated by the solver once an integer feasible solution candidate has been found and added to the model on demand if they are violated. For each circle of length three the impact on the objective is calculated (71) and the circles where this value is above a certain threshold[9] are added lazy, the other circles are added directly into the model. The idea behind this strategy is that the solver will not often create circles which have a very negative impact on the objective so the constraints forbidding these circles can be lazy constraints.

$$\Delta_{Circle_{i_1,i_2,i_3}} = \left|\pi - A^2_{i_1} - A^1_{i_2}\right| + \left|\pi - A^2_{i_2} - A^1_{i_3}\right| + \left|\pi - A^2_{i_3} - A^1_{i_1}\right| \tag{71}$$

- Circles of length four and above are detected and prevented on demand by a callback mechanism once a feasible solution is found: All circles in a sheet are detected and all circles but the largest one are forbidden via new constraints.

4.2.5 Complete LAilp Formulation

The following is a summary of LAilp with the `twovars` delta mode for lengths and the `direct` delta mode for angles:

minimise

$$\sum_{w\in\mathcal{K}} \Delta^{L_{pos}}_w + \sum_{w\in\mathcal{K}} \Delta^{L_{neg}}_w + W_{\text{LAilp}} \sum_{\substack{i\in(\mathcal{N}\setminus\mathcal{C}^1)\\ j\in(\mathcal{N}\setminus\mathcal{C}^2)}} \left|\pi - A^2_i - A^1_j\right| \cdot y_{i,j} \tag{72}$$

subject to

$$S_w + \Delta^{L_{pos}}_w = \Delta^{L_{neg}}_w + \sum_{i\in\mathcal{N}} L_i \cdot x_{i,w} \qquad \forall w \in \mathcal{K} \tag{73}$$

$$\sum_{w\in\mathcal{K}} x_{i,w} = 1 \qquad \forall i \in \mathcal{N} \tag{74}$$

[8] This is the `--laacir3` command line parameter of the `solver` tool described in Section 5.3.

[9] This is the `--laalazy` command line parameter of the `solver` tool. In preliminary experiments a good value for this threshold was found to be 0.15 radians which means about 8 degrees. This parameter was not further evaluated for reasons of scope but could be the subject of future work.

$$\sum_{i\in\mathcal{N}} x_{i,w} \geq 2 \qquad \forall w \in \mathcal{K} \tag{75}$$

$$\sum_{w\in\mathcal{K}^L} \left(x_{C_v^1,w} + x_{C_v^2,w}\right) = 1 \qquad \forall v \in \mathcal{U} \tag{76}$$

$$\sum_{w\in\mathcal{K}^S} \left(x_{C_v^1,w} + x_{C_v^2,w}\right) = 1 \qquad \forall v \in \mathcal{U} \tag{77}$$

$$\sum_{i\in\mathcal{C}^1} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{78}$$

$$\sum_{i\in\mathcal{C}^2} x_{i,w} = 1 \qquad \forall w \in \mathcal{K} \tag{79}$$

$$\sum_{j\in\mathcal{N}} y_{i,j} = 1 \qquad \forall i \in \mathcal{N} \tag{80}$$

$$\sum_{i\in\mathcal{N}} y_{i,j} = 1 \qquad \forall j \in \mathcal{N} \tag{81}$$

$$\left|\sum_{w_1\in\mathcal{K}} w_1 \cdot x_{i,w_1} - \sum_{w_2\in\mathcal{K}} w_2 \cdot x_{j,w_2}\right| \leq k \cdot (1 - y_{i,j}) \qquad \forall i \in \left(\mathcal{N} \setminus C^1\right) \quad \forall j \in \left(\mathcal{N} \setminus C^2\right) \tag{82}$$

$$\begin{aligned} &x_{C_v^1,w} = x_{C_v^2,w_{next}} \\ &w_{next} = w - (w \bmod 4) + ((w+1) \bmod 4) \end{aligned} \qquad \forall v \in \mathcal{U}, \forall w \in \mathcal{K} \tag{83}$$

$$0 \leq \Delta_w^{L_{pos}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{84}$$

$$0 \leq \Delta_w^{L_{neg}} \leq \Delta_{Max} \qquad \forall w \in \mathcal{K} \tag{85}$$

$$y_{C_v^1,j} = \begin{cases} 1 & \text{if } j = C_v^2 \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in \mathcal{U}, \forall j \in \mathcal{N} \tag{86}$$

$$y_{C_v^2,C_v^1} = 0 \qquad \forall v \in \mathcal{U} \tag{87}$$

$$y_{i,i} = 0 \qquad \forall i \in \mathcal{N} \tag{88}$$

$$y_{i,j} + y_{j,i} \leq 1 \qquad \forall i \in \mathcal{N}, j \in (\mathcal{N} \setminus \{i\}) \tag{89}$$

5 Tools

Several tools were developed in this thesis to simulate the whole process of reassembling ruptured paper sheets. The real world process consists of tearing pages into snippets, shuffling several torn pages into a collection, scanning the snippets to get a digital version of the snippets and then using a solver algorithm on the data obtained from the scanning process.

Tearing one page is simulated by the `simulator`. This tool gets a file as input which contains various parameters of the rupture process and the randomisation. The output of the `simulator` contains the parameters as well as a list of tearing actions done and the tearing hierarchy. The tearing hierarchy is a tree structure containing at the root node the initial page, the children of each node are the snippets which were created by tearing the snippet stored in the node. The leaves of the hierarchy are the final snippets which are used as an input to the `mixer` tool. Therefore this hierarchy contains information of the tearing process and the correct solution.

The `mixer` processes one or more output files of the `simulator`. It simulates shear effects on the tear edges of the snippets as well as edge detection problems during the scanning process. The main output of the `mixer` is a file containing the modified snippets in a shuffled list which can subsequently be used as a problem instance. To have the possibility to verify the solutions obtained from the `solver`, the `mixer` is also capable of calculating optimal solutions for each input page. This can be done because the `mixer` has the information which snippet belongs to which page and because the `mixer` has the unmodified snippet data from the `simulator` which contains the exact locations of the snippets in the page. The `mixer` creates one solution file for each input file.

The `solver` processes the problem instance files from the `mixer` and creates a solution file. This solution file has exactly the same format as the solution file from the `mixer` but it contains all pages whereas `mixer` solution files contain only one page each.

The `verifier` tool is able to calculate a quality rating of the solver output in comparison with the optimal solutions calculated by the `mixer`. For visualising tearing hierarchies, generated problem instances and solutions, the `displayer` tool was developed.

Figure 5 shows a flowchart of the whole process when using an instance containing a single document page. Figure 6 shows the `mixer`, `solver` and `verifier` parts of the process for instances containing multiple document pages.

5.1 Simulator

The `simulator`—as its name indicates—simulates the tearing of paper. It uses CGAL [1] for geometric calculations and the CGAL default pseudo random number generator for randomisations. The `simulator` is able to replay tearing processes and to output histograms of randomisation parameters. Figure 7 gives a sample output of a simulation process which was visualised using the `displayer`.

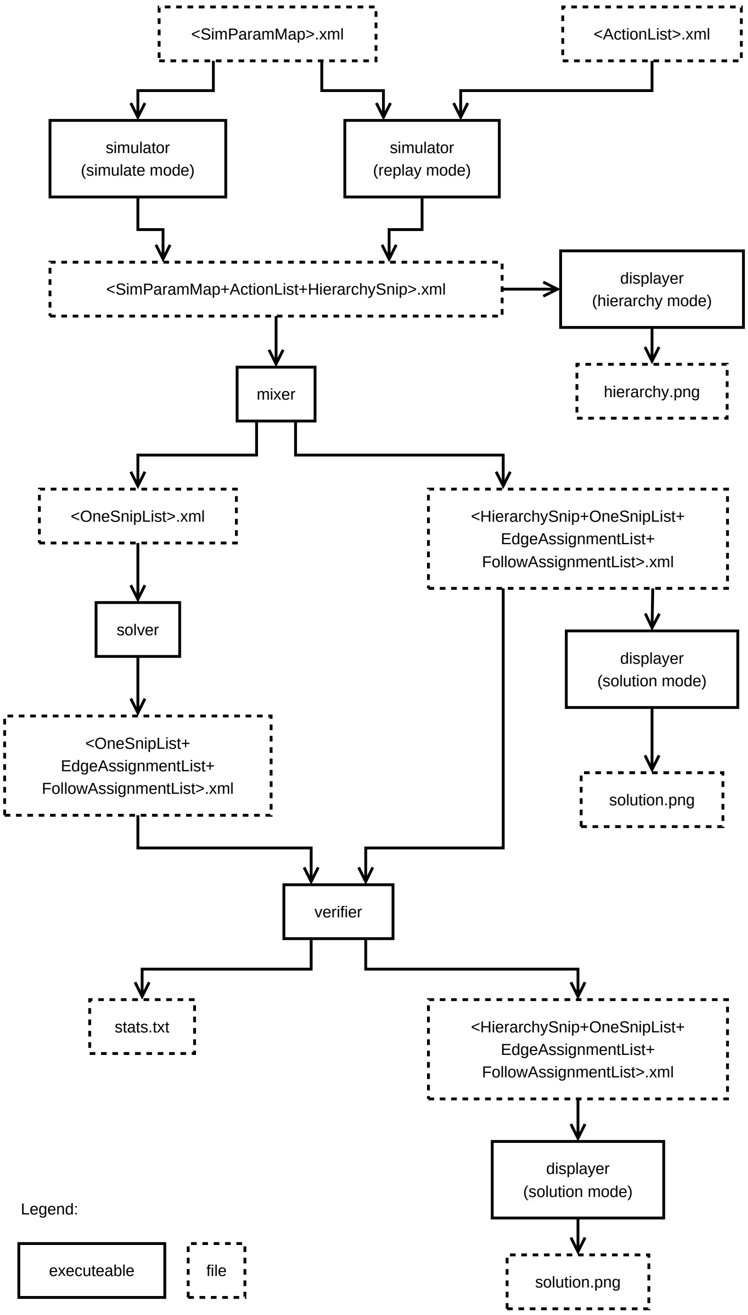


Figure 5: Dataflow diagram for simulated rupture and subsequent reconstruction of a single document page.

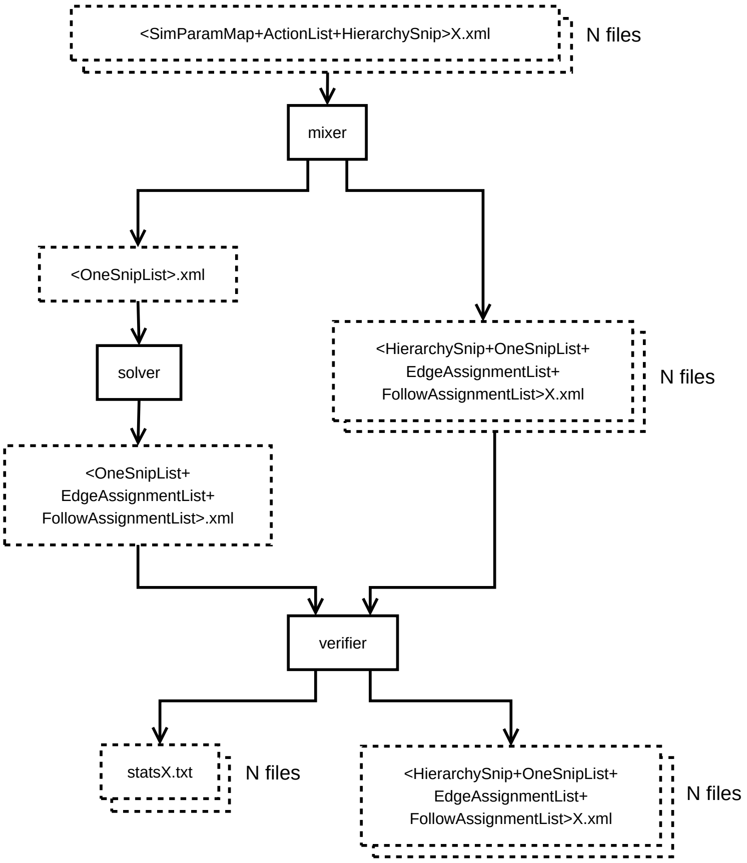

<SimParamMap+ActionList+HierarchySnip>X.xml
N files
mixer
<OneSnipList>.xml
solver
<HierarchySnip+OneSnipList+ EdgeAssignmentList+ FollowAssignmentList>X.xml
N files
<OneSnipList+ EdgeAssignmentList+ FollowAssignmentList>.xml
verifier
statsX.txt
N files
<HierarchySnip+OneSnipList+ EdgeAssignmentList+ FollowAssignmentList>X.xml
N files

Figure 6: Dataflow diagram for simulated rupture and subsequent reconstruction of multiple document pages.

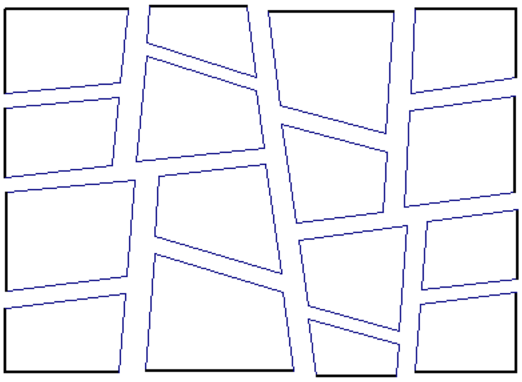


Figure 7: Visualisation of a typical `simulator` output.

5.1.1 Usage

The `simulator` is called using the following command line:

```
simulator -m <mode> -i <infile> [ -hv -p <paramfile> -o <outfile> ]
```

The arguments are:

`-m <mode>` Specify the mode which is one of:

- `simulate` Simulate tearing according to parameters given in a file specified by `-p <paramfile>` and store the snippets and the tearing hierarchy into the file specified by `-o <outfile>`.
- `testrandom` Output histograms for parameters given in a parameter file indicated by `-p <paramfile>`.
- `replay` Replay the tearing protocol in the file specified by `-i <infile>` according to parameters in a file specified by `-p <paramfile>` and write the result to the file specified by `-o <outfile>`. (`<infile>` may be the same as `<paramfile>`). This mode was used to evaluate numerical problems in CGAL by replaying a previously simulated tearing process without new randomisation influence and verifying the output files or tracing problems in CGAL intersection operators. It can also be used to verify the correctness of the `simulator` after a change to the `Geometry` or `Processor` class implementations (see Section 6 for implementation details).

5.1.2 Tearing Process

To create problem instances which resemble real world problem instances as closely as possible, the `simulator` uses a tearing algorithm which resembles observed behaviour of people when tearing paper. The algorithm is implemented by the `Processor` module which is documented in Section 6.2.4. An example for two tears is visualised in Figure 8. Parameters controlling this algorithm are provided to the `simulator` in an XML file. This algorithm works as follows:

- Start with a rectangle centred around the origin representing the original page in landscape format (Figure 8a).
- Simulate a tear represented by a straight line. One end point of this “cut line” is at the top side of the rectangle, the other end point is at the bottom side of the rectangle. The horizontal positions of the two end points are obtained from the *cut.top* and *cut.bottom* parameters, respectively. Using this cut line, the rectangle is torn into two snippets—the snippet to the left of the cut line and the snippet to the right of the cut line. Geometrically this is done by intersecting the rectangle with the cut line (Figure 8b).

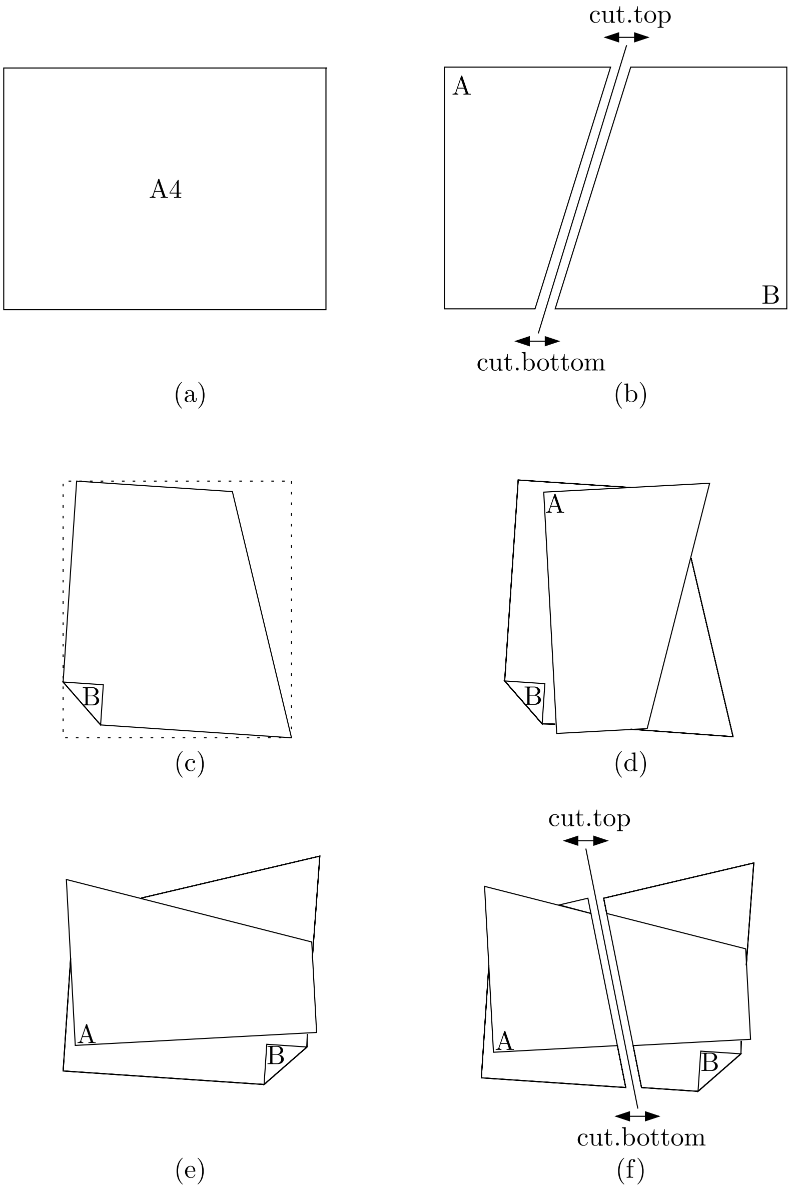


Figure 8: Tearing process visualisation for two tears.

- Place the right resulting part using the following transformations whereas the origin is used as reference point (Figure 8c):
    - center the bounding rectangle[10],
    - mirror as specified by the *mirror.x* and *mirror.y* parameters,
    - rotate as specified by the *rotate* parameter and
    - translate as specified by the *translate.x* and *translate.y* parameters.
- Apply the same transformations to the left part but this time omit the mirroring process, because to mirror one part against the other one it is only necessary to mirror one part. After doing this the two parts are located on top of each other and put together into a stack (Figure 8d).
- Rotate this stack left or right by 90 degrees as specified by the *turn.left* parameter (Figure 8e).

To simulate the desired number of cuts specified by the *cut.count* parameter the last four items of this list are repeated as often as necessary. The last iteration consists of a cut only (Figure 8f).

5.1.3 Parameters

The simulation parameters are specified to the `simulator` in an XML file and are stored into each simulated XML file along with the tearing actions and tearing hierarchy data. A simulation parameter is more than just a value given in an XML configuration file, it is a value generator configured by the XML data.

Two flavours of parameters exist: the parameters giving a boolean output and the parameters giving a numerical output. The boolean output number generator is configured with a given probability to return `true`. For the numerical output number generators there are two implementations: a generator returning an fixed number configurable in the XML and a generator returning a gauss-distributed value where the center and standard deviation can be configured in the XML[11].

Additionally, numerical parameters can be configured with a setting whether the numbers they return should be scaled to given boundary values or not—this setting is called *relation* and is one of *absolute* or *relative*. Whenever a value is retrieved from a parameter, a lower and upper bound is given[12]. If *return* is the returned value and *value* is the value obtained from the number generator, the *absolute* setting limits the value from the number generator at the bounds using the formula

$$return = \min\left(bound_{upper}, \max\left(bound_{lower}, value\right)\right) \tag{90}$$

[10] The bounding rectangle is depicted as a dashed line in Figure 8c

[11] The number generators can easily be extended by another number generator returning specially distributed values.

[12] For example the value of `cut.top` is bounded by the left and right X coordinates of the bounding rectangle enclosing the page or the stack of snippets which shall be torn next.

Type	Value Type	Configuration
bool	boolean	probability
fixed	numerical	value
gauss	numerical	center, dev

Table 3: Parameter types of the `simulator`.

whereas the *relative* setting scales the value retrieved from the number generator in a way that if the number generator returned values in $[-1, 1]$ those values are mapped to $[bound_{lower}, bound_{upper}]$ by the following formula:

$$value' = \frac{bound_{lower} + bound_{upper}}{2} + \frac{value \cdot (bound_{upper} - bound_{lower})}{2} \tag{91}$$

which is afterwards again limited using the formula

$$return = \min\left(bound_{upper}, \max\left(bound_{lower}, value'\right)\right) \tag{92}$$

The following example illustrates the *relative* parameter type: if the `simulator` wants to retrieve a value for the X position of the first tear on the top edge of a page, it asks the `cut.top` parameter for a value in $[X_{\text{top left corner}}, X_{\text{top right corner}}] = [-147.5\text{ mm}, 147.5\text{ mm}]$[13]. If the value generator is configured *relative* and would return 0.5, the cut would start at 73.25 mm. The setting *absolute* would return 0.5 mm. This setting was introduced to model the fact that people start tearing approximately in the middle of the paper sheet or snippets they are holding and the smaller those stacks of paper get the more thorough is the middle of the paper chosen before the tearing is done. Table 3 gives an overview of the available parameter generator types and their configuration. Table 4 shows the concrete parameter names used by the `simulator` along with their units, sensible *relation* settings and example values.

5.2 Mixer

The `mixer` is used to create an unordered collection of polygons created by one or several executions of the `simulator`. The output of the `mixer` is of the format which is expected by the `solver`. Additionally the `mixer` is capable of creating solution files containing perfect solutions. These perfect solution files are used by the `verifier` as a benchmark for solutions created by the `solver`.

To ensure that any solver for finding the original document pages is not biased by the position or sequence of snippets processed, the `mixer` generates a random permutation of the snippets and normalises the position of each snippet by centring it around the origin and rotating it by an arbitrary amount. Figure 9 shows part of a typical `mixer` output.

[13] A landscape A4 page has a length of 295 mm of which the half length is 147.5 mm.

Name	Unit	Relation	Generators	Useful Example value
paper.width	mm	absolute	fixed	295 for A4 landscape.
paper.height	mm	absolute	fixed	210 for A4 landscape.
cut.count	amount	absolute	fixed	4
cut.top	mm	relative	numerical	gauss: center = 0, dev = 0.06
cut.bottom	mm	relative	numerical	gauss: center = 0, dev = 0.18
mirror.x	boolean	-	bool	50% probability
mirror.y	boolean	-	bool	50% probability
rotate	radians	absolute	numerical	gauss: center = 0, dev = 0.08
translate.x	mm	absolute	numerical	gauss: center = 0, dev = 3
translate.y	mm	absolute	numerical	gauss: center = 0, dev = 3
turn.left	boolean	-	bool	50% probability

Table 4: Simulator parameters.

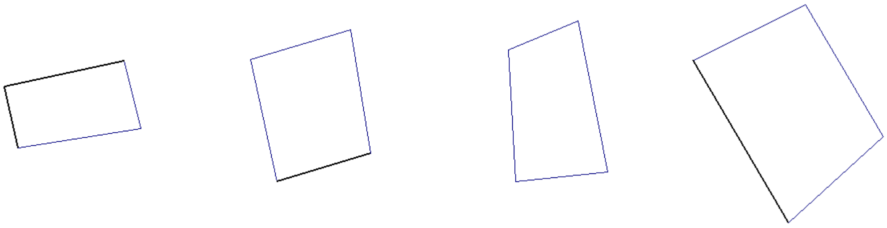


Figure 9: Visualisation of mixer output.

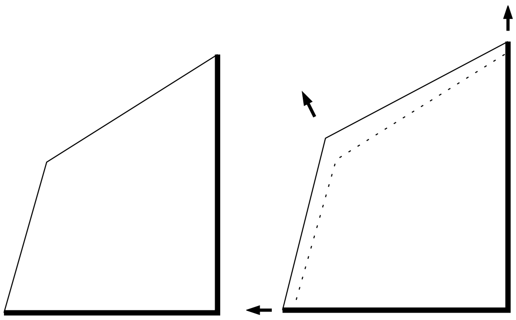


Figure 10: The `mixer` option `-d` visualised on a corner snippet.

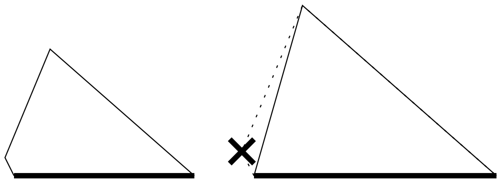


Figure 11: The `mixer` option `-k` visualised on an outer non-corner snippet.

Two important characteristics of the real world paper tearing process are modelled by the `mixer`: shear effects and edge detection problems during the snippet scanning process. Shear effect simulation is visualised in Figure 10, it is modelled by displacing the vertices of the snippets subject to the following rules:

- do not displace corner points,
- displace points along page edges only along those edges[14],
- displace other points away from the centre of the snippet so that the area of the snippet increases.

Edge detection problems during the snippet scanning process are simulated by removing snippet edges shorter than a given minimum length. As visualised in Figure 11, this can reduce the area of a snippet.

The `mixer` is called using the following command line:

```
mixer -o <outfile> [ -hvcrms -d <displace> -k <rmdist> -p <solutionprefix> ]
      <inputpage1.xml> <inputpage2.xml> ...
```

Some useful `mixer` arguments are:

`-c` Center snippets around the origin.

[14] This implicates that only tear edges are moved by this displacement.

-r Randomly rotate snippets.

-m Randomly mirror snippets.

-s Shuffle snippet list before export

-d <displace> Randomly displace snippet points (shear effect simulation). The maximum displacement is given in <displace>.

-k <rmdist> Remove edges shorter than <rmdist> (edge detection problem simulation).

-p <solutionprefix> Activate the solution generator and create solution files as an additional output. Solution files get their name from the given <solutionprefix> and the basename[15] of the respective <input file>, e.g. the input files my/instances/page1.xml and my/instances/page2.xml in combination with -p my/solutions/solution_ would yield the solution files my/solutions/solution_page1.xml and my/solutions/solution_page2.xml.

-o <outfile> Output file for the resulting XML collection.

5.3 Solver

The solver gets a Puzzle problem instance—an unordered collection of polygons—as input[16] and returns a solution file containing this collection plus assignments of outer snippet edges to page edges. If configured for using LAILP, the solver also returns the order of outer snippet edges around each document page in its output file. The solver is called using the following command line:

```
solver -i <infile> -o <outfile> [ -hv -t <tilim> -m <modcon>
  --cplexNames --cplexExportModel --cplexExportFinalModel
  --cplexExportSolution --cplexIncumbentExport <incdir>
  --cplexMipDisplay <dispprio> --cplexMipInterval <dispint>
  --cplexMipEmphasis <emph> --cplexRootAlg <ralg>
  --cplexNodeAlg <nalg> --cplexVarSel <vsel> --cplexBrDir <br>
  --cplexNodeSel <nsel> --cplexProbeTime <prtilim> --cplexBtTol <bttol>
  --laexact --lacir3 --laangles --laeldelta <ldeltamode>
  --laadelta <adeltamode> --laalazy <lazy> --lalweight <lweight>
  --laaweight <aweight> ]
```

[15] The basename of a file name is the last part of the name if the file name includes a path, i.e. the basename of /tmp/directory/input.xml is input.xml

[16] The solver therefore cannot process an output coming directly from the simulator because such an output contains a tearing hierarchy and no snippet list. The mixer has to be used to create an input file for the solver.

Important `solver` arguments are:

`-i <infile>` Specification of the input file which has to be an XML file containing a problem instance.

`-o <outfile>` Specification of the output file which will be an XML file containing the problem instance and its solution.

`-t <tilim>` This specifies the time limit for optimisation in seconds, the default is the CPLEX default of 1e75 seconds, i.e. eternity.

`--cplexNames` Store the names for CPLEX extractables (variables, constraints) in the model (this is very useful for debugging).

`--cplexExportModel` Export the initial CPLEX model (to the file `initialmodel.lp`).

`--cplexExportSolution` Export the CPLEX solution (to the file `solution.sol`).

`--cplexIncumbentExport <incdir>` Export incumbent solutions into the directory specified by `<incdir>`. The filenames are `<incdir>/incumbent_<IDX>_<OBJ>.xml`, created from the index of the incumbent `<IDX>` and the objective value `<OBJ>`.

`--cplexMipEmphasis <emph>` Specify the CPLEX *MIPEmphasis* parameter. The default is the CPLEX default 0, `<emph>` $\in \{0, 1, 2, 3, 4\}$)

`--cplexBrDir <br>` Specify the CPLEX *BrDir* parameter. The default is the CPLEX default 0, `<br>` $\in \{-1, 0, 1\}$)

`--lacir3` Create anti-circle constraints prohibiting circles of three edges following one another. If this option is not set the constraints are created on demand in the incumbent callback as soon as a violation is detected in the incumbent callback. This option can only be used when solving with LAilp.

`--laangles` Solve using LAilp (if this argument is omitted, the `solver` uses Lilp).

`--laeldelta <ldeltamode>` Set the length delta calculation mode to minimise length differences: Possible values: `iloabs`, `iloabsdirect`, `helpvar`, `nohelpvar`, `twovars`, `posdif`, `limitpagemax` and `limitglobalmax`. The mathematical formulations corresponding to these values were described in Section 4.1.2.

`--laadelta <adeltamode>` Set the angles delta calculation mode to minimise angle differences: Possible values are `direct` (this is the default) and `limitglobalmax`. The mathematical formulations corresponding to these values were described in Section 4.2.2.

`--laalazy <lazy>` Specify a threshold for the angle difference above which a constraint is put into the model as a lazy constraint. (default is 0.15 radians). The evaluation of this argument is proposed as future work, in this thesis the default value was used for all tests.

`--lalweight <lweight>` The weight for length differences (how to scale millimetres). The default is 1.0, the evaluation of this parameter is proposed as future work.

`--laaweight <aweight>` The weight for angle differences (how to scale radians). The default is 15.0 which means that a length difference of 5 mm has the same effect on the objective function as an angle difference of 15°. The evaluation of this argument is proposed as future work.

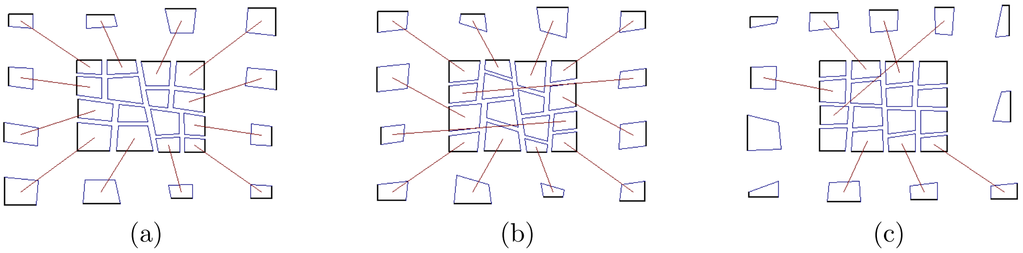
One of the following option combinations has to be specified to select one of the two ILP formulations:

- `--laeldelta <ldeltamode>` to solve using LILP with the specified delta mode,
- `--laangles --laeldelta <ldeltamode>` to solve using LAILP, again the delta mode for lengths has to be specified.

5.4 Verifier

The `verifier` gets a `solver` output file containing one or several solved pages and solution files containing one page each as input. It verifies the `solver` output against the solution files which were created by the `mixer`. The output of the `verifier` is a statistics file and a "verified solution" file for each input solution file. The input and output data flow has been visualised in the diagram in Figures 5 and 6 for one and several input pages, respectively.

A verified solution file has the same structure as a solution file but it contains the hierarchy of an original solution file together with the best match from the solver output assigned to this original solution. This way the `solver` output is combined with the correct solution information and can be drawn using the `displayer` and visually inspected. Figure 12 shows three verified solution files.



(a) (b) (c)

Figure 12: Solution display examples from three separate instances: (a) shows a perfect single-page solution, (b) an partially correct one, and (c) shows one page of a partially correct solution of a two-page instance.

The following command line is expected by the verifier:

```
verifier -d <dir> -s <solverfile> [ -hv -o <statout> ]
  <solution1.xml> <solution2.xml> ...
```

The most important arguments are:

-o <statout> Output file for writing verification statistics (the default is to use standard output).

-d <dir> Specification of directory where verified output files shall be created (XML files containing a hierarchy and an assigned collection of snippets).

-s <solverfile> The solver output file which shall be verified.

Finding the best match between a solver output and several solution files is not a trivial task. The approach taken in this thesis is to find a best match between one of the pages in the solver output and one of the perfect solutions and calculate statistics depending on this best match. Two possibilities to do this were taken into consideration, in the following we will call them “solution-centred” and “solver-centred”.

The solution-centred approach stores all perfect solutions into the verified file and assigns the best matching solver page to each of the solutions. The solver-centred approach stores all pages from the solver output in the verified file and assigns the best matching original page to each page in the solver output. This means that the first approach always stores each solution but sometimes may assign the same page from solver output twice to different solutions which furthermore means that parts of the solver output will not be present in the verified file. Contrarily the whole solver output will be present in the verified file in the second approach, with the possibility that some solution files will not be present because one solution file never was the best match to a page in the solver output.

The verifier implemented in this thesis uses the second approach which ensures that the solver output is always completely present in the verified file. This allows visual inspection of the whole solver output with the possible drawback of missing tearing hierarchies in the verified file.

After assigning the best match, the verifier checks if it is possible to increase the number of correct edge assignments by rotating each page by 180° and each page for which this is the case is stored into the verified solution rotated this way. Algorithm 1 sketches the algorithm used for verification.

5.5 Displayer

The displayer is used to visualise simulated tearing hierarchies, polygon collections and solutions as shown in Figures 7, 9 and 12. This tool supports drawing multiple input

```
Create a global statistics container;
foreach solution file s given as argument to the verifier do
    Find the page p in the solver output which contains the highest amount of snippets in s;
    // Calculations on p are now done in comparison to s.
    Create a data container for a verified file;
    Add the tearing hierarchy of s to the container;
    Calculate the number of correct edge assignments of p;
    if rotating p by 180° increases this number then
        Rotate p by 180°;
    end
    Calculate the edge and follow assignment statistics of p;
    Add this statistics to the global statistics container;
    Add the snippets and the edge and follow assignments of p to the verified file data container;
    Create a name for the verified file using the name of s;
    Write the verified file container to this file name;
end
Write the global statistics container to the statistics file;
```

Algorithm 1: Verification Algorithm

entities in tiles as can be seen in Figure 9. An input entity can be a file—as is the case when drawing solutions—or it can be a snippet in a file—as is the case when drawing problem instances.

The `displayer` is called using the following command line:

```
displayer -m <mode> -o <outfile> [ -hvfa
  -n <tiles> -p <width> -d <drawdepth> -s <stretch> ]
  <input1.xml> <input2.xml> ...
```

The most important parameters are:

`-m <mode>` Specification of display mode. Valid values are `hierarchy`, `collection` and `solution`.

`-n <tiles>` Number of items/tiles displayed in the same row.

`-p <width>` Pixel width of one item/tile output.

`-o <outfile>` Output file name (output is in PNG format).

6 SnipLib

To implement the tools described in the previous section, the SnipLib C++ library was designed and implemented. This library facilitates the implementation of tools in the Puzzle domain. The tasks of creating instances, solving instances and verifying the results are cleanly separated in the tools abstractions provided by the generic SnipLib framework.

An overview of the most important classes, templates and interfaces is given in Figure 13 using the UML [4] notation. The inheritance relations between classes are shown as well as class templates and—drawn as dashed class symbols—instantiated class templates. The UML package symbols are used to show the C++ namespaces in which the classes are located. On top of the figure generic classes can be seen, below is the `SnipLib` namespace containing the core data types and the `XML` and `Tools` namespaces for more specialised data types within SnipLib. In some of the UML diagrams in the following sections, classes which are not part of the discussed module are shown for the sake of completeness (e.g. as base classes). These classes are depicted using a grey background to emphasise the fact that they are not part of the discussed module.

6.1 Generic Modules

Some generic modules were put into the global namespace because they are very useful but not present in the C++ Standard Template Library (STL).

PtrTarget is a class which allows to access objects *only* via reference counted smart pointers from the boost library [6]. It is the universal handle class used in SnipLib. Usage of smart pointers via `PtrTarget` results in less memory leaks. This paradigm also unifies the object management in SnipLib.

Visitor Pattern is a programming pattern for visiting class hierarchies where different actions have to be taken for different classes using RTTI[17]. This pattern consists of the `VisitorBase` class and the `Visitor<T, R, TRef>` and `Visitable<R>` template classes. The code was taken from [3] and adapted to allow visiting `const` objects.

CmdlineConfig provides a generic method to process command line arguments and to store them into a configuration object. The `CmdlineConfig` class is used by deriving from it and defining command line arguments, conversions, consistency conditions and help messages in the derived class.

[17] Run Time Type Identification or Run Time Type Information, a C++ language feature which allows to compare types of objects referenced by pointers of arbitrary type.

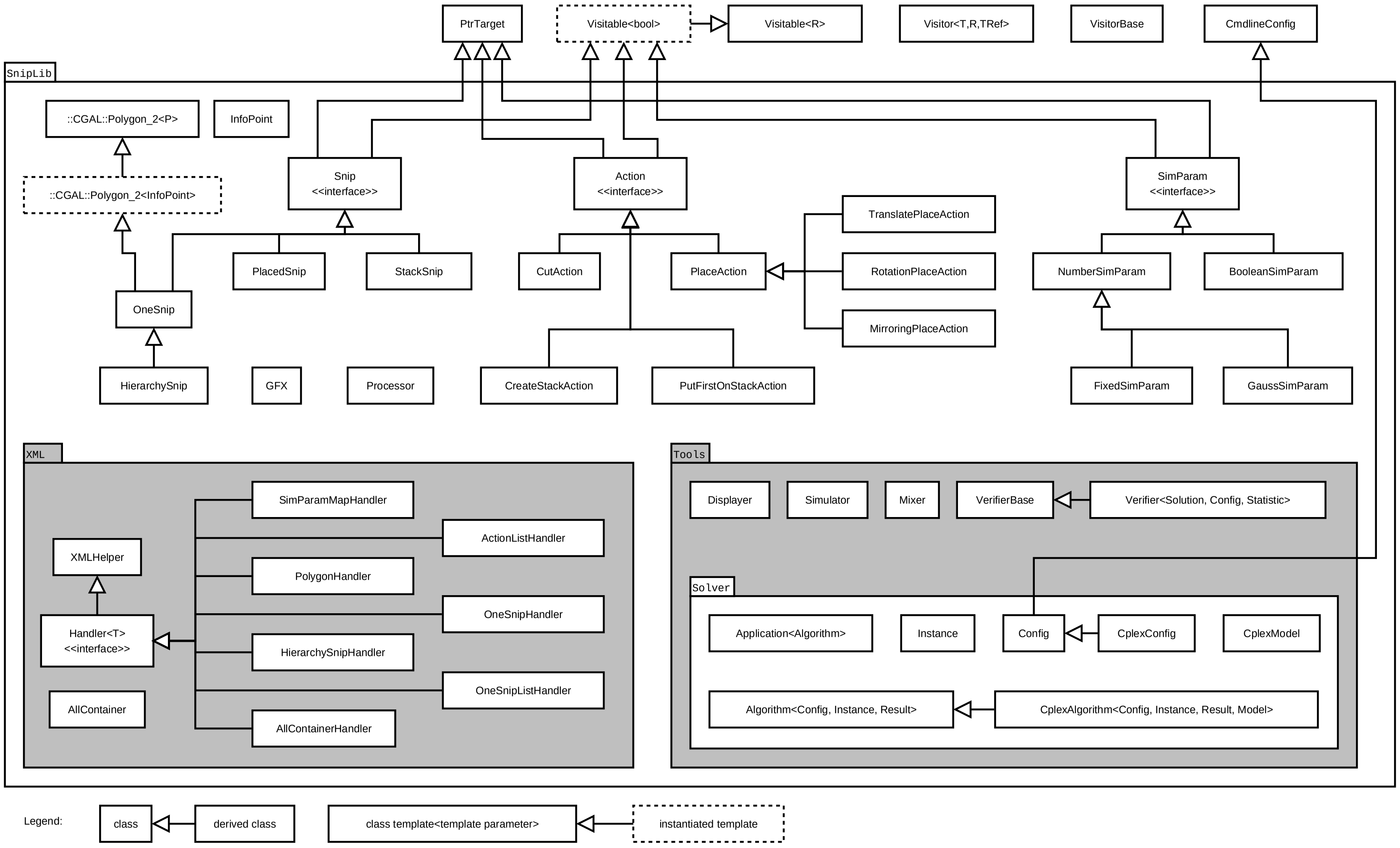


Figure 13: UML diagram: important SnipLib classes and interfaces.

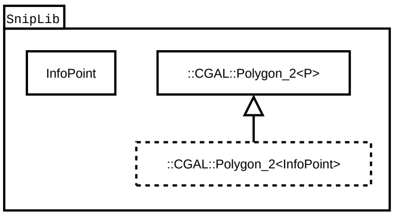


Figure 14: UML diagram: classes in the Geometry module.

6.2 Snippet Processing Modules

This section covers the SnipLib modules which provide functionality for managing snippets, tearing simulation and graphical output. Among other classes, three important class hierarchies are implemented with an abstract interface class on top: `Action`, `Snip` and `SimParam`. These are interfaces to all tearing actions, all snippets and all `simulator` parameters, respectively. All these abstract classes are managed using the `PtrTarget` paradigm and can be visited using the Visitor Pattern.

Geometry provides basic geometric types used for paper tearing. The `InfoPoint` class is derived from the `CGAL::Point_2` class and adds meta information about the `ID` of the point and whether the edge starting at this point is an original page edge or a tear edge. This class is used in connection with the `CGAL::Polygon_2` template to create polygons containing meta information about tearing. Figure 14 gives an overview of the classes in this module.

GFX provides a wrapper around the `GD` library [7] which allows the user to define a drawing canvas with a physical size measured in pixel and a logical size measured in millimetres. This canvas can be used to draw snippets, lines and text using physical and/or logical coordinates.

6.2.1 Action

The `Action` module contains all tearing actions supported by the `Processor` module. The structure of this module is shown in Figure 15. As the `Processor` module documentation (Section 6.2.4) will cover the calculations done with the storage classes in this module, only a brief description of the classes in this module is given here. The following classes are derived from the abstract `Action` interface class:

CutAction which stores a straight cut line,

CreateStackAction which creates a new empty `StackSnip` and contains no data elements,

PutFirstOnStackAction which puts the first `Snip` in the `Processor` snippet list onto the currently active `StackSnip` and contains no data elements and

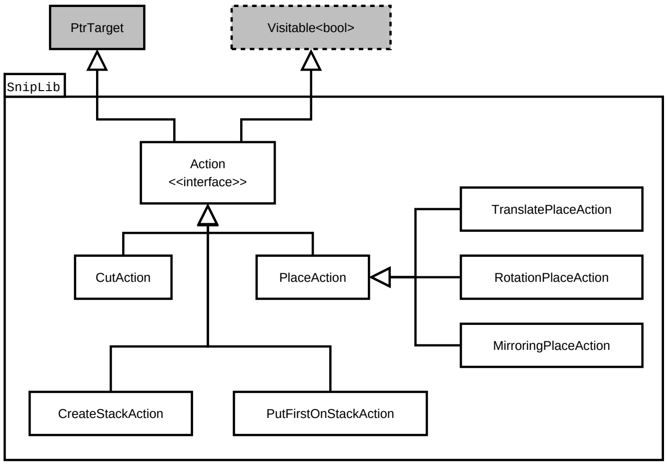


Figure 15: UML diagram: classes in the Action module.

PlaceAction which is an abstract class representing a linear transformation. Concrete subclasses of `PlaceAction` are:

- **TranslationPlaceAction** which stores a `CGAL::Vector` and represents a snippet translation,
- **RotationPlaceAction** which stores an angle and represents a snippet rotation and
- **MirroringPlaceAction** which stores two boolean values and represents mirroring a snippet along the X and/or Y axis.

6.2.2 Snippet

This module contains all classes representing snippets. The generic interface to snippets is the abstract `Snip` class which stores an `ID` and allows to retrieve all polygons contained in the snippet[18] and also allows retrieval of the bounding rectangle of all those polygons. Figure 16 shows the classes defined in this module.

The `OneSnip` class represents one paper snippet and is a single CGAL polygon with edge annotations (inner versus outer edge) in its `InfoPoint` vertex storage elements. Instances of this class can also store a reference to a parent snippet and the information whether this `OneSnip` is located to the left or to the right of the line cutting the parent snippet.

The `HierarchySnip` class represents a `OneSnip` which has been torn into two pieces by the `Processor` and yielded two snippets. Instances of this class store references to their two `OneSnip` children and to the line which has been used to cut. As the children eventually also get torn into several pieces, this class contains a method to obtain all leaf snippets of type `OneSnip` below the current `HierarchySnip` instance.

[18] As explained in the following paragraphs, a `Snip` can contain a polygon or a stack of other `Snip` objects, therefore a `Snip` can contain several polygons which is important to remember when discussing operations on `Snip` objects.

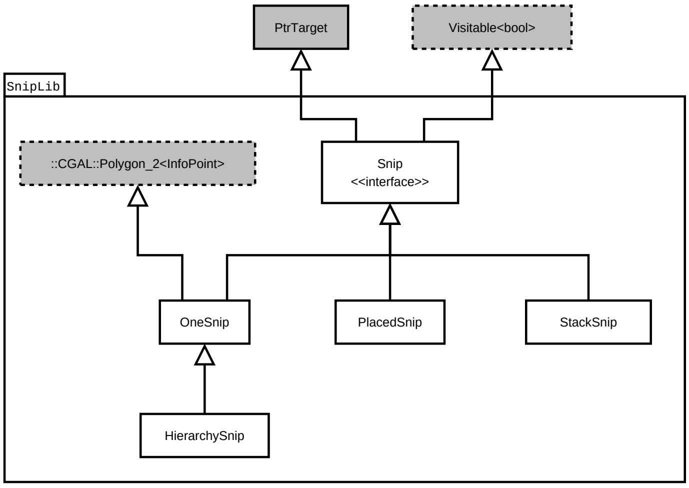


Figure 16: UML diagram: classes in the Snippet module.

The PlacedSnip class represents a Snip which has been placed using a PlaceAction. Similarly, the StackSnip class represents a stack of Snip instances which typically occurs when tearing a paper into pieces and stacking them to tear them again in another direction. Stacking is triggered by an action of type CreateStackAction followed by two actions of type PutFirstOnStackAction in the Processor.

The snippet module also contains a debugging function which can print any Snip to an std::ostream in human readable form. The verbose modes of the simulator outputs this debugging information after each action of the Processor which allows close inspection of the tearing process.

6.2.3 SimParam

This module contains the representations of the simulation parameters which were introduced in Section 5.1.3. The generic interface to simulation parameters is the abstract SimParam class which provides methods to obtain the name, type and relation of a simulation parameter and to obtain values from parameters. Figure 17 shows an UML diagram of this module.

The BooleanSimParam class can only return boolean values and is configurable with a probability to return true, whereas the abstract NumberSimParam class restricts child implementations to returning numeric values. Derived numeric parameter classes are the FixedSimParam class which always returns a preset value and the GaussSimParam class which returns numeric values distributed according to a Gaussian (standard normal) distribution. The numbers are generated using a “Box-Muller normalised distribution generator” [5] .

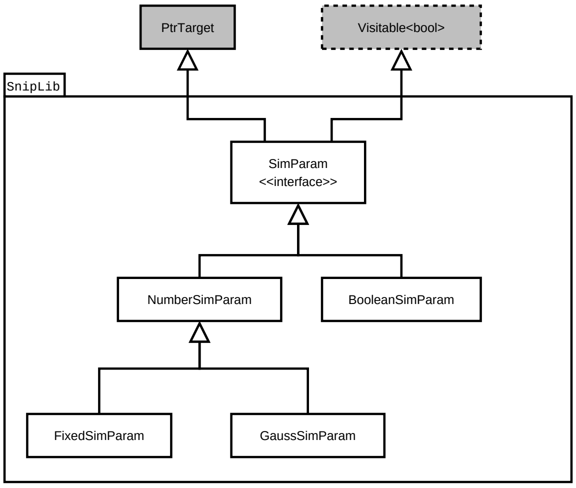


Figure 17: UML diagram: classes in the SimParam module.

6.2.4 Processor

This module implements the `Processor` class which is able to apply `Action` instances to `Snip` instances to simulate the tearing process. The underlying process has already been specified in Section 5.1.2, this section describes the data types which have been created to execute this process in software.

The tearing process is modelled as an algorithm operating on two lists, the input list of actions and the working list of snippets with the initial state of one polygon of A4 paper size stored as the only element of the snippet list. In the following description of all possible actions, the snippet list is referred to as “the List” and a stack of snippets is referred to as “a Stack”. The following actions are sufficient to model the tearing process:

CutAction cuts the only item in the List into two items and put the resulting two items into the List, the item which was created left of the cut line comes into the List first. This action can only be executed with a single item currently in the List.

PlaceAction transforms the first item in the List using the affine transformation returned by the `PlaceAction`.

CreateStackAction appends an empty Stack to the end of the List.

PutFirstOnStackAction removes the first item in the List and puts it onto the Stack which is the last item of the List. To execute this action, the List must contain at least two items and the last item must be a Stack.

6.2.5 XML

This module contains generic XML serialisation base classes and serialiser classes which allow to serialise and deserialise many of the SnipLib types and assists in the creation

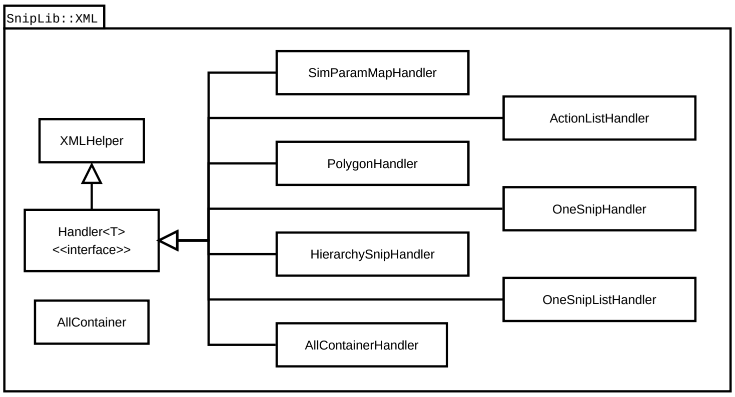


Figure 18: UML diagram: classes in the XML module.

of custom XML serialisation classes for custom solution representations. It is a wrapper around the `libxml2` [14]. Figure 18 shows an UML diagram of the classes in this module. Appendix A shows the file `sniplib.dtd` which is the document type definition for all XML containers.

The following generic classes are defined in this module: `XMLHelper` provides static methods which assist in using `libxml2` and take care of memory allocation and character conversion issues, eliminating the need to explicitly cast values and to explicitly free values returned by `libxml2`. The `Handler<C>` template is derived from XMLHelper and defines an interface for serialisation of the `class C` into an XML node and vice versa. This template contains functionality to write serialised XML nodes to files and to load XML nodes from files. All XML serialisation should be done using a class derived from `Handler<C>`. The following serialisation handler classes for SnipLib types are implemented in this module:

SimParamMapHandler serialises STL maps containing `SimParam` instances indexed by their name. The following example describes the fixed simulation parameter `paper.width`, the gauss distributed parameter `cut.top` and the boolean parameter `mirror.x`:

```
<simparams>
 <param name="paper.width" relation="absolute">
  <fixed value="295" />
 </param>
 <param name="cut.top" relation="relative">
  <gauss center="0" dev="0.05" />
 </param>
 <param name="mirror.x" relation="bool">
  <bool probability="0.5" />
 </param>
</simparams>
```

ActionListHandler serialises STL lists containing `Action` instances. The following is an example for a tearing protocol from the `simulator` containing two tears:

```
<protocol>
 <cutaction paramA="210" paramB="-12.07" paramC="1622.063"/>
 <createstackaction />
 <placetaction translateX="77.61" translateY="-0"/>
 <placeraction rotate="-0.077"/>
 <placetaction translateX="1.7497" translateY="-1.89"/>
 <putfirstonstackaction />
 <placetaction translateX="-69.88" translateY="-0"/>
 <placemaction mirrorX="no" mirrorY="yes"/>
 <placeraction rotate="-0.022"/>
 <placetaction translateX="-0.26" translateY="1.21"/>
 <putfirstonstackaction />
 <placeraction rotate="-1.57"/>
 <cutaction paramA="167.06" paramB="-47.6" paramC="5258.2"/>
</protocol>
```

PolygonHandler serialises polygons holding `InfoPoint` instances. An example is given in the description of the `HierarchySnipHandler`.

OneSnipHandler serialises `OneSnip` instances using the `PolygonHandler`. An example is given in the description of the `HierarchySnipHandler`.

HierarchySnipHandler serialises `HierarchySnip` instances using the `OneSnipHandler`. The following example shows a `HierarchySnip` which is the initial page of size A4. This snippet contains a cut line and parts of the first child which is a `OneSnip` are also shown:

```
<hierarchysnip id="Sn4xEUs4">
 <cutline paramA="210" paramB="-42.2" paramC="3799.71"/>
 <polygon>
  <point x="-147.5" y="-105" startcut="0" id="Sn4xEUp0"/>
  <point x="147.5" y="-105" startcut="0" id="Sn4xEUp1"/>
  <point x="147.5" y="105" startcut="0" id="Sn4xEUp2"/>
  <point x="-147.5" y="105" startcut="0" id="Sn4xEUp3"/>
 </polygon>
 <children>
  <child type="left">
   <onesnip id="Sn4xEUs11">
    <polygon> ...  </polygon>
   </onesnip>
  </child>
  <child type="right"> ...  </child>
 </children>
</hierarchysnip>
```

OneSnipListHandler serialises STL lists containing `OneSnip` instances using the `OneSnipHandler`. The XML of `OneSnipList` classes looks as the following example:

```
<collection>
 <onesnip ... > ... </onesnip>
 <onesnip ... > ... </onesnip>
 ...
</collection>
```

AllHandler serialises the `AllContainer` data type which contains an STL map of `SimParam` instances, an STL list of `Action` instances, a `HierarchySnip` and an STL list of `OneSnip` instances. `AllHandler` directly or indirectly makes use of all the other handler classes described above. The following example shows the structure of a complete SnipLib XML file:

```
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE sniplib SYSTEM "sniplib.dtd">
<sniplib>
 <simparams> ...  </simparams>
 <protocol> ...  </protocol>
 <hierarchysnip id="..."> ...  </hierarchysnip>
 <collection> ...  </collection>
</sniplib>
```

Section 6.4 will show how to reuse the functionality of this module to create a custom solution representation and corresponding XML handlers useable by all tools in SnipLib.

6.3 Tool Modules

This section describes the library modules corresponding to the software tools from Section 5. As the library is solution representation independent, the library provides either a complete tool as in the case of the `simulator` or a tool missing solution representation dependent functionality as in the case of the `mixer` and `displayer`, or no tool at all but a framework for creating a tool as in the case of the `verifier` and `solver`.

The `Simulator` module provides the `simulator` tool which is used to simulate paper tearing and to create problem instances. The `displayer` tool is provided by the `Displayer` module. This tool is able to display collections of snippets and snippet hierarchies which were created using the `simulator` or the `mixer`. The solution display functionality is missing in this module and can be implemented by deriving a custom implementation. The `Mixer` module provides the `mixer` tool which is able to merge multiple simulated torn pages into a collection of snippets which then constitute problem instances and can be

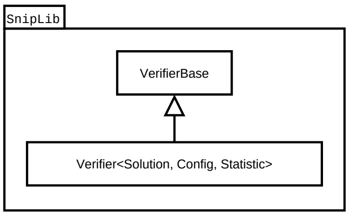


Figure 19: UML diagram: classes in the Verifier module.

used as an input to the `solver`. To generate solution files during the mixing process it is necessary to create a custom implementation of the `mixer` which reuses this module and writes solution files. The usage and further capabilities of the mixer tool are documented in Section 5.2. Information about the implementation of custom `displayer` and `mixer` tools will be given in Section 6.4.

6.3.1 Verifier

This module provides generic functionality to create `verifier` tools which are able to

- verify the consistency of a solver output file,
- verify the solver output file with solution output files generated by a mixer,
- calculate statistics about the correctness of the solution returned by the solver compared to the solutions created by the mixer and
- create "verified solutions" which contain both the tearing hierarchy of the perfect solution and the best matching solver output. This allows easy visual inspection of `solver` results (for an example see Figure 12).

The `VerifierBase` class is a concrete base class defining and implementing the verifier configuration (using `CmdlineConfig`) and a generic statistics gathering module. The `Verifier<SolutionT, ConfigT, StatisticT>` template is derived from the `VerifierBase` and provides a generic framework for reading `solver` output, reading solution files, performing the verification tasks and writing solution output into user-specified locations. Figure 19 shows the classes in the `Verifier` module.

6.3.2 Solver

This module provides a framework to create `solver` tools for Puzzle problem instances. Figure 20 shows the UML diagram of this module. The module is enclosed in the `Solver` namespace and provides the following classes and templates:

Config provides the basic configuration a `solver` needs, and custom implementations should derive their own configuration from this class.

Instance is the representation of a Puzzle problem instance. An object of type `Instance` is created from a list of `OneSnip` objects. This class creates various comfort data structures to make it easier to create a solver algorithm. Instances are passed to the solver algorithm as a `const` variable, they can be used but not modified by the solver algorithm. This reduces the possibility for programming errors while not reducing the performance of accessing every detail of the instance.

This class defines its own `Instance::Snippet` class derived from `OneSnip`. It stores additional information about snippets; the area of the snippet is calculated as well as the zero based index of a snippet in the collection of snippets in the instance (which is an STL vector for performance reasons).

`Instance` also defines a new internal class `Instance::SnippetEdge` which stores information about each inner and outer snippet edge like their ID, index, previous and next edges' indices, the length of the edge and whether the edge is a page or a tear edge.

Finally, the `Instance` class also calculates the set of all indices of snippets which are corner snippets.

Algorithm<ConfigT, InstanceT, ResultT> is the template which is the base class for all custom `solver` implementations. An `Algorithm` receives configuration and instance objects of type `const ConfigT&` and `const InstanceT&` and contains three abstract methods which have to be overridden:

solve This method has to solve the given instance and return true upon success and false otherwise.

getResult This method shall return the solution representation obtained by a previous run of `solve`. The solution representation type is defined by the `ResultT` template parameter.

outputResult This method has to be overridden to store the solution into a file. Most times this method will simply call the appropriate XML serialiser.

Application<AlgorithmT> is a template configured with the solver algorithm which shall be used for solving. This template is a wrapper around the algorithm providing management functionality and facilitating the easy creation of a `solver` binary.

The concept of separating the application and the algorithm was created to allow algorithms to use other algorithms as it is done by the `WrapperAlgorithm` template in the LAilp implementation described in Section 7.

6.3.3 CplexSolver

This module provides a generic interface for creating `solver` tools using the CPLEX optimiser and is a specialisation of the `Solver` module. Figure 21 shows the classes in this module and how they are derived from classes in the `Solver` module:

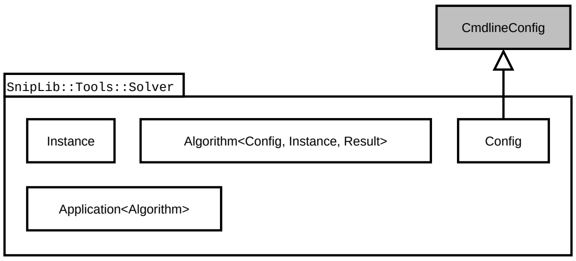


Figure 20: UML diagram: classes in the Solver module.

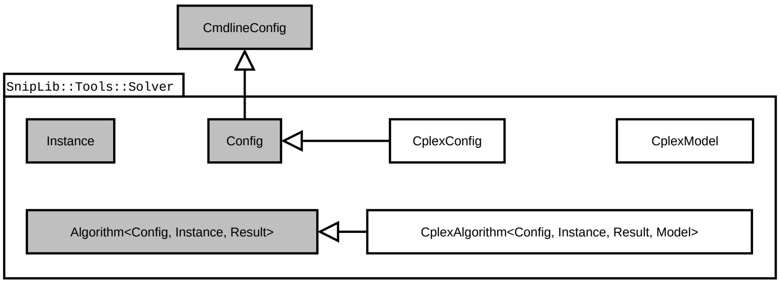

CmdlineConfig
SnipLib::Tools::Solver
Instance
Config
CplexConfig
CplexModel
Algorithm<Config, Instance, Result>
CplexAlgorithm<Config, Instance, Result, Model>

Figure 21: UML diagram: classes in the CplexSolver module.

CplexConfig is a class which allows to configure basic CPLEX parameters in addition to generic `solver` parameters. The parameters which are directly mapped to CPLEX parameters have the CPLEX default values as defaults.

CplexModel holds basic data structures of the CPLEX model. It controls access to the objective expression and is able to handle user cuts and lazy constraints.

CplexAlgorithm<ConfigT, InstanceT, ResultT, ModelT> is the specialisation of the `Algorithm` template and provides a framework to create a solver algorithm class using an ILP formulation which is solved with CPLEX. The following important methods have to be provided by custom implementations deriving from this template:

createModel creates the CPLEX model.

configureCplex configures the CPLEX solver. This method must register the very important incumbent callback. Other tasks this method may perform are registering other callbacks, setting parameters, setting a starting solution and setting branching priorities.

Solutions of the solving process are retrieved via incumbent callbacks. The `CplexAlgorithm` internally provides the `CplexIncumbentI` class—an abstract incumbent callback class—which has to be used as a base class for a custom incumbent callback. This callback has to be registered in the `configureCplex` call. By using the incumbent callback to retrieve solutions, this module is able to create incumbent solution output without additional effort by the user of SnipLib.

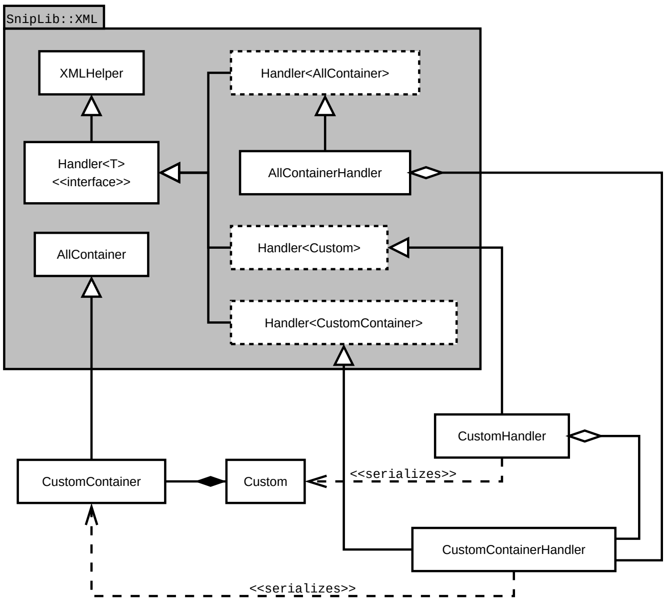


Figure 22: UML diagram: custom XML serialisation using the XML module.

6.4 Reuse

Many modules and classes in SnipLib facilitate reuse when working with SnipLib. This section gives an overview of the classes which are very likely to be reused and how this should be done.

All of the generic modules—as their name already suggests—are very likely to be reused. Especially the Visitor Pattern is very helpful in creating drawing functionality for new solver strategies. The `CmdlineConfig` will be reused indirectly every time a `Config` class of a generic tool is reused.

Whenever a new solving algorithm is implemented it will be necessary to create new XML processing classes derived from the `XML::Handler<C>` template class. Suppose the new formulation is called "Custom" and the type of the new data is therefore `Custom`. Then a new "`CustomContainer`" data type should be derived from the `XML::AllContainer`, containing `Custom`. A handler class has to be derived from `XML::Handler<Custom>`, assume to call it "`CustomHandler`". Finally, a handler for the new `CustomContainer` has to be implemented. This custom handler uses `CustomHandler` to serialise the custom data and `XML::AllHandler` to serialise the base class (`XML::AllContainer`) data. Figure 22 shows an UML diagram of this example.

The `Displayer` module is designed for reuse and adapted for drawing custom solution representations by deriving from the internal `DrawItem` class and providing a custom `DrawVisitor`. This visitor has to be able to draw the custom `DrawItem` containing a solution representation read from an XML file.

Custom implementations of the `mixer` should reuse the `Mixer` class and override the `createAndOutputSolution` method to allow usage of the `-p` option to generate a solution file for each input file which then can be used to verify solver outputs. A `mixer` without a custom `createAndOutputSolution` method throws an exception and aborts when usage of the `-p` option is attempted.

The `Verifier` module must be reused to create a `verifier` tool working with a custom solution representation. This is done by deriving from the `Verifier<SolutionT, ConfigT, StatisticT>` template.

If there is a need for new simulation parameter random number generators, a new `SimParam` class can be derived from the `BooleanSimParam` or `NumberSimParam` class.

As the `Solver` and `CplexSolver` modules cannot be used without deriving from them, their reuse has already been described in Sections 6.3.2 and 6.3.3, respectively.

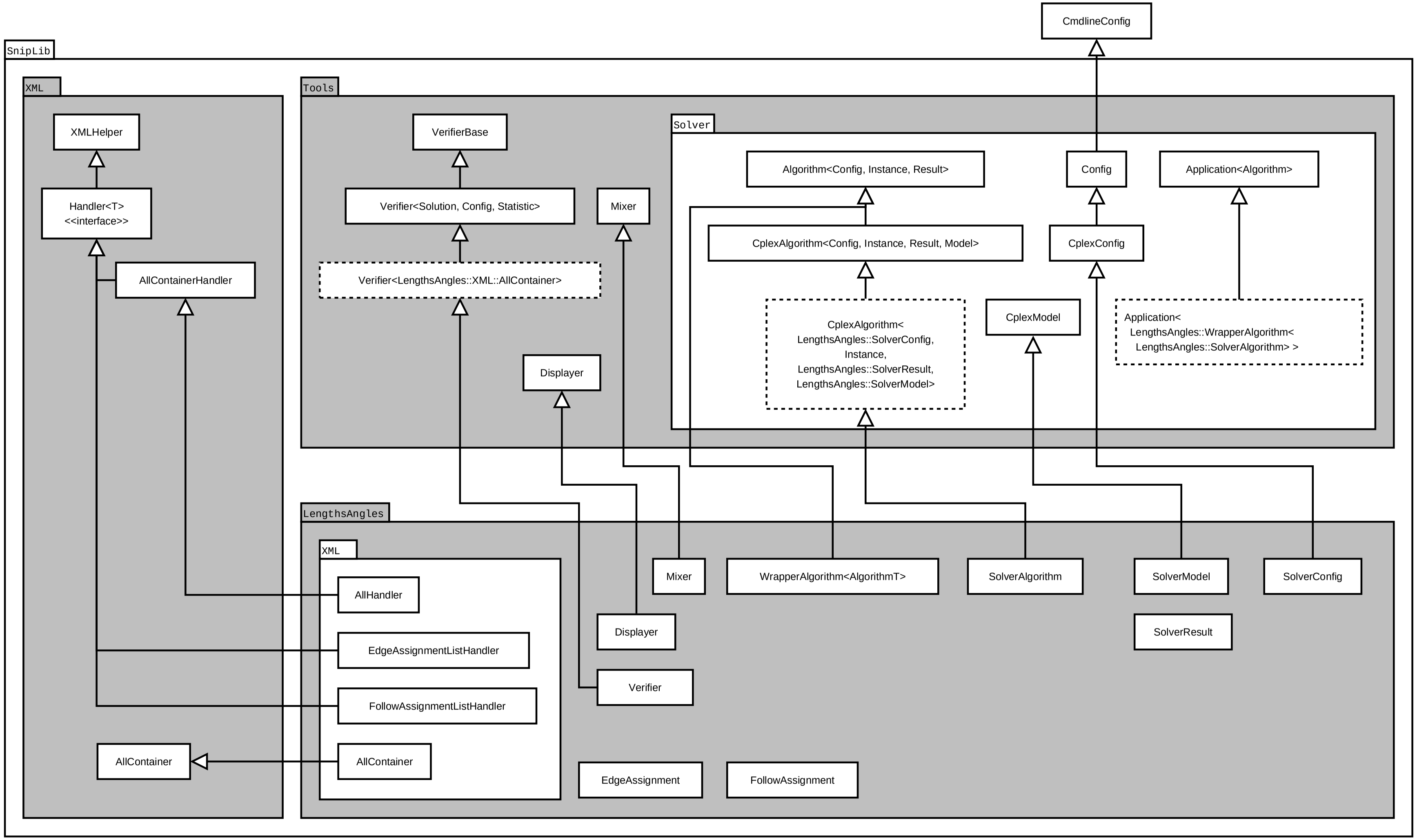

CmdlineConfig
SnipLib
XML
XMLHelper
Handler<T>
<<interface>>
AllContainerHandler
AllContainer
Tools
VerifierBase
Verifier<Solution, Config, Statistic>
Mixer
Verifier<LengthsAngles::XML::AllContainer>
Displayer
Solver
Algorithm<Config, Instance, Result>
Config
Application<Algorithm>
CplexAlgorithm<Config, Instance, Result, Model>
CplexConfig
CplexAlgorithm<
LengthsAngles::SolverConfig,
Instance,
LengthsAngles::SolverResult,
LengthsAngles::SolverModel>
CplexModel
Application<
LengthsAngles::WrapperAlgorithm<
LengthsAngles::SolverAlgorithm> >
LengthsAngles
XML
AllHandler
EdgeAssignmentListHandler
FollowAssignmentListHandler
AllContainer
Mixer
WrapperAlgorithm<AlgorithmT>
SolverAlgorithm
SolverModel
SolverConfig
Displayer
SolverResult
Verifier
EdgeAssignment
FollowAssignment

Figure 23: UML diagram: Implementation of the LAilp tools.

7 Implementation of both ILP formulations using SnipLib

This section describes how SnipLib is used to implement tools to test the Lilp and LAilp formulations. As Lilp is part of LAilp only one set of tools is implemented for which it is possible to activate either Lilp or LAilp in its calculations. These tools will be called the “LAilp tools” in the following. Figure 23 shows an UML diagram of the most important classes implemented including the classes of SnipLib which are directly reused. Data types and implementation details of the LAilp tools are given in the following sections.

7.1 Storage Classes

The following classes are defined for storage purposes:

EdgeAssignment stores assignments of snippet edges to page edges, corresponding to variables $x_{i,w}$ in Lilp and LAilp.

FollowAssignment stores assignments of snippets to following snippets, corresponding to variables $y_{i,j}$ in LAilp.

XML::AllContainer extends the `SnipLib::XML::AllContainer` data type and adds the possibility to store lists of edge and follow assignments and mirror assignments.

SolverResult stores edge and follow assignments and is used to extract solver results from the CPLEX model. This data type takes the place of the `ResultT` template parameter of the `CplexAlgorithm<ConfigT, InstanceT, ResultT, ModelT>` template and is used for implementing the `SolverAlgorithm`.

To serialise lists containing objects of type `EdgeAssignment` and `FollowAssignment` or to serialise an `XML::AllContainer`, the XML handler classes `XML::FollowAssignmentListHandler`, `XML::EdgeAssignmentListHandler` and `XML::AllHandler` are implemented.

7.2 Tool Implementation Classes

The following describes the LAilp tools which are derived from SnipLib tools or implemented using SnipLib tool templates.

Mixer is derived from the SnipLib `Mixer` class and extended to allow the generation of perfect solution files. This is done using an algorithm which calculates the perfect edge and follow assignments from the coordinates of the vertices *before* the `mixer` changes them to create the `solver` input instance. Although mirrored problem instances are not solved in this thesis, the `mixer` supports them and also generates solution information including mirroring information.

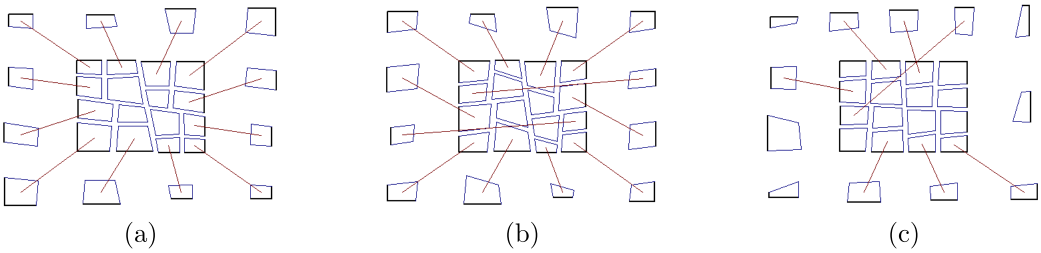


Figure 24: Solution display examples.

Displayer is derived from the SnipLib `Displayer` class and extended to be able to display solution information. Solutions are drawn as can be seen in the examples given in figure 24. Snippets assigned to page edges are drawn around the hierarchy in the centre of the canvas. If an assigned snippet really belongs to the page it is connected to the snippet in the solution by a line. Solution 24a is a perfectly correct solution, 24b contains snippets assigned to wrong edges and wrong follow assignments and 24c contains snippets from other pages (detectable by the missing connection line), snippets assigned to wrong edges of the page and wrong follow assignments.

Verifier is derived from the SnipLib `Verifier<SolutionT, ConfigT, StatisticT>` template instantiated as `Verifier<LengthsAngles::XML::AllContainer>` using the defaults for the `Config` and `Statistic` template parameters. The main effort in the implementation of this tool was the algorithm for finding the best match between the solver output pages to the solution pages (see Algorithm 1 in Section 5.4).

Solver implements the Lilp- and LAilp-based `solver` tool. This is done using two classes derived from the `Algorithm` template; the `WrapperAlgorithm< AlgorithmT>` template and the `SolverAlgorithm`. The configuration class is `SolverConfig` which is derived from `CplexConfig`.

The `WrapperAlgorithm<AlgorithmT>` is a simple algorithm which takes the problem instance and creates a modified problem instance by adding as many very small corner snippets as necessary so that the number of corner snippets in the whole instance becomes an integer multiple of four[19]. The modified instance is passed to the algorithm given in the template parameter.

The `SolverAlgorithm` is a `CplexAlgorithm` template which uses the custom `SolverConfig` for configuration, the default `Instance` as an instance data type, the custom `SolverResult` class as result representation and the custom `SolverModel` as CPLEX model container.

By default the solver creates a model containing the Lilp formulation. Using the `--laangles` command line switch causes the LAilp formulation to be generated in the model so that this `solver` can be used for solving both formulations.

[19] This functionality is required by the assumptions made in Section 3.

8 Tests and Results

Within this section, test settings and results for LILP and LAILP are presented. The tests were performed using the tools explained in previous sections which were implemented using SNIPLIB. The test server was a machine with two Dual Core AMD Opteron(tm) 270 Processors with 2 GHz and 8 GB RAM in total with CPLEX version 10.0.

Instances for testing were created using the `simulator` and `mixer` tools. Each instance consists of the simulated pages, the `solver` input file and the solution files created by the `mixer` which are needed for verification of the `solver` output.

For each tested parameter combination, ten instances were generated and each test was run on all instances. In the evaluation of the results, only the averages of the ten instances were analysed to rule out the influence of special cases in the results.

A very important parameter is the ILP formulation and the instance size (amount of pages in the instance), the following instance sizes were created: 1, 2, 3, 4, 5 and 10. As LILP can handle larger instance sizes than LAILP, the latter formulation was never evaluated with ten pages in the input because preliminary tests showed this would take days to find any result (if any result was found at all).

The `simulator` and the `mixer` are responsible for separate parts of the instance generation and both have their own parameters. The size of the parameter space makes it impossible to test all possible combinations of settings within reasonable time. Therefore "default parameter sets" for the `simulator` and `mixer` parameters were calculated from observations on real world instances. The `simulator` parameter combinations are tested only in combination with the `mixer` default parameters and vice versa. The `solver` parameters, which have an even larger parameter space, are tested using instances created with default `simulator` and `mixer` parameters and the best performing parameters are used as `solver` default parameters.

This reduction of the parameter space still results in a large number of parameter combinations which have to be tested. An overview of the evaluated parameter space is given in Table 5.

8.1 Tested Parameters

This section explains which parameters of the `simulator`, `mixer` and `solver` were used for testing and shows the tested values and the default value for each parameter.

8.1.1 Simulator

The `simulator` parameters control how the paper tearing is simulated (the details were explained in Section 5.1.3). Table 6 shows the parameters and their default values and the parameter settings for the `cut.count`, `cut.top` and `cut.bottom` parameters which were selected for testing.

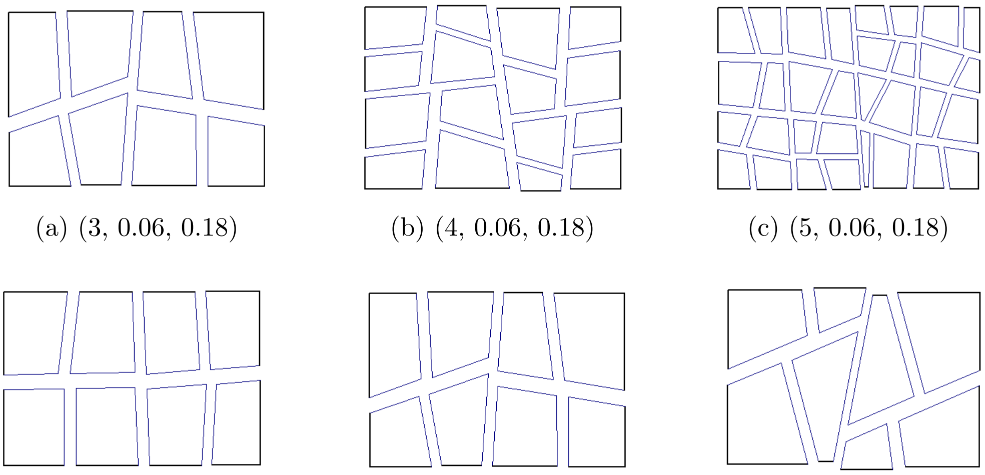
Test	Simulator	Mixer	Solver	
ILP	Lilp & LAilp	Lilp & LAilp	Lilp	LAilp
`simulator` parameter combinations	9	1	1	1
`mixer` parameter combinations	1	12	1	1
`solver` parameter combinations	1	1	15	15
Instance sizes	1-5	1-5	1-5, 10	1-5
Distinct configurations [a]	2x45	2x60	90	75
Number of runs	2x450	2x600	900	750
Total runs	3750			

[a] This is calculated from the number of ILP formulations times the parameter combinations of `simulator`, `mixer` and `solver` times the amount of different instance sizes tested.

Table 5: Amount of parameter combinations and configurations tested. The tool parameter combinations are multiplied by the distinct instance sizes to be tested. This result is multiplied by ten because each result is averaged over ten different runs. If each run is limited to twenty minutes the worst case runtime of all 3750 tests is 1250 hours or 52 days. As the solver is single threaded and three tests could be done simultaneously on three processors and because small instances sometimes did not need the whole twenty minutes, the wall clock runtime for all tests was about two weeks.

Parameter	Tested	Relation	Generator	Tested Values	Default
`paper.width`	no	absolute	fixed	-	295
`paper.height`	no	absolute	fixed	-	210
`cut.count`	yes	absolute	fixed	$\{3, 4, 5\}$	4
`cut.top`	yes	relative	gauss	center = 0 dev = $\{0.03, 0.06, 0.09\}$	center = 0 dev = 0.06
`cut.bottom`	yes	relative	gauss	center = 0 dev = $\{0.09, 0.18, 0.27\}$	center = 0 dev = 0.18
`mirror.x`	no	boolean	bool	-	50%
`mirror.y`	no	boolean	bool	-	50%
`rotate`	no	absolute	gauss	-	center = 0 dev = 0.08
`translate.x`	no	absolute	gauss	- -	center = 0 dev = 3
`translate.y`	no	absolute	gauss	- -	center = 0 dev = 3
`turn.left`	no	boolean	bool	-	50%

Table 6: Parameters of the `simulator` with regard to testing.



(a) (3, 0.06, 0.18) (b) (4, 0.06, 0.18) (c) (5, 0.06, 0.18)

(d) (3, 0.03, 0.09) (e) (3, 0.06, 0.18) (f) (3, 0.09, 0.27)

Figure 25: Samples of `simulator` test instances: the numbers below the tearing hierarchies indicate the parameters: (`cut.count`, `cut.top`, `cut.bottom`). The top three pages differ in the `cut.count` parameter, the bottom ones in the `cut.top` and `cut.bottom` parameter settings. The configuration (3, 0.06, 0.18) is shown twice for the sake of comparison.

The reason for selecting those three parameters for testing is that the amount of tears done (`cut.count`) and the amount of variation in one tear (`cut.top` and `cut.bottom`) contributes most to differences in the size of the LP model and are also most likely to vary in real manual paper tearing.

Figure 25 shows typical tearing results when using the selected parameter combinations. In the upper three pages the influence of different numbers of cuts is shown, in the lower three pages the effect of different tear slope variability is shown.

8.1.2 Mixer

The `mixer` parameters control how snippets from single pages are changed to model shear effects during tearing and the inexact process of digitalisation of the snippets. Table 7 shows all `mixer` parameters, their default values and how the `-d` and `-k` parameters are varied for testing.

The `-d` parameter models shear effects which cause the snippet borders to be extended outwards and was selected for testing because shear effects are a central reason that the paper tearing problem is difficult to solve. With exact tear edges and angles the solver could always pick the exact match or enumerate possible solutions from a very limited amount of exact matches. It will be shown, though, that a total absence of fuzziness of the input data can prevent the `solver` from proving that the currently best found solution really is the optimal solution.

Parameter	Tested	Tested Values	Default
`-c`	no	-	active
`-r`	no	-	active
`-m`	no	-	not active
`-s`	no	-	active
`-d`	yes	-d $\{0, 1, 2, 3\}$	-d 2
`-k`	yes	-k $\{0, 5, 10\}$	-k 5

Table 7: Parameters of the `mixer` with regard to testing.

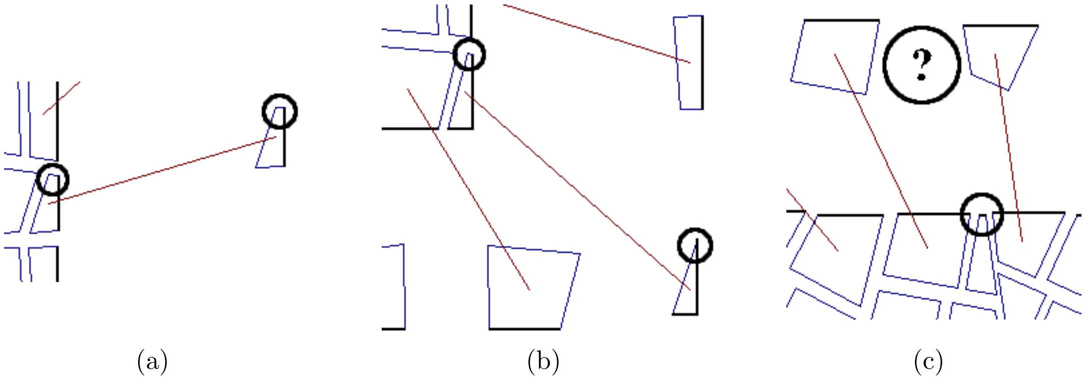


Figure 26: Samples of `mixer` test instances: (a) shows a changed outer angle, (b) shows a removed edge which changes the outer angle even more and (c) shows a snippet which is no longer an outer snippet because the short outer edge has been removed.

The `-k` parameter degrades the data by modelling that some very short snippet edges will not be detected by the scanning process. This parameter was selected for testing because it models a crucial part of the Puzzle problem which is not modelled in any other way in this thesis—the digitalisation process of the snippets.

Mirroring was not evaluated because it would only affect corner snippets in LILP solving and because LAILP is not capable of handling mirrored snippets. The extension of LAILP for the handling of mirrored snippets could be the subject of future investigations.

Figure 26 shows two examples for the effect of the `mixer` parameters in test instances: 26a illustrates the effect of the `-d` parameter which changes the outer angle of the marked vertex, 26b and 26c visualise the two possible effects of the `-k` parameter which removes edges, thereby changing angles and possibly even reducing the number of outer snippets.

8.1.3 Solver

Gathering empirical data about the solving process is done with an emphasis on the performance of the `solver`. The reason for this are the following observations made in preliminary tests:

Parameter	Tested Values
`--cplexMipEmphasis`	$\{0, 1, 2, 3, 4\}$
`--cplexBrDir`	$\{-1, 0, 1\}$
`--laeldelta`	{iloabs, iloabsdirect, nohelpvar, helpvar, twovars, posdif, limitpagemax, limitglobalmax}
`--lacir3`	{not active, active}

Table 8: Tested parameters of the `solver`.

- some parameter settings of CPLEX cause the `solver` to find the optimum within seconds, whereas other settings result in hours of computation time until the first feasible solution is found,
- if the optimality gap of the solution found by CPLEX is greater than a few percent, the correctness of the solution is bad. That means it pays off to find a near optimal or optimal solution.

Table 8 shows the tested `solver` parameters. The delta mode selection in combination with the `--cplexMipEmphasis` parameter which controls whether CPLEX solves with an emphasis on optimality or feasibility was selected for evaluation because these two parameters seemed to be the most promising performance tuning parameters. In some cases, the `--cplexBrDir` branch direction setting "-1" seemed to outperform any other setting and so this parameter was also evaluated in combination with the other parameters.

For LAilp, the delta modes `posdif` and `limitglobalmax` were not considered for evaluation as they performed bad on Lilp which is a subset of LAilp. The `--lacir3` parameter was evaluated to compare the CPLEX "lazy constraint" functionality to a circle elimination procedure which methodically checks circle constraint violations as soon as a new best integer feasible solution is found by CPLEX (see Section 4.2.4).

8.2 Page Sets

The first step in creating an instance of m pages is to use the `simulator` to simulate paper tearing m times with the same parameters. This yields a collection of torn pages which is called "page set" in the following. To produce comparable results, each page set of a certain size and with certain parameters was created only once for all tests requiring a page set with this number of pages and simulation parameters. Table 9 shows all generated page sets and for which tests the respective page sets were used.

8.3 Solver Test Results

Finding the `solver` parameter combination which performs best is the objective of the solver tests. The results of these tests are shown prior to the `simulator` and `mixer` test results. They were evaluated prior to those tests because the empirically "best"

(`cut.count`, `cut.top`, `cut.bottom`)	Pages	Tests
(3, 0.03, 0.09)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(3, 0.06, 0.18)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(3, 0.09, 0.27)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(4, 0.03, 0.09)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(4, 0.06, 0.18)	1, 2, 3, 4 and 5	`solver`, `simulator`, `mixer`: Lilp and LAilp
	10	`solver`, `simulator`, `mixer`: Lilp
(4, 0.09, 0.27)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(5, 0.03, 0.09)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp and LAilp
	10	`simulator`: Lilp
(5, 0.06, 0.18)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp
(5, 0.09, 0.27)	1, 2, 3, 4 and 5	`simulator`: Lilp and LAilp
	10	`simulator`: Lilp

Table 9: Page sets created for evaluating Lilp and LAilp.

Parameter	Tested Values
`--cplexMipEmphasis`	$\{0, 1, 2, 3, 4\}$
`--cplexBrDir`	$\{-1, 0, 1\}$
`--laeldelta`	{iloabs, iloabsdirect, nohelpvar, helpvar, twovars, posdif, limitpagemax, limitglobalmax}
Instance Size	1, 2, 3, 4, 5 and 10

Table 10: Solver tests: parameter combinations for Lilp.

Parameter	Tested Values
`--cplexMipEmphasis`	$\{0, 1, 2, 3, 4\}$
`--cplexBrDir`	$\{-1, 0, 1\}$
`--laeldelta`	{iloabs, iloabsdirect, nohelpvar, helpvar, twovars, limitpagemax}
`--lacir3`	{not active, active}
Instance Size	1, 2, 3, 4 and 5

Table 11: Solver tests: parameter combinations for LAilp.

parameter combination from the solver tests was used as `solver` default parameters for performing the `simulator` and `mixer` tests.

The `solver` tests were done on the default `simulator` parameters `cut.count`=4, `cut.top`=0.06 and `cut.bottom`=0.18 and the default `mixer` parameters `-d 2 -k 5`. Each instance size was tested for each parameter combination using ten instances, all results presented in the following diagrams are accumulated results over ten instances of the same size and also accumulated over all parameters not shown in the respective diagram. Tables 10 and 11 show the tested `solver` parameter combinations for evaluating Lilp and LAilp, respectively.

Lilp

The results of the `solver` tests for the Lilp formulation can be seen in Figures 27, 28 and 29. Figure 27 shows the percentage of instances which could be solved to optimality versus the `--cplexMipEmphasis` parameter setting. For an instance size of one page, all instances could be solved to optimality. This figure shows that the setting of "4" (documented in the CPLEX manual as "find hidden feasible solution") performs significantly better for larger instance sizes than all the other settings and that values "2" and "3", documented as "emphasise optimality" and "emphasise the best bound", respectively, perform inferior to the remaining settings "0" (this is the default which balances optimality and feasibility) and "1" (emphasise the search for feasible solutions). Because of these results, the `solver` tests of the LAilp formulation do not test the `--cplexMipEmphasis` setting for values of "2" and "3" and the `simulator` and `mixer` tests for Lilp use `--cplexMipEmphasis 4` as default `solver` parameters.

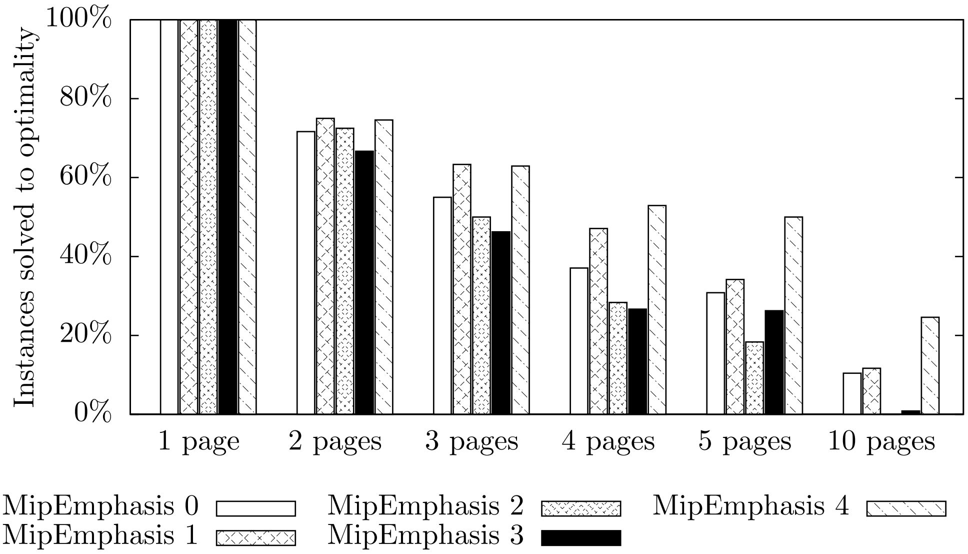


Figure 27: Optimally solved LILP instances versus `--cplexMipEmphasis`.

Figure 28 shows the percentage of instances solved to optimality versus the delta mode setting. It can be seen that `limitpagemax` and `limitglobalmax` are never solved optimally for instances containing more than one page. Furthermore the performance of `helpvar` and `posdif` degrades faster than the remaining delta mode settings and are clearly unfavourable for instances containing three or more pages. Because of the performance degradation of `posdif` which is even worse than the performance degradation of `helpvar`, the delta mode `posdif` was not considered for LAILP tests. The `limitglobalmax` delta mode was omitted for the LAILP tests too, because it performs very similar to `limitpagemax` while `limitpagemax` has more degrees of freedom—one delta variable for each page instead of one delta variable for all pages. It might seem illogical to use `limitpagemax` in the LAILP tests because it yields no optimal solutions for more than one page, but in fact this delta mode always yielded a *feasible* solution, even for large instances where `helpvar` and `posdif` sometimes yielded no solution. Because of its superior performance shown in the next paragraph, the `limitpagemax` was also evaluated with LAILP. Candidates for the delta mode setting of the `simulator` and `mixer` tests in combination with LILP are `iloabsdirect`, `iloabs`, `nohelpvar` and `twovars`. Of these four, `twovars` is best for the largest instance sizes and never falls far behind other delta modes and so `twovars` was used for the `simulator` and `mixer` tests.

Figure 29 shows the amount of correct edge assignments from LILP versus the delta mode setting. The performance of `limitpagemax` and `limitglobalmax` with regard to correct edges is clearly superior to all other delta modes for an instance of one page. For larger instances the performance does not vary much between the delta modes except for the `helpvar` and `posdif` delta modes which fall behind because they yielded no optimal solutions at all for larger instance sizes. Although the `limitpagemax` and `limitglobalmax` delta modes never achieve optimal results for instance sizes greater than one page, they generate solutions with a quality comparable to the other delta modes. As the experi-

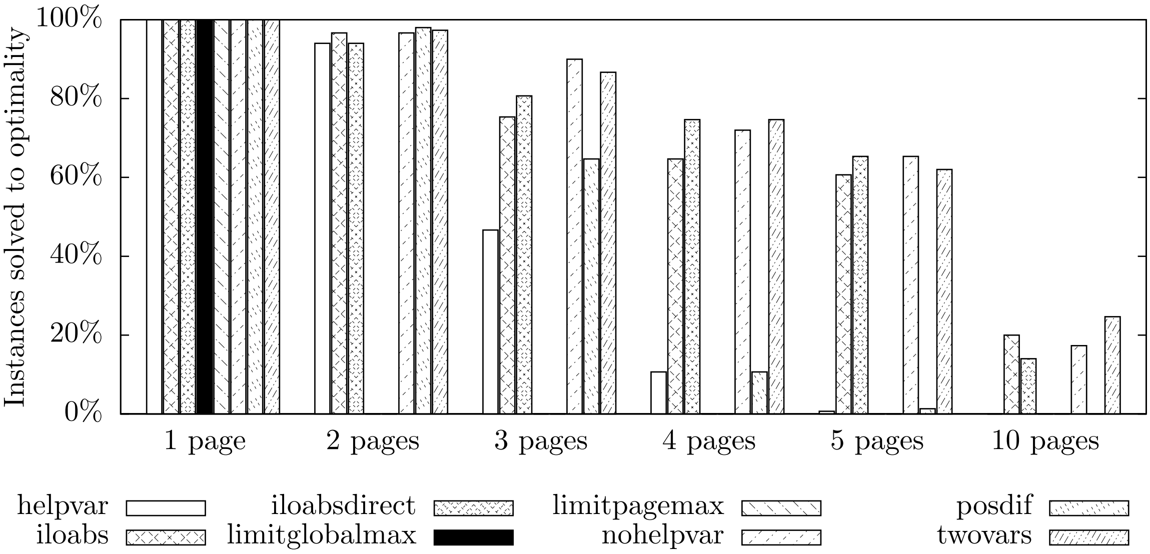


Figure 28: Optimally solved LILP instances versus `--laeldelta`.

ments were time bounded by twenty minutes each, it cannot be known from these experiments whether `limitpagemax` and `limitglobalmax` would perform equal or better than other delta modes for larger instance sizes if they were solved to optimality. For an instance size of one page where all instances in all delta modes could be solved to optimality within one second or less, these two delta modes should clearly be favoured over the other delta modes and are therefore ideal candidates for use in a hybrid algorithm which uses LILP for optimisation of single pages only.

A fact not shown in the figures is that the branch direction setting did not significantly influence the `solver` performance or solution quality. As this setting has a default value, this default value was used as for `simulator` and `mixer` tests. Also not shown is the fact that from all delta modes only `helpvar` and `posdif` failed to yield a feasible solution for some large instances. This supports omitting those two delta modes from further evaluation.

LAILP

The following paragraphs describe the test results obtained using LAILP. Because LAILP never solved any instance to optimality for instance sizes of two and more pages (within twenty minutes) the ratio of optimal to feasible solutions is not analysed. Instead, the ratio of feasible solutions to runs without any feasible solution is analysed in Figures 30, 31 and 32.

Figure 30 shows the percentage of feasible solutions for the reduced set of MIPEmphasis values. The value of “1” which stands for an emphasis in feasible solutions clearly outperforms the balanced emphasis “0” and the emphasis on hidden feasible solutions “4”.

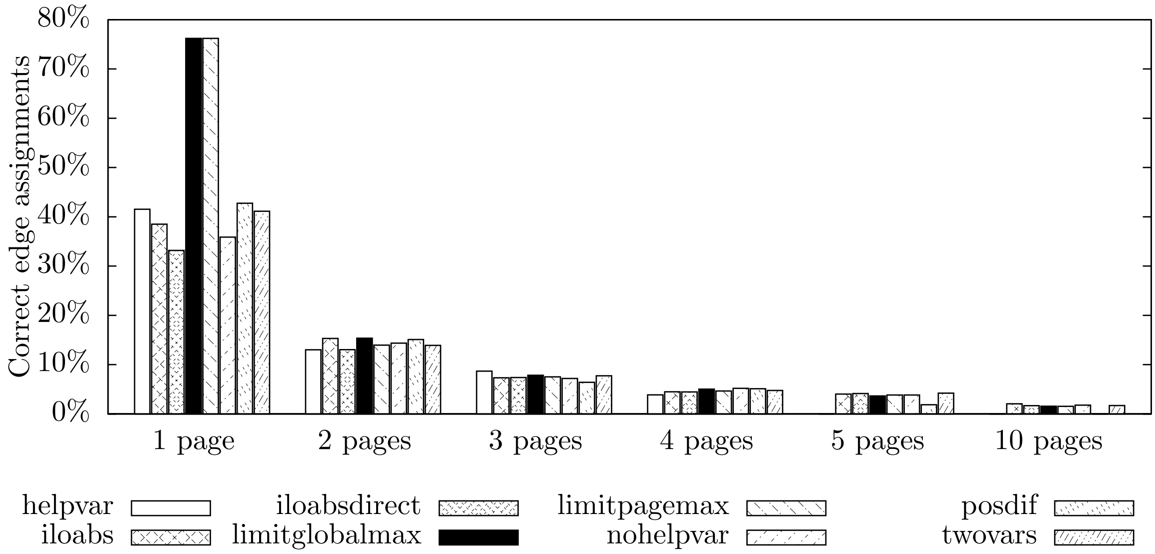


Figure 29: Correct edge assignments from LILP versus `--laeldelta`.

Therefore the `solver` and `mixer` tests use `--cplexMipEmphasis 1` during the evaluation of LAILP.

Figure 31 shows how the circle elimination without lazy constraints performs slightly better than the circle elimination using the callback together with lazy constraints. As the callback is always required for correct identification of inconsistent solutions and as the lazy constraints do not improve performance, the `solver` and `mixer` tests for LAILP do not use the `--laacir3` parameter.

The histogram in Figure 32 shows the percentage of feasible solutions versus the reduced set of delta modes. As could be expected from observations from the LILP tests, the performance of the `helpvar` delta mode degrades very fast for increasing instance sizes. The `iloabsdirect` delta mode stays ahead of the other delta modes for instance sizes of two and three pages but falls behind them for larger instance sizes. The other delta modes `iloabs`, `limitpagemax`, `nohelpvar` and `twovars` perform nearly identical with regard to output of feasible solutions.

Figures 33 and 34 show the percentage of correctly reproduced edge and follow assignments versus the delta modes. The `limitpagemax` delta mode performs better than all other delta modes for an instance size of one page, for larger instances the performance does not vary much between the delta modes.

A fact not shown in the diagrams is that for the `--cplexMipEmphasis 1 --laeldelta limitpagemax` setting, each instance containing one page was solved to optimality within five to ten seconds. Other delta modes were sometimes faster but not by more than one or two seconds and with far worse results for correct edge and follow assignments. This means that the `--cplexMipEmphasis 1 --laeldelta limitpagemax` setting could be used in a hybrid algorithm which uses LAILP for solving sub-instances consisting of one page in short time and with reasonable quality results.

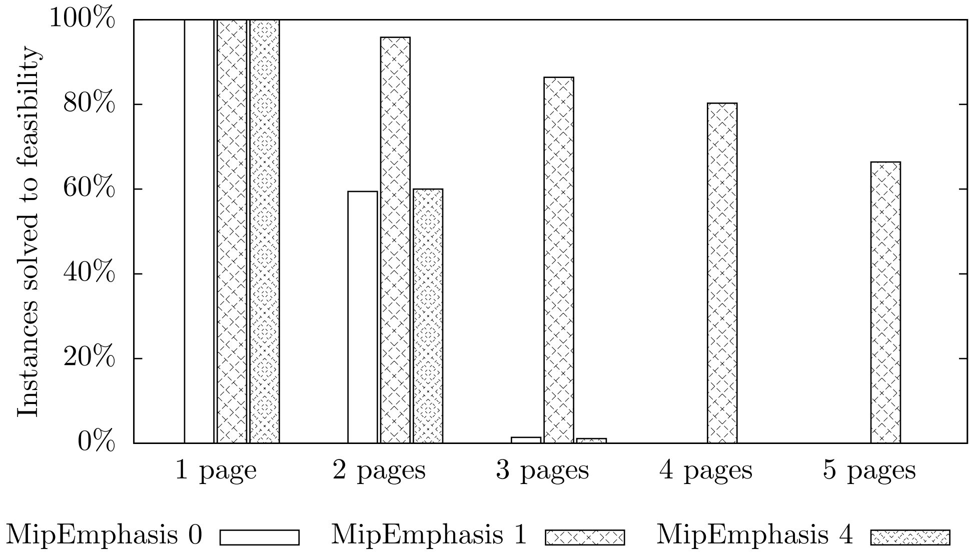


Figure 30: Feasibly solved LAILP instances versus `--cplexMipEmphasis`.

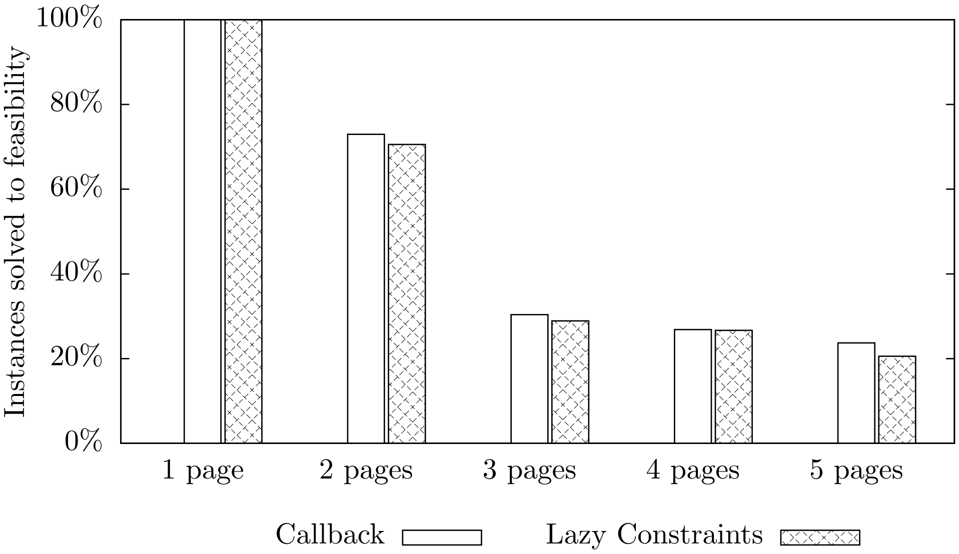


Figure 31: Feasibly solved LAILP instances versus `--laacir3`.

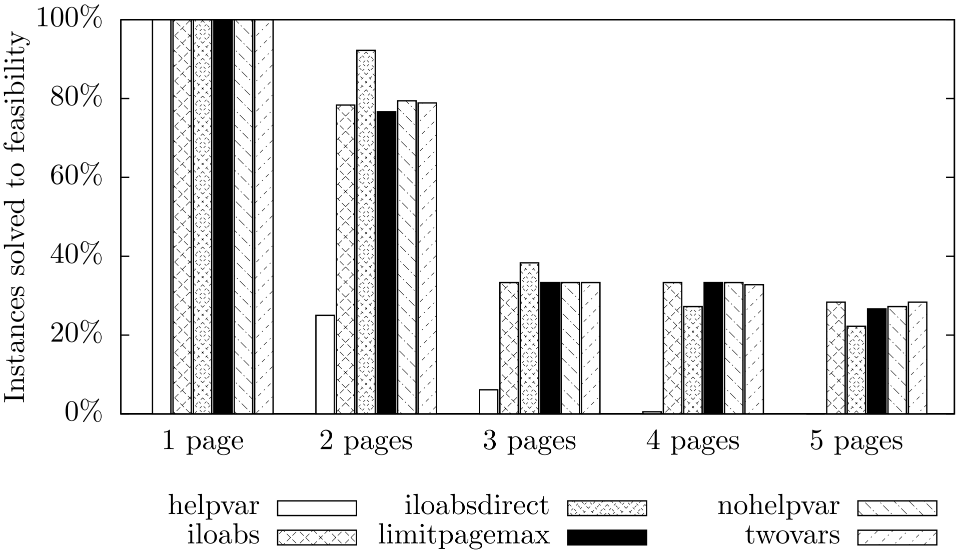


Figure 32: Feasibly solved LAilp instances versus `--laeldelta`.

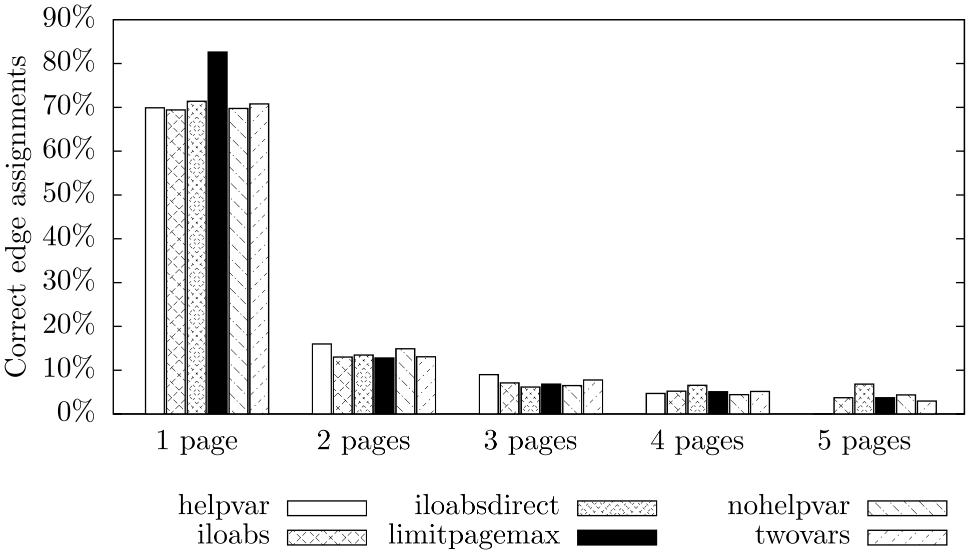


Figure 33: Correct edge assignments from LAilp instances versus `--laeldelta`.

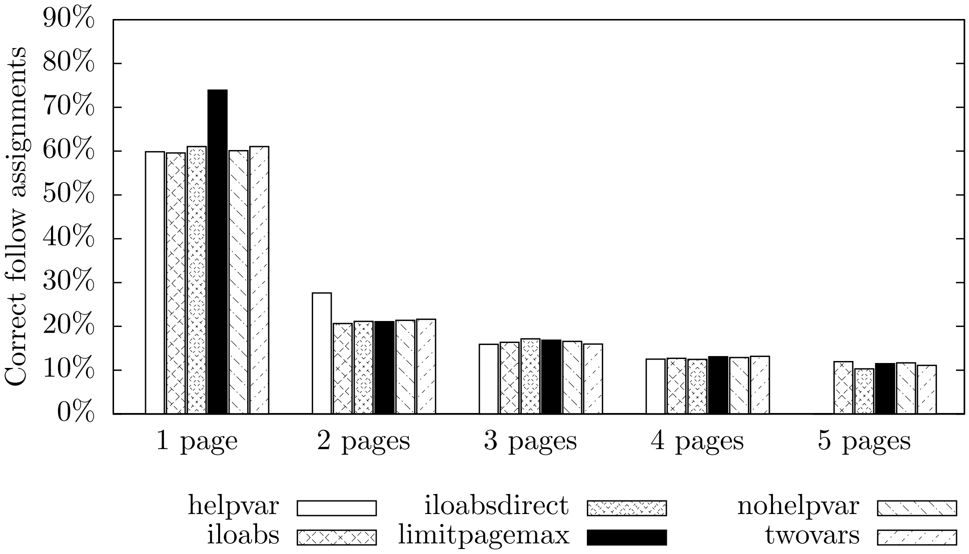


Figure 34: Correct follow assignments from LAILP instances versus `--laeldelta`.

Figure 35 compares the LILP and LAILP formulations with regard to the number of correct edge assignments in the `solver` tests. We can see that LAILP improves the number of correct edges for single page instances, for all bigger instances there is no significant difference in the percentage of correct edges. The number of correct edges of the `limitpagemax` delta mode is better by a small amount of six percent because the performance of this deltamodes is already very good in LILP, so it cannot improve much more in LAILP.

For the delta mode in the `simulator` and `mixer` tests, the `twovars` setting was chosen like in the LILP tests. This was done to be able to compare the `simulator` and `mixer` test results between LILP and LAILP and because this delta mode neither outperforms nor under-performs the other delta modes. The `twovars` delta mode seems to be the best delta mode in average and so it was chosen as default value for subsequent `simulator` and `mixer` tests.

8.4 Simulator Test Results

The `simulator` tests evaluate the `simulator` parameter influence on the performance and correctness of the results of the ILP formulations, each parameter combination is tested on each instance size with ten instances. Table 12 shows the `mixer` and `solver` parameters used for these tests. All results presented in the following diagrams are accumulated results over ten instances of the same size.

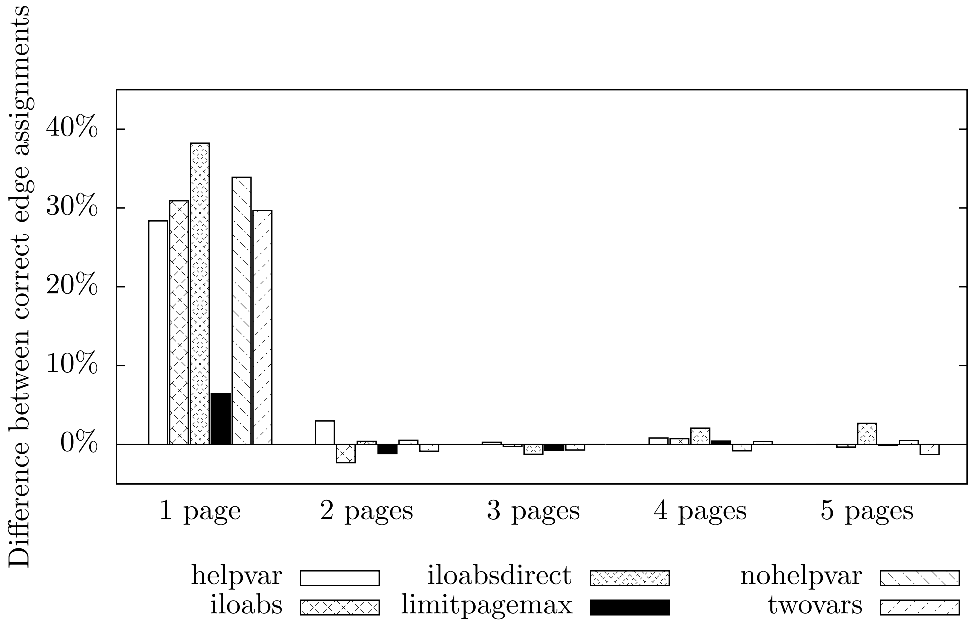


Figure 35: Comparison of correct edge assignments between Lilp and LAilp.

	Parameters Lilp	Parameters LAilp
`mixer`	`-d 2 -k 5`	

`solver`	`--laeldelta twovars`
	`--cplexMipEmphasis 4`
	`--cplexBrDir 0`
		`--laeldelta twovars`
		`--cplexMipEmphasis 1`
		`--cplexBrDir 0`

Instance Sizes	1, 2, 3, 4 and 5	

Table 12: Simulator tests: default parameter settings.

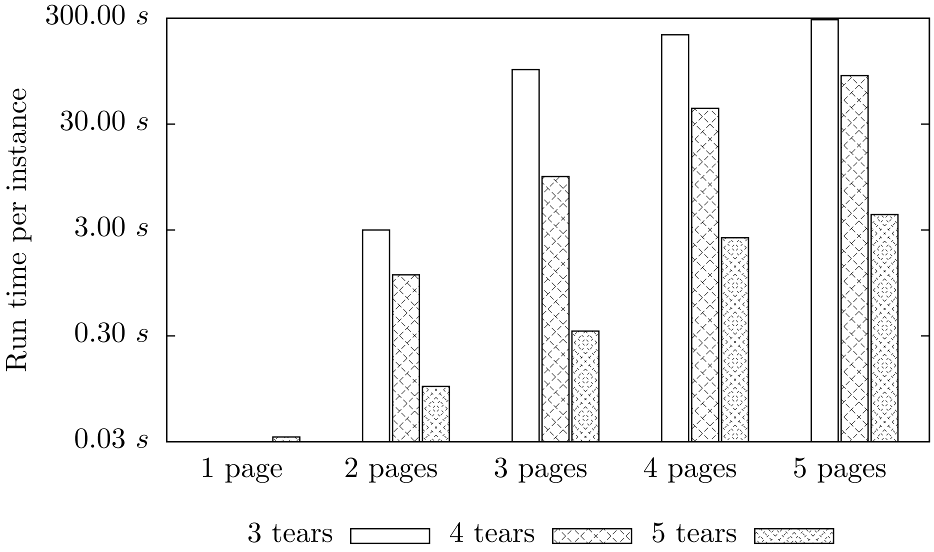


Figure 36: Run time for LILP instances versus number of tears.

LILP

The `simulator` test results for LILP are shown in Figures 36 and 37. The first figure shows a plot of the average runtime of one instance versus the number of tears done, where the time is shown on a logarithmic scale. In this plot we see that for fewer tears—which means for fewer snippets per page—the runtime *increases* dramatically. Of all instances, only two with five pages and *three* tears per page could not be solved optimally. Instances with fewer snippets per page clearly have a smaller search space of integer solutions and therefore should need less time, not more, than bigger instances. One possible explanation for the better performance of larger models is that they have less dense constraint matrices and can be processed faster by CPLEX because the branch and cut process is guided to optimal solutions faster.

Figure 37 shows that for instances containing one to three pages, fewer tears per page and therefore fewer snippets mean a better correctness of the results. This effect clearly occurs at the step from three to four tears per page which approximately corresponds to eight and sixteen snippets per page, respectively. For even more snippets per page this effect is no longer so clear and sometimes the correctness even becomes better by a small amount. This effect can be explained by the `mixer` variation of the vertex coordinates which has a small effect if a page is torn into eight snippets (three tears) because the snippets do not become very small. Tearing a page into about 16 or 32 snippets (four and five tears) creates much smaller snippets and so the `mixer` degrades the data used for the reassembling process far more than for three tears.

Not shown in plots is the effect of the `cut.top` and `cut.bottom` parameters which affect the tear slope variability. These parameters affect the solve time and edge assignment correctness only by a small amount where a larger tear slope variability means a slightly larger solve time and a slightly better edge assignment correctness.

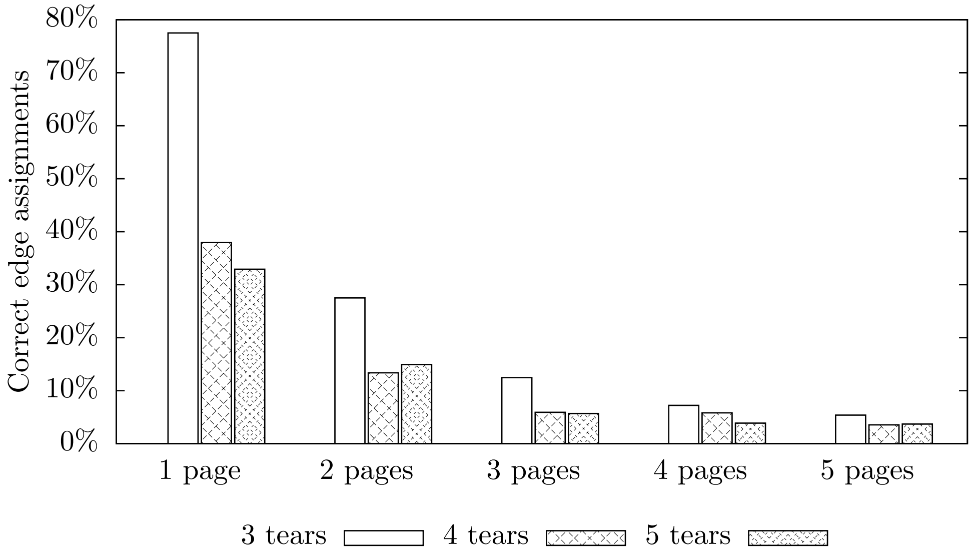


Figure 37: Correct edge assignments for Lilp instances versus number of tears.

LAilp

In contrast to the surprising run time results with Lilp, the LAilp `simulator` tests show that more tears per page cause the solving process to take longer or even find no result at all. Instances with five tears per page could never be solved optimally within the time limit of twenty minutes, some instances with five tears per page and five pages did not yield any feasible solution at all.

Figures 38 and 39 compare the percentage of correct edge and follow assignments versus the number of tears. It can be seen very clearly that the correctness is very good for instances containing one page if these instances contain three or four tears per page. In general the accuracy becomes worse for higher numbers of tears and larger instances. A further result is that the follow assignment correctness is always about twice as good as the edge assignment correctness (except for single page instances where both are nearly the same). This means that the model performs better for follow assignments and worse for edge assignments and improvements of the model should first be attempted in the edge assignment strategy.

As in Lilp, the effect of the `cut.top` and `cut.bottom` parameters which affect the tear slope variability is not shown in figures as these parameters do not affect correctness or run time significantly.

Figure 40 compares the number of correct edge assignments between Lilp and LAilp instances in the `simulator` tests. We can see that LAilp improves the number of correct edges for single page instances and for two page instances with three tears. Above that, there is no significant difference in the percentage of correct edges.

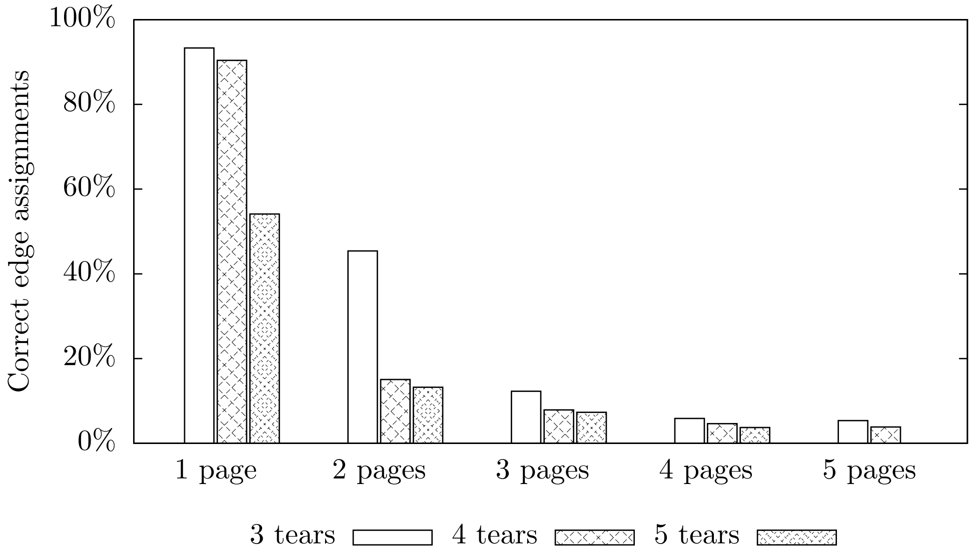


Figure 38: Correct edge assignments for LAilp instances versus number of tears.

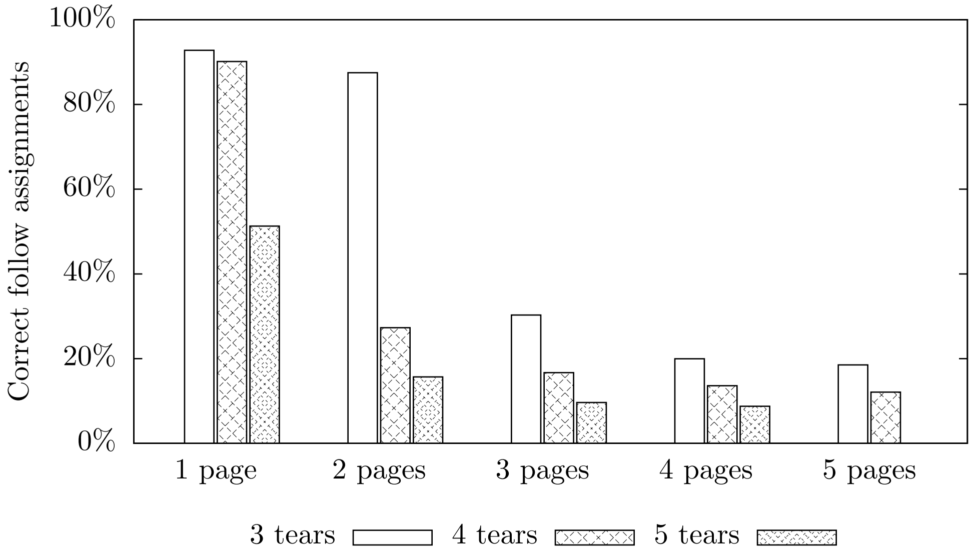


Figure 39: Correct follow assignments for LAilp instances versus number of tears.

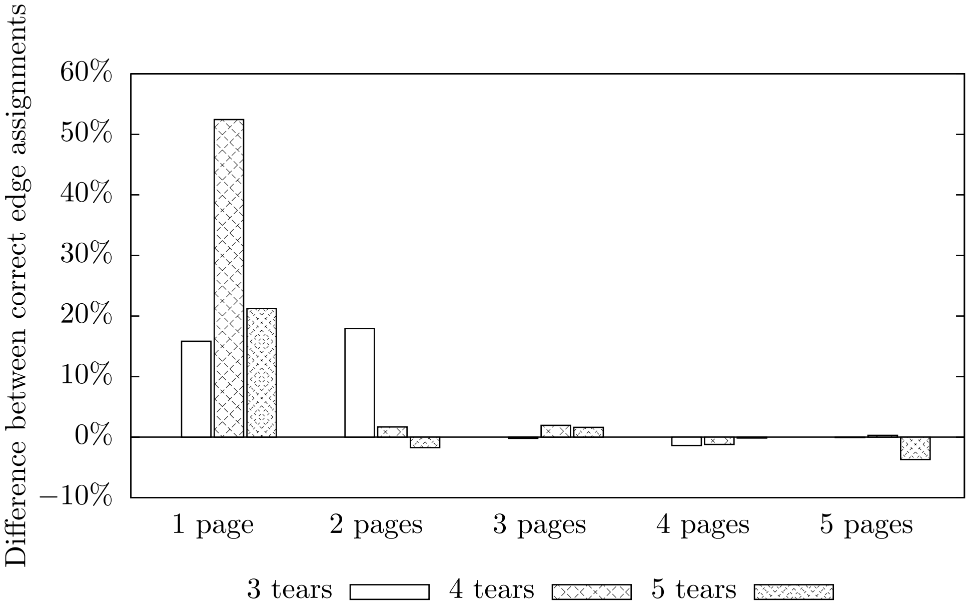


Figure 40: Comparison of correct edge assignments between LILP and LAILP.

	Parameters LILP	Parameters LAILP
`simulator`	`cut.count = 4 cut.top = 0.06 cut.bottom = 0.18`	

`solver`	`--laeldelta twovars`
	`--cplexMipEmphasis 4`
	`--cplexBrDir 0`
		`--laeldelta twovars`
		`--cplexMipEmphasis 1`
		`--cplexBrDir 0`

Instance Sizes	1, 2, 3, 4 and 5	

Table 13: Mixer tests: default parameter settings.

8.5 Mixer Test Results

In order to test the influence of `mixer` parameters on the performance and solution quality of the `solver`, two pagesets of the original ten pagesets with default `solver` parameters are "mixed" into five instances each, creating a total of ten instances (this is done for all tested instance sizes). This strategy allows to evaluate the effects of the `mixer` parameters on the same pageset but there are still two pagesets used for all tests, reducing the possible effect of special cases in the randomised pagesets from the `simulator`. Table 13 gives a compact overview of the `mixer` test settings.

LILP

The results are presented in Figures 41 and 42, whereas the first figure shows that no randomised vertex movement causes the solve times to increase. This is especially true for small instance sizes, with two pages the solver process needs only $\frac{1}{100}$ as much time if the results are randomised using `-d 1` compared to `-d 0`, for three pages this factor is still $\frac{1}{10}$. The second figure shows that for larger randomisation values the results get worse

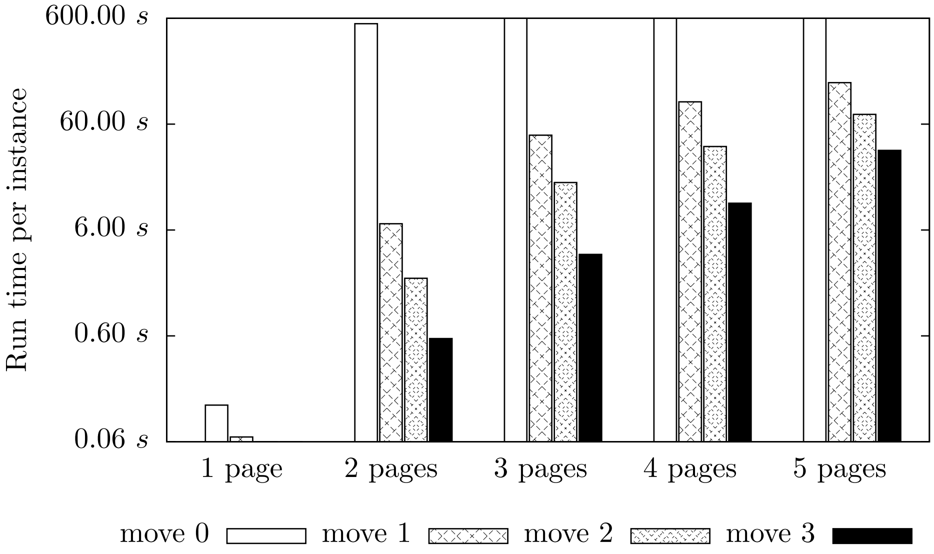


Figure 41: Run time for LILP instances versus amount of vertex randomisation.

which is an intuitive result. The fact that no randomisation means long run times can be explained by problems which occur if the relaxed solution of the ILP has an objective value of zero which is always the case if all snippets are in the model exactly as they were torn. In this case CPLEX has to visit a large amount of the search space to prove optimality and so no optimal solution is found because the found solution was not proved optimal within twenty minutes.

LAILP

The same effect occurs for LAILP where the only instances which did not yield any feasible solution were instances with no vertex randomisation. Figures 43 and 44 show the percentages of correct edge and follow assignments when using LAILP. No vertex randomisation means longer solve times, but as shown in the figures this also means a better edge and follow assignment correctness. Because of the time limit it is difficult to state results for larger instance sizes except that only small instances can be solved efficiently and that this conclusion is independent of the vertex randomisation.

In all the `mixer` test results, the effect of the `-k` parameter which deletes small edges was not shown, because this parameter does not have a noticeable effect on run time or edge or follow assignment correctness.

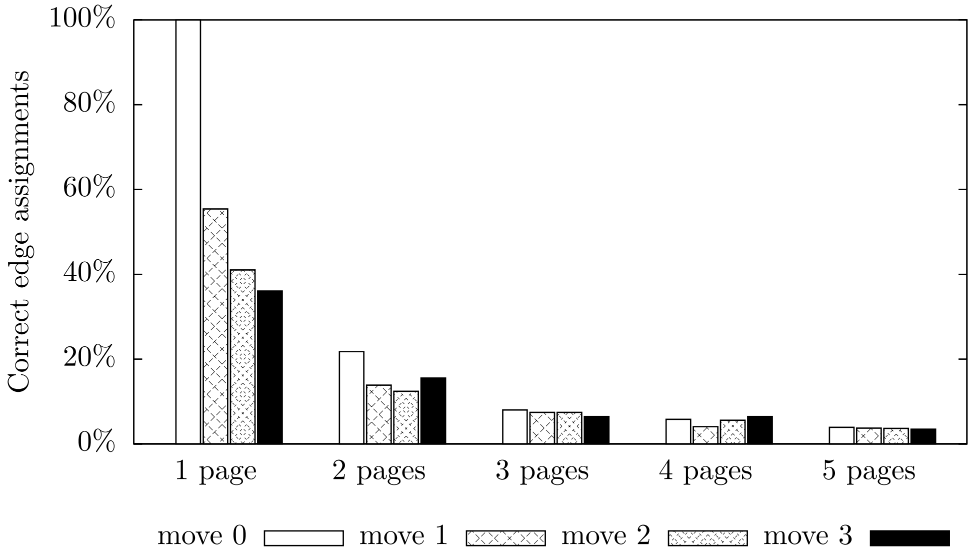


Figure 42: Correct edge assignments for LILP instances versus amount of vertex randomisation.

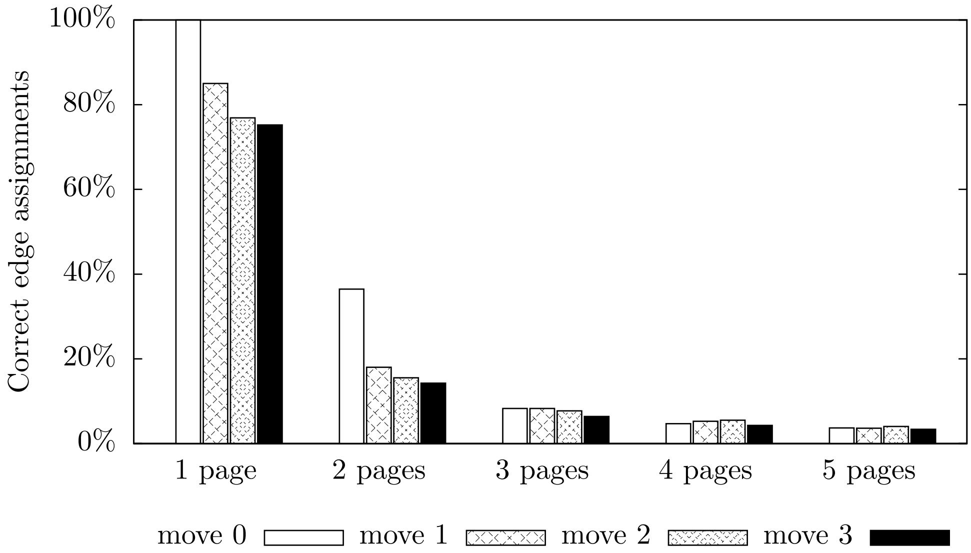


Figure 43: Correct edge assignments for LAILP instances versus amount of vertex randomisation.

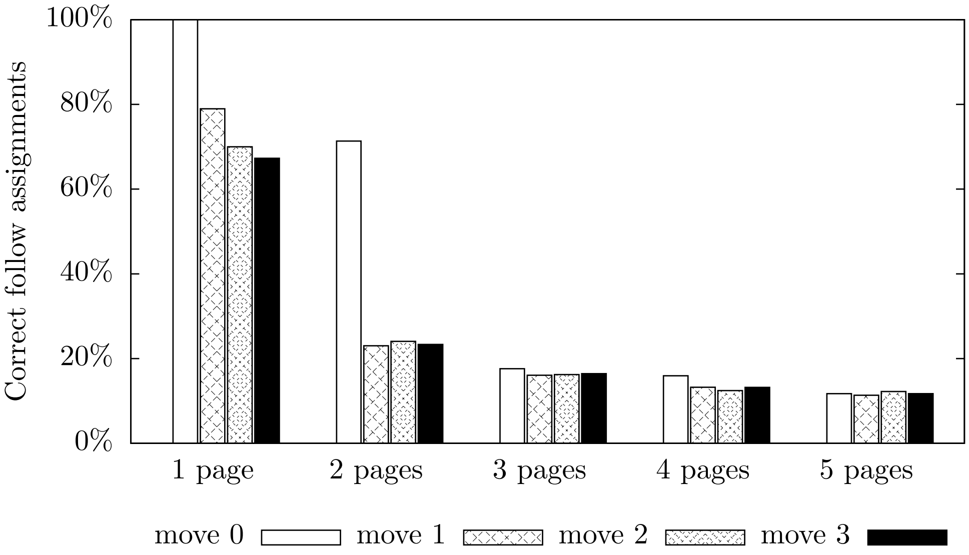


Figure 44: Correct follow assignments for LAilp instances versus amount of vertex randomisation.

9 Conclusions and future work

Conclusions

Two ILP formulations were developed, Lilp for solving with regard to lengths of snippet edges only and LAilp which also takes into account the angles between snippets. For the implementation and evaluation of these formulations, a C++ library was developed. This library allows to create solver applications and verification tools for custom solver approaches. It offers a generic framework using an extensible XML data interface for input and output of instance and solution data. The library facilitated the evaluation of the ILP formulations and it provides an foundation for future work in this problem domain.

A central conclusion from the tests is that the ILP formulations perform very good—both in run time and correctness—for the reconstruction of single page instances. For larger instances the run times quickly get unacceptable and no optimal solutions are found, in addition the correctness of the found solutions gets worse. This means that neither Lilp nor LAilp is useable as a stand-alone solution for reconstruction of torn documents, but both approaches could well be used in a hybrid algorithm where instances containing one page have to be solved often as a subproblem.

The ILPs were evaluated using the CPLEX solver from ILOG, Inc. where one configuration parameter—the MIPEmphasis—was identified as very important regarding the performance of LAilp where only the emphasis on feasible solution yields results for instance sizes above three pages. Without evaluating the influence of this parameter on the solver performance no instance sizes above three pages could have been evaluated for LAilp.

Eight flavours of objective functions for edge lengths and two flavours of objective functions for angles were developed and evaluated, where six of the edge lengths approaches yield the same objective function but with different ILP formulation methods for absolute value calculations, sometimes using special features of CPLEX. From the evaluation we can conclude that it pays off to use a good absolute value calculation method and sometimes it pays of not to use the CPLEX specific absolute value calculation method.

A conclusion not central to the Puzzle problem itself is that the most challenging programming task of the whole thesis was the visualisation tool, but it pays off to have such a tool which is able to show problem instances, torn pages and verified solutions where it is possible to visually verify the quality of a solution and check if the verification algorithms work correctly.

Future work

Future work is possible in several areas of the Puzzle domain in general and in several areas of possible solutions covered in this thesis in particular. As the evaluation of the parameters was very time consuming only a few CPLEX solver parameters could be evaluated and only a few parameters of the developed ILPs could be evaluated. Promising CPLEX parameters for future evaluation could be the variable and node selection parameters, for LAilp the ratio between the weight of length differences and angle differences and their effect on solution quality could be evaluated. Another promising possibility is to extend LAilp to take into account not only the angles between snippets but also the length of the cut edges enclosing these angles. If two long edges enclose an angle the angle is presumably correct, contrarily an angle enclosed by a long and a short edge can change very much if one of the two vertices of the short edge is displaced by shear effects (see Figure 26 for an example).

In the general problem domain of reconstructing torn sheets of paper various heuristic approaches could be taken to reconstruct pages, possibly using one of the ILP formulations developed in this thesis as sub algorithms for solving single page instances, but even without using any solver approach from this thesis it is possible to use the C++ library and the tools and data format from this thesis for generation of problem instances, solution verification and visualisation.

10 Indices

10.1 List of Figures

10.2 List of Tables

10.3 List of Algorithms

10.4 Bibliography

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. (last checked 11th June 2007).

[2] *ISO/IEC 14882:1998: Programming languages: C++*. American National Standards Institute, 1998.

[3] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-70431-5.

[4] J. Arlow and I. Neustadt. *UML and the unified process: practical object-oriented analysis and design*. Pearson Education Limited, 2002. ISBN 0201770601.

[5] D. Coetzee. Box-muller transform (C). http://en.literateprograms.org/Box-Muller_transform_%28C%29. (last checked 4th May 2006).

[6] G. Colvin, B. Dawes, D. Adler, and further contributors. Boost C++ Libraries. http://www.boost.org/. (last checked 7th January 2007).

[7] P.-A. Joy, T. Boutell, and further contributors. GD library - an open source code library for the dynamic creation of images. http://www.libgd.org/. (last checked 18th May 2007).

[8] J. Lakos. *Large-scale C++ software design*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996. ISBN 0-201-63362-0.

[9] M. Marciniszyn, A. Steger, and A. Weißl. E-Jigsaw - Computergestützte Rekonstruktion zerrissener Stasi-Unterlagen. *Informatik-Spektrum*, 27(Issue 3), 2004. DOI 10.1007/s00287-004-0395-8.

[10] S. Meyers. *More Effective C++: 35 New Ways to Improve Your Programs and Designs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 020163371X.

[11] S. Meyers. *Effective C++ (2nd ed.): 50 specific ways to improve your programs and designs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0-201-92488-9.

[12] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[13] B. Stroustrup. *The C++ programming language*. Addison-Wesley, 3rd edition, 1997.

[14] D. Veillard and further contributors. Libxml2 - the XML C parser and toolkit developed for the Gnome project. http://www.xmlsoft.org/. (last checked 18th May 2007).

[15] E. W. Weisstein. Box-muller transformation. `http://mathworld.wolfram.com/Box-MullerTransformation.html`. (last checked 4th May 2006).

A XML document type definition.

```
<!ELEMENT sniplib (
simparams?, protocol?, hierarchysnip?,
collection?,
edgeassignments?, followassignments?, mirrorassignments?)>

<!ELEMENT simparams (param+)>
<!ELEMENT param (fixed|gauss|bool)>
<!ATTLIST param
name CDATA #REQUIRED
relation (absolute|relative|bool) #REQUIRED
>

<!ELEMENT fixed EMPTY>
<!ATTLIST fixed
value CDATA #REQUIRED
>

<!ELEMENT gauss EMPTY>
<!ATTLIST gauss
center CDATA #REQUIRED
dev CDATA #REQUIRED
>

<!ELEMENT bool EMPTY>
<!ATTLIST bool
probability CDATA #REQUIRED
>

<!ELEMENT onesnip (polygon)>
<!ATTLIST onesnip
id CDATA #REQUIRED
>

<!ELEMENT polygon (point+)>
<!ELEMENT point EMPTY>
<!-- the id is the id of the polygon edge following this point -->
<!ATTLIST point
x CDATA #REQUIRED
y CDATA #REQUIRED
id CDATA #REQUIRED
startcut CDATA #REQUIRED
>
```

```
<!ELEMENT hierarchysnip (cutline, polygon, children)>
<!ATTLIST hierarchysnip
id CDATA #REQUIRED
>

<!ELEMENT cutline EMPTY>
<!ATTLIST cutline
paramA CDATA #REQUIRED
paramB CDATA #REQUIRED
paramC CDATA #REQUIRED
>

<!ELEMENT children (child*)>
<!ELEMENT child (hierarchysnip|onesnip)>
<!ATTLIST child
type (left|right) #REQUIRED
>

<!ELEMENT protocol (
(cutaction|placemaction|placeraction|placetaction|
 createstackaction|putfirstonstackaction)+
)>
<!ELEMENT cutaction EMPTY>
<!ATTLIST cutaction
paramA CDATA #REQUIRED
paramB CDATA #REQUIRED
paramC CDATA #REQUIRED
>
<!ELEMENT placemaction EMPTY>
<!ATTLIST placemaction
mirrorX (yes|no|true|false|1|0) #REQUIRED
mirrorY (yes|no|true|false|1|0) #REQUIRED
>
<!ELEMENT placeraction EMPTY>
<!ATTLIST placeraction
rotate CDATA #REQUIRED
>
<!ELEMENT placetaction EMPTY>
<!ATTLIST placetaction
translateX CDATA #REQUIRED
translateY CDATA #REQUIRED
>
<!ELEMENT putfirstonstackaction EMPTY>
```

```
<!ELEMENT createstackaction EMPTY>

<!ELEMENT collection (onesnip+)>
<!-- onesnip: see above -->

<!ELEMENT edgeassignments (assignedge+)>

<!ELEMENT assignedge EMPTY>
<!-- point denotes the snippet edge following this point -->
<!-- edge denotes the complete paper edge to which
 this snippet edge is assigned -->
<!ATTLIST assignedge
point CDATA #REQUIRED
paper CDATA #REQUIRED
edge CDATA #REQUIRED
>

<!ELEMENT followassignments (assignnext+)>

<!ELEMENT assignnext EMPTY>
<!-- point denotes a snippet edge -->
<!-- nextPoint denotes the snippet edge
     following the previous snippet edge  -->
<!ATTLIST assignnext
point CDATA #REQUIRED
nextPoint CDATA #REQUIRED
>

<!ELEMENT mirrorassignments (assignmirror+)>

<!ELEMENT assignmirror EMPTY>
<!-- id denotes a snippet id -->
<!ATTLIST assignmirror
id CDATA #REQUIRED
>
```