Automatic Label Placement for Technical Drawings

Master's Thesis
University of Turku
Department of Future Technologies
Software Engineering
2020
Ville Lehtinen

UNIVERSITY OF TURKU
Department of Information Technology

VILLE LEHTINEN: Automatic Label Placement for Technical Drawings

Master's Thesis, 65 p., 0 app. p.
Software Engineering
June 2020

A lot of research has been done to automatically label geographical maps. In the field of study, other types of images, like technical drawings, are almost completely disregarded. In enormous industrial projects, like shipbuilding or designing process plant facilities, there are countless of drawings produced. Labeling these drawings needs to be done manually by the designers and automating the labeling of these drawings would save countless of hours of the designers time.

This thesis aims to develop a method to conduct automatic labeling in the context of technical drawings. The main target are drawings produced in process plant design and marine industry, but the final solution is general enough to be used for other types of technical drawings as well. To achieve this, a set of requirements for the task is collected and an implementation principle for automatic labeling of technical drawings is presented. The solution is based on prior research done regarding automatic labeling of geographical maps as well as a new label candidate generation heuristic. The developed solution is evaluated by conducting an empirical study on an implementation of the new principle.

The empirical study indicates that the solution is practical enough to be used in real environments. However, it also reveals some improvements needed in the quality of the labeling as well as its performance.

Keywords: labeling, automatic labeling, technical drawings, empirical study

TURUN YLIOPISTO
Informaatioteknologian laitos

VILLE LEHTINEN: Automatic Label Placement for Technical Drawings

Diplomityö, 65 s., 0 liites.
Ohjelmistotekniikka
Kesäkuu 2020

Maantieteellisten karttojen automaattinen labelointi on laajalti tutkittu aihe. Tutkimusalalla muun tyyppiset kuvat, kuten tekniset piirustukset, eivät ole juuri saaneet huomiota. Suurissa teollisissa projekteissa, kuten laivojen tai prosessiteollisuuslaitosten suunnittelussa, tuotetaan suuria määriä piirustuksia. Projekteissa suunnittelijat joutuvat itse labeloimaan kyseiset piirustukset, mikä vie merkittävästi aikaa. Näiden piirustusten automaattinen labelointi säästäisi lukemattomia työtunteja suunnittelijoilta.

Tämä diplomityö pyrkii kehittämään automaattisen labelointimenetelmän teknisille piirustuksille. Päätavoite on laivanrakennuksessa ja prosessiteollisuuslaitosten suunnittelussa tuotettujen piirustusten labeloinnin automatisointi. Kehitetty menetelmä on kuitenkin yleiskäyttöinen myös muunkaltaisissa piirustuksissa. Työssä ensin kartoitetaan teknisten piirustusten automaattisen labeloinnin vaatimukset, minkä jälkeen kehitetään ko. vaatimukset täyttävä menetelmä. Menetelmä perustuu tutkimustietoon karttojen automaattisesta labeloinnista sekä tässä työssä kehitettyyn uuteen labeleiden kandidaattien generointi -heuristiikkaan. Kehitetty menetelmä arvioidaan empiirisellä tutkimuksella.

Empiirinen tutkimus osoittaa, että kehitetty menetelmä on riittävän käyttökelpoinen todellisessa ympäristössä. Menetelmässä ilmeni kuitenkin vielä parannettavaa labeloinnin laadussa sekä menetelmän suoritusnopeudessa.

Asiasanat: labelointi, automaattinen labelointi, tekniset piirustukset, empiirinen tutkimus

Contents

Chapter 1

Introduction

Drawings are everywhere. When almost anything is designed, the plans how to build it are drawn as drawings, a blueprint that describes and conveys the design and instructs its build. Drawings require not only the geometrical shapes and dimensions of the parts and their composition, but also a textual representation which part is which. The textual representations of the information content of a drawing are called *labels*. The labels can be in many shapes and sizes, but quite often they are text boxes that are on top of the represented feature or connected to it with a reference line called *leader*. When a designer adds labels to extend the information content of a drawing, he is *labeling* it. This labeling activity is quite often manual when technical drawings are considered. In some industries, like process plant or ship building, where the construction projects are massive, the amount of drawings produced can be thousands for a single project. Labeling all of these drawings manually is laborious activity. Automating it could save a substantial amount of time and money.

Automatic labeling is quite extensively researched subject when consider geographical maps. A lot of different approaches has been taken to overcome that problem. However, applying these methods is not very straightforward for technical drawings. Even though the purpose of a map and a technical drawing is the same: they describe objects and their relative locations. The graphical content and the way the labels are placed relative to their *features* (the objects they reference) are so different that the methods are not directly applicable to technical drawings. Some modification and applying has to be done in order to get those methods to work in technical drawings.

This thesis aims to develop a practical solution to automatically label technical drawings. The thesis is issued by CADMATIC ltd. who develops CAD-software(*Computer Aided Design*) aimed at designing and constructing process plants, ships and offshore constructions. That's why the main focus is drawings in plant and marine industry, but the final solution is general and can also be used for other types of drawings. As the labeling of technical drawings is ambiguous and there are no clear standards or rules how it should be done, requirements to automatically label technical drawings has to be determined. A design of a robust solution can then be developed that fulfills those requirements. This thesis therefore aims to answer the following research questions:

1. What are the requirements for a practical label placement solution for technical drawings?
2. What kind of automatic label placement solution fulfills those requirements?

This thesis looks deeper into labeling and its automation in general in Chapter 2. Requirements and rules for labeling technical drawings in plant and marine industry are collected in Chapter 3. Methods from past research of automatic labeling are looked into in Chapter 4. In the Chapter 5 a *label candidate generation* heuristic is developed that uses Luus-Jaakola heuristic as its base. A solution for the problem is presented in Chapter 6 that is designed using the methods discussed in the other chapters. An empirical study is conducted in Chapter 7 on the implementation made for CADMATIC to verify the solutions success and practicality. Finally a conclusion for the thesis is presented in Chapter 8 where the future possibilities are also discussed.

Chapter 2

Background

The main purpose of this thesis is to research methods and techniques for automatic labeling and implement a robust solution to automatically label technical drawings for plant and marine industry. The total time designers use for labeling drawings in a single project can take reportedly up to thousands of hours and automation could save a lot of this time. Additionally automatic labeling could improve the labeling quality, not necessarily aesthetic quality but consistency and correctness. Human errors are not uncommon and forgetting to label a single object can have quite costly consequences. In the worst case those errors can lead to errors in installation and the effects can propagate to be quite extensive.

This chapter presents some general background information about labeling as well as state of the art of automatic label placement methods.

2.1 Labeling

The purpose of labeling images is to visualize and link information associated with different graphical features. The labels extend the information content of an image and make it easier and faster to perceive. For complex images, labels can bring order and structure to the information the image presents and some images may be completely useless without

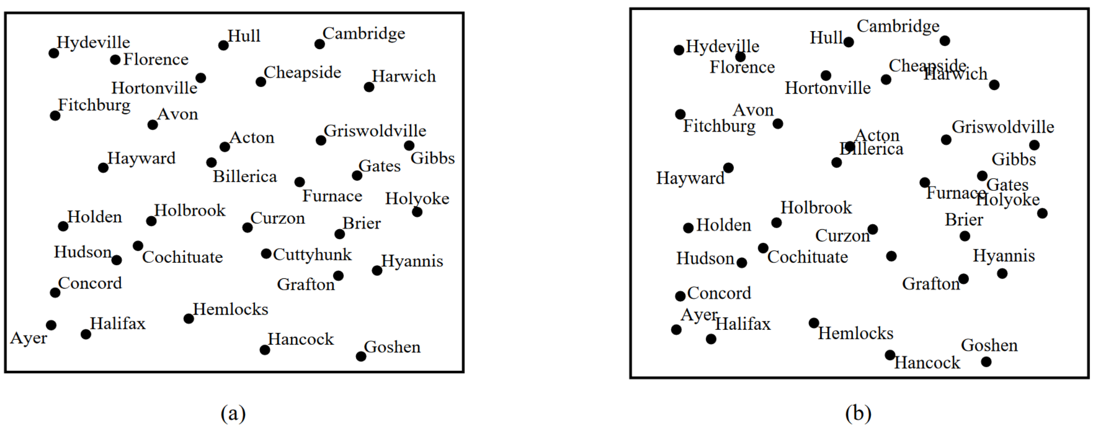

Figure 2.1: Example of a good (a) and bad (b) labeling of a map [1]

a textual description about the features it holds. That's why labeling is essential in many different areas like cartography, graphs and technical drawings.

The process of labeling images is not trivial. For a label to be considered good it needs to satisfy multiple requirements. The label should not obscure other graphical features of the image. By obscuring other features, the label may destruct crucial information of the image as well as make it difficult to distinguish the label and its association. The labels shouldn't overlap with each other since it could make it hard to see them apart. And finally, the labels' associations with their respective features should be easily distinguished so labels can't be mixed with other features.

Especially with crammed images, fulfilling all these requirements can be difficult and some compromises may have to be made. The fact that placing a label has global consequences by blocking possible placements from other labels makes the problem all more difficult. Traditionally the label placement has been made manually by people and the goodness of a label placement has been determined by the aesthetic perception of the experts. To find the best, or even a good, solution can take a lot of time and effort and the growing need for labeling dynamic maps, generated graphs and drawings makes it crucial to develop automatic solutions for the label placement problem. The figure 2.1 shows an example of a good and bad labeling of a map and figure 2.2 shows an example of a

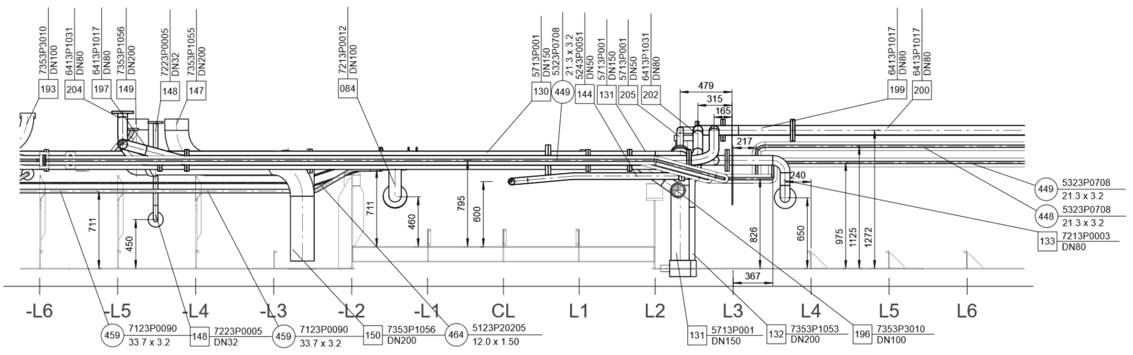

Figure 2.2: Example of a manually labeled technical drawing

manually labeled technical drawing.

2.2 Automating Label Placement

Automatic labeling has been researched quite a lot in the past. Labeling itself is quite an old subject dating back centuries even millennia. Ever since people started making maps they have also had to be labeled. The earliest references to automatic label placement are from 1960s when computers started to appear, but the main research took place a bit later when computers started to become powerful enough to be useful for such a demanding task. Especially in the 1990s and 2000's quite a lot of papers were published on the topic. An online bibliography of several related papers has been kept up by Alexander Wolff [2] who has also contributed in the field by publishing several studies on the subject. However the site has not been updated since 2009 and is not fully complete.

Automatic labeling is a broad subject. There are countless algorithms and heuristics for slightly different problems and applications. Arguably the most studied area is the *point-feature label placement problem* (PFLP) where the focus has mostly been on labeling geographical maps [3]. And for PFLP there are many different algorithms that try to optimize different aspects of the labeling like label number maximization and label size maximisation [3][4]. Extensive research has also been made for maps in the form of automatic line and area feature labeling [3]. Several papers touch multiple label placement problem, where more than one label is assigned to a single feature [5][6]. One quite popular area has been *boundary labeling*, where the labels are placed outside the actual image and the labels are connected to the features with leaders [7][8][9]. Some studies have also been conducted for labeling graphs and nodes [6]. The need to perform automatic labeling fast in order to label interactive maps in real-time has brought up new methods to do fast labeling in recent years [10][11]. Some research has also been made in a bit different fields than just plain 2D drawings like adding labels to 3D surfaces [12]. Even though extensive research has been made for labeling all kinds of maps and graphs, no studies were found dedicated to labeling technical drawings.

Due to the fact that automatic labeling is a broad and relatively old subject, many of different approaches, algorithms and heuristics have been developed. For a general labeling problem, determining whether a complete labeling exist, where no labels overlap, is proven to be NP-hard [3]. Almost all approaches to general labeling problem treat it as a combinatorial optimization problem [1]. There are two aspects to this kind of problems: *search space* and *objective function*. The search space is technically a sub problem for generating *label candidates* that are possible locations for labels to take. The objective function then determines which of these label candidates in the search space to choose. The objective function is also referenced as *label selection*. The approaches can be divided into two by the search space whether it is continuous or discrete. Most of the solutions use a discrete search space as continuous complicates the problem even more [13].

Since most of the research concentrates on the point-feature labeling problem with focus on geographical maps, the labels are required to touch their respective features. For this reason the search space for labeling is often a fixed number of label candidates where usually the label is placed so that the sides or the corners touch the point feature as seen in figure 2.3. Because most of the studies use this very simple search space, most of the

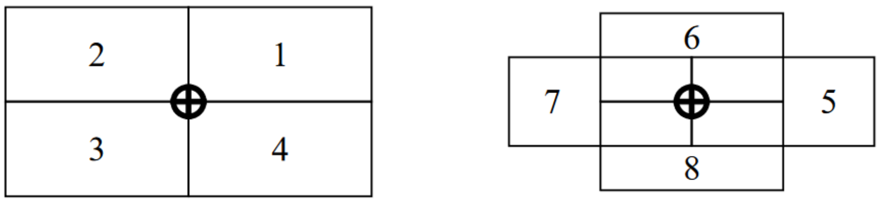

Figure 2.3: The most common eight label positions for a point-feature[1]

algorithms and heuristics developed target the optimization of the objective function.

Because in technical drawings the labels can be in arbitrary places and the label is connected to the feature with a leader, determining the search space is not as trivial as with maps. There seems to be no research made for this kind of setting. Chapter 6.2 discusses the label candidate generation problem in more detail and proposes a solution for determining the search space using a method based on Luus-Jaakola heuristic [14]. Chapter 4 explores the existing algorithms and heuristics for the objective function in more detail and discusses how they would apply to implement labeling for technical drawings.

Chapter 3

Label Placement Requirements

Before exploring the research done in field of automatic labeling, the requirements for labeling technical drawings in the plant and marine industry should be discussed. As it came apparent in the previous chapter, the research on automatic labeling has mostly focused on labeling maps and the solutions are tailored regarding their needs and criteria. A deeper look into the differences between geographical maps and technical drawings should be taken in order to distinguish the differences the methods will also bear.

This chapter discusses the differences between labeling maps and technical drawings, analyses an interview of two designers from plant and marine industry who have experience in labeling technical drawings, discusses the label readability and finally concludes the requirements and limitations for labeling technical drawings.

3.1 Labeling Maps vs. Technical Drawings

The purpose of maps and technical drawings is very similar. They both describe the relative positions of the objects that they contain. Still, the differences between them, like scale, representation and features, makes the overall labeling to be also different. As seen in the figure 3.1 the settings look so different from each other that one can only wonder whether they share anything in common at all. Despite all the differences, the methods to

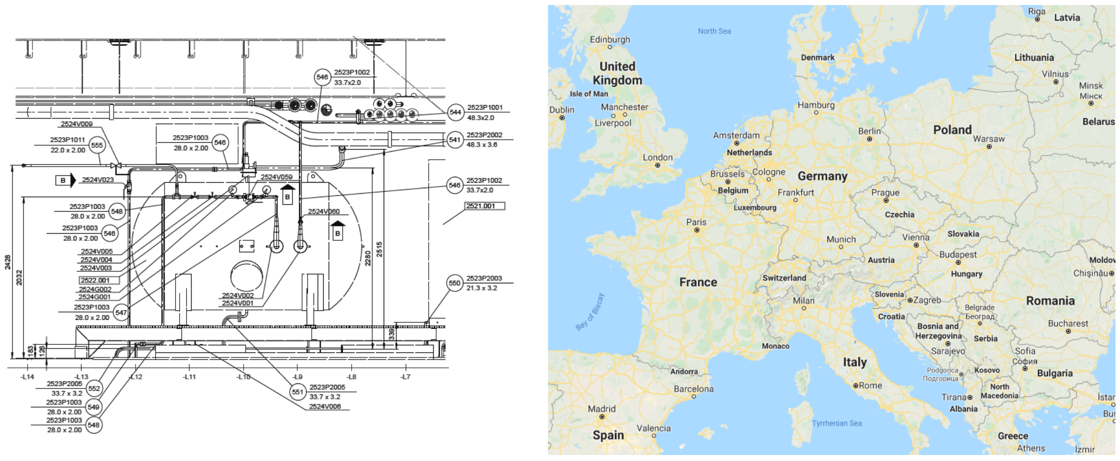

Figure 3.1: Example of a technical drawing and a map to elaborate the differences between them. The map is a screenshot from Google Maps.

automatically label both of them are surprisingly similar.

In traditional maps the most common feature is a point-feature, like a city, which is arguably the most research problem. In drawings there really aren't many point-features that requires labeling. Most of the features in drawings are either area-features, like equipments and structural components, or line-features, like pipes and rails, and even those features are fundamentally different from their counterparts in maps. For example area-features in maps, like lakes, are usually empty and the label can be placed inside the feature while in drawings the area-features are filled with details and it would be difficult or impossible to place the label inside it without destroying crucial information.

Another difference between maps and drawings is the amount of empty space in them. Maps are generally much more sparse and there is quite a lot of empty space to place the labels. Drawings in contrast are usually much more cramped with details. Especially in marine industry the space usage is optimized and the required components are packed as tight as possible. Because of this labels can't always be placed next to their features. Instead of placing the label next to its feature, the label is connected to it with a leader. This has become a standard way in the industry since using leader in all labels, even when it's next to its feature, makes the link between the label and its feature more explicit.

Unfortunately the usage of leaders complicates the process of automatic labeling. The leaders bring more graphical features to the already cramped image making it less readable and reducing the available space even further. When placing a label, it is not only required to consider what the label obscures, but also what the leader does. Also, the arrangement of the labels need to be considered well since the leaders shouldn't intersect in any circumstances. Even with all the drawbacks, the usage of leaders for drawings is required. More often than not there simply isn't enough space for a leaderless label to be in an unambiguous position.

3.2 Interviews to Identify Labeling Requirements

Two expert designers of the industry were interviewed in order to ensure that the labeling requirements meet the industry standards and practices. One of the designers was working in marine industry and the other one was a designer from process design. The main goals for these interviews were to find out how designers label drawings manually, do they have some industry standards or policies, are there differences in different drawings or industries and what in overall makes a good or bad labeling.

According to their answers no industry standards, policies or guidelines exist regarding label placement. However, in the marine industry the shipyard provides the designers some guidelines regarding the their drawings' style, content and policies. Otherwise the overall layout and labeling is relatively free and up to the designer to decide as long as the drawing is clear and readable. There also didn't seem to be any difference in labeling strategy between different drawings or between the two industries.

Neither of the interviewed designers seemed to have any clear strategy for placing labels. They both said to place labels by intuition, where it doesn't obscure other objects and what seems to be clear and aesthetic. They couldn't point out any non-obvious reasons for a label placement to be good or bad.

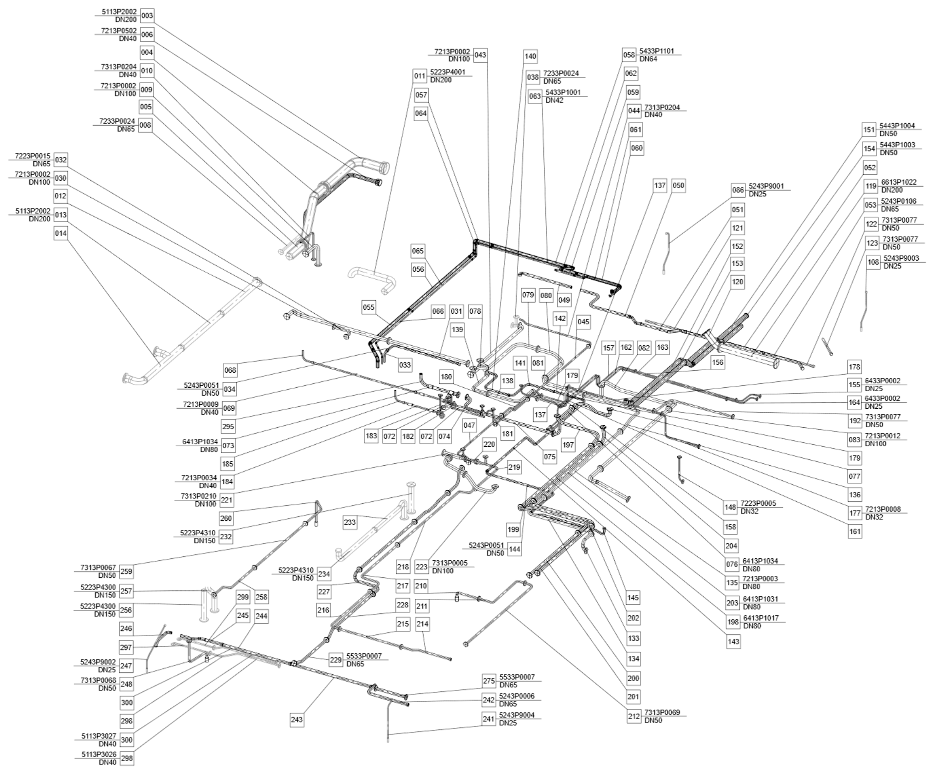

Figure 3.2: Example of grouping labels

Grouping of labels was also discussed. Quite often some nearby labels are grouped together in a list-like structure as shown in figure 3.2. To an outsider, the grouping may suggest potential association with labels in a group, but more often than not they're only grouped together to bring some structure to the drawing, enhance readability and make it more aesthetic. Like many other aesthetic properties, it is subjective matter whether it improves the quality of the labeling. The interviewed designers stated that grouping would definitely be a useful feature but incorporating a robust method to group labels can be a difficult challenge and thus is left as a bonus requirement for future research.

Both of the designers were a bit sceptical whether a complete automation would be possible. For simpler drawings a full automating of labeling could defiantly work, but for complex and cramped drawings they thought that designers would be needed to fix the errors the automatic labeling commits. They both suggested for a semi-automatic labeling where the automation would help the manual labeling by suggesting label placements and the designer would make the decision to approve or change the suggested position. It is also stated by Kakoulis and Tollis [15] that full automation with quality comparable to manual labeling may never be achieved and the most practical results could be received by some kind of interactive semi-automatic system. Even though the ultimate goal of this thesis is to find and implement a solution that shortens the amount of time used for labeling drawings, these kinds of features can be considered in the future.

The interviews answered the initial questions about how manual labeling is done in real environment relatively well. Some of the answers were a bit more vague than what would have been ideal, like the fact that there's no well defined strategy for labeling and the goodness of label is subjective apart from the fact that it shouldn't obscure anything. The interviews gave an answer to the main concern of whether there are some industry standards or policies that should be taken into account by the automatic labeling implementation. They also confirmed that the most important aspect of a label is its readability.

3.3 Readability of Labels

The interviews showed that maximizing readability should be one of the main requirements. Readability is a bit difficult concept since it depends a lot on the viewers subjective perception of aesthetics. Some studies have been conducted on the subject of label readability. Hartmann et al.[16] collected metrics for functional and aesthetic label layouts that tries to classify label layout styles and find aesthetic attributes that affect the readability of labeling. Also Barth et al.[17] have studied the readability of different leader styles for labels.

Hartmann et al.[16] divide label layout into two basic groups: internal and external labels. The external labels are outside of the feature they describe and are connected to it with a leader and the internal labels are just labels placed inside an area feature that they describe. Even though technically some labels in the technical drawings can be inside

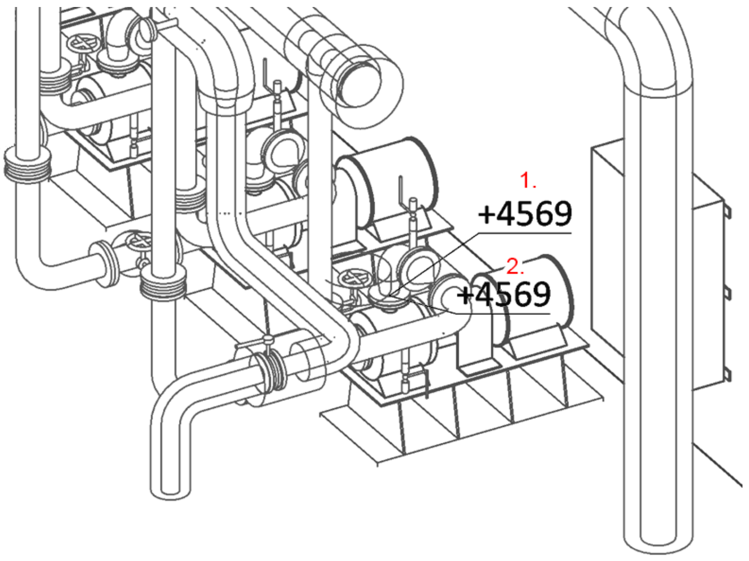

Figure 3.3: Example of a good (1.) and bad (2.) label placement from readability's point of view.

their respective features, they still can be considered external labels. It is more or less a standard way in the plant and marine industry that the labels still have leaders that point to a predefined point in the feature. Because the labels have a ambiguous point of interest where the leader connects to, the problem is regarded as point feature label problem.

In their study Hartmann et al.[16] present four different attributes for the labeling metrics: readability, unambiguity, aesthetic considerations and frame-coherency. The readability attribute includes metrics that already became apparent: labels shouldn't overlap with each other or other visual objects and the contrast between label and the background color should be high. The metrics of unambiguity are the closeness of the label to its corresponding feature, the position of the feature anchor-point and the number of bends in the leader. The aesthetic considerations include the more subjective metrics: the distribution of reference-points and labels and the alignment of labels, i.e grouping them together. The frame-coherency can be ignored as it considers dynamic labeling while technical drawings are often considered static. Examples of bad label readability and unambiguity are shown in figures 3.3 and 3.4. In the readability example the label number

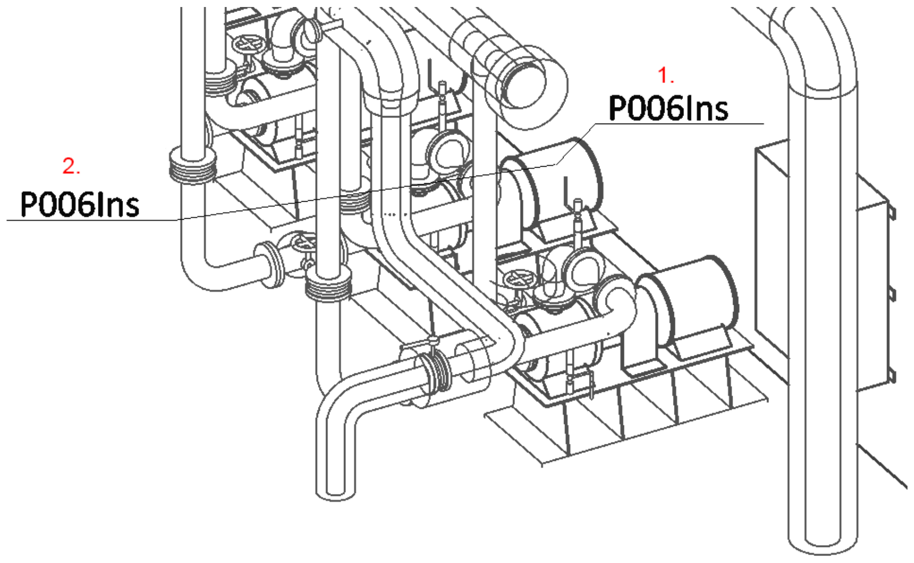

Figure 3.4: Example of a good (1.) and bad (2.) label placement from unambiguity's point of view.

two is bad since it overlaps with other objects in the drawing making it difficult to read. In the unambiguity example the label number two is relatively far away and the label's leader overlaps with a lot of graphics making it difficult to tell which feature it is trying to point.

These presented criteria can be useful, but they do not apply well in defining precise metrics that could be implemented in an automatic solution. They also include attributes that are general to the drawing style rather than required properties of the automatic labeling solution. For example the readability metric high contrast between label and background is part of the overall drawing style and not a concern of the labeling solution. The descriptions of some of the metrics are also a bit vague and incorporating them to the final solution is not straight forward as they don't provide a precise way to actually measure them. Take for example the distribution of anchor-points and labels. According to Hartmann the distribution shouldn't be too scattered or too uniform, but leaves it unclear what those mean in practice. Also because the aesthetic considerations are so vaguely described, this metric will be left outside the scope of this thesis.

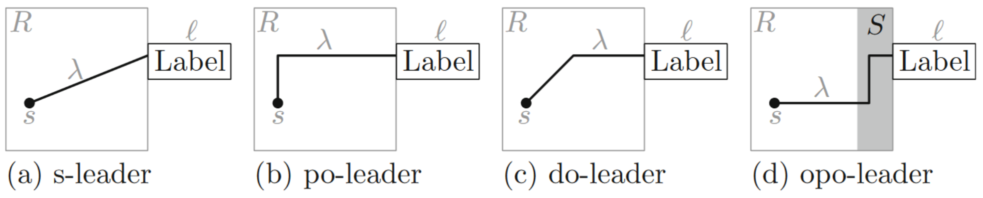

Figure 3.5: The different leader types introduced by Barth et al. [17]

Even though Hartmann et al.[16] state that a leader is more readable the less bends it has, Barth et al. [17] disagree in their paper that solely concentrates on leader readability. The different leader types studied by Barth can be seen in the figure 3.5. The paper concludes that the so called do-leader is the most preferred one by users but the s- and po-leaders did the best in an assignment task. It depends on the scenario which of these would be preferred. The do-leaders are perceived better which would make them the preferred one. For this case though, the final implementation should allow any kind of polyline as a leader since the current system allows all kinds of custom polyline leaders and the customers have already defined their own style and preference. The do-leaders could be used as the default leader type.

3.4 Time Requirement

Only functional requirements have been discussed but it's also important to consider the requirement of time. The ultimate goal of this thesis is to develop an automatic labeling method that reduces the time it takes for designers to label drawings. It would make sense to say the faster the better but obviously there is almost always a trade-off between quality and speed. It depends on the usage scenario what time scale the final solution should target.

If the algorithm takes longer to run than it would take for the designer to finish labeling, one could argue that the automated solution has failed. However the objective is to reduce the *designers'* usage of time. If a system produces drawings automatically, for example during night, it wouldn't matter if it took hours for the solution to conduct the labeling. Unfortunately the creation of drawings is usually not a scheduled task and the designers create them manually. In this case few hours would be way too long.

The usage scenario targeted is one where the designer makes the drawing and selects the group of objects that are automatically labeled. In this case it doesn't have to be absolutely instant, like label generation in dynamic maps, but the designer shouldn't be forced to wait for a long time either. An ideal time would be from few seconds to mostly around a minute. This way the designer wouldn't have to distract himself with something else and the result can be seen almost immediately.

Few seconds for a moderate amount of labels to place would be ideal, but if a robust solution can't be developed to execute in such a short time, a bit longer solution is still feasible. A designer could start the automatic labeling process and do something else, for example go for a coffee break, while waiting for the automatic labeling to finish.

3.5 Final Requirements

The section 3.1 concluded that labels should be connected to their respective features via leaders to reduce ambiguity. The interviews in section 3.2 showed that there does not exist any industry standards or policies that would affect the label placement and that clarity and readability are the most important requirements. The readability section 3.3 discovered that the label placement criteria that should be considered are readability, unambiguity and aesthetic considerations. And finally the section 3.4 conducted that the automatic labeling run time should be in the time scale of few seconds to a minute. From these, the final requirements are gathered and presented in the table 3.1.

Requirements	
R1. Readability	Labels should not obscure other graphical features or other labels
R2. Unambiguity	Labels are connected to their features with leaders and they should be as close together as possible
R3. Time	The run time of the algorithm shouldn't take more than few seconds
R4. Aesthetic considerations (bonus)	The labels distribution shouldn't be too scattered or uniform and they should be aligned to groups

Table 3.1: Requirements for automatic labeling in technical drawings

Chapter 4

Automatic Label Placement Algorithms

This chapter explores known potential algorithms and heuristics from literature that could be used for automatically placing labels in technical drawings. There are countless different algorithms and approaches for tackling the automatic labeling problem for slightly different scenarios and with different attributes. Most of the algorithms explained in this chapter are not necessarily directly applicable to the technical drawing scenario, but could still be used with some adjustments.

Most papers about automatic labeling methods concentrate on developing the objective function which in most cases doesn't depend on the implementation of the search space. That's why they can be applied with relative ease depending on how the final search space is going to be implemented. As most of the methods assume that the search space is predefined with eight points around a point feature, which doesn't apply to technical drawings where the labels can be placed technically wherever, a method for determining the search space is required. There doesn't seem to be any dedicated studies regarding the label candidate generation to compose a search space, not at least one that could be applied to technical drawings. The Chapter 5 proposes a solution for candidate generation problem that is based on Luus-Jaakola heuristic. Whenever search space is discussed, a predefined, finite and fixed set of label candidates (i.e. possible places for labels) is meant unless otherwise specified.

4.1 Algorithms for Objective Function

There is a vast amount of different algorithms and heuristics that try to find solutions for the general label placement problem. Like previously mentioned vast majority of them describe a solution for the objective function (label selection) with a discrete search space. They all have different attributes and slightly different goals to achieve.

Most of the earliest algorithms use exhaustive search methods to find the globally most optimal solutions or they search until a good enough configuration has been found. As the problem is NP-hard, these solutions aren't very practical in case with even a moderate amount of labels. Different algorithms have been developed to improve different aspects of the early methods to be more suitable for specific applications.

While many algorithms try to find optimal or good-enough solution, some algorithms try to maximize certain aspects of the overall labeling. Wolff [3] introduces algorithms for label number maximization as well as label size maximization. However, there is no point in applying them to technical drawings as all required labels must be placed and the label sizes should not be defined by the algorithm.

The algorithms presented in the following subsections are handpicked from the vast set of algorithms developed over the decades. The simple greedy algorithms are mostly for the purpose of an introduction to the methods and to give a general idea how a solution tries to solve the problem of objective function. The later heuristics, namely discrete form of gradient descent and simulated annealing, are more sophisticated methods for the objective function of automatic labeling. This is not at all an extensive survey of the automatic labeling algorithms. The main goal is to give a short introduction, and with that, justify the decision of using simulated annealing as the objective function in the final solution.

4.1.1 Greedy Algorithms

Several different greedy algorithms have been proposed for automatic labeling. Like greedy algorithms in general, the greedy labeling algorithms are generally fast but with the expense of producing worse quality labeling than many of their counterparts. Even though greedy algorithms do not usually provide a globally optimal solution, they are quick at generating a decent solution by only considering the best local decision on each step. Greedy algorithms can be a good alternative if more sophisticated algorithms are too time consuming. They're also usually very simple and easy to implement.

Tomáš [13] introduces the usage of *greedy randomized adaptive search procedure* (GRASP) for automatic labeling. The GRASP uses another greedy (random) algorithm as an initial step and then proceeds to do a local search that tries to find improvements to the label candidate selection in the context of the initial solution. For the initial greedy solution Tomáš [13] proposes so called *advance greedy* -algorithm that selects the locally optimal solution on each iteration and re-evaluates the scoring according to the newly made selection. The advanced greedy -algorithm can be used by itself, but the best result can be gained by combining it with GRASP.

Advanced Greedy

The basic algorithm goes as follows:

1. Calculate penalties for each label candidate according to how much they intersect with other candidates
2. Select the candidate with the lowest penalty for its label
3. For all labels that has not yet a candidate selected, recalculate the penalties for their candidates
4. Continue from step 2 until all labels have a candidate selected

The algorithm is simple yet quite powerful. Its biggest weakness is its greediness. As it selects the immediate best choice each time for non popular places the final choices might be very bad and thus the final result becomes bad as well (i.e. the final labels don't have any non-conflicting candidates). The algorithm also devalues popular places. Even though it does this by design so that labels want to be in non-popular places to avoid popular places becoming prone to overlaps, it is not an ideal way to handle the problem.

The solution is not straight applicable to technical drawings as it explicitly applies candidate intersections as a way to calculate penalties instead of a general cost function of the search space. However it could be applied if the penalty calculation is changed to an application-specific cost calculation. But then again, that defeats the idea of devaluing popular places to reduce overlaps. This would require an application-specific "popular place" calculation in order to keep the idea alive.

The advanced greedy is a bit naive solution, but it's purpose is not really to be the only solution but instead work as a stepping stone for the GRASP to get an initial solution to be improved.

GRASP

The GRASP (greedy randomized adaptive search procedure) consist of two steps: the initial greedy label placement and local search to improve the initial solution. Generally randomness is included in both of the steps and the procedure is ran multiple times to prevent the solution from getting stuck in a local minimum. A solution proposed by Tomáš' [13] does not use randomness (making it also deterministic) but it still follows the basic idea of a GRASP: It tries to improve the quality of the labeling that the advanced greedy produces. It does this by traversing through all the labels again and changing the selected candidates according to their current penalties which are not affected by the non-placed candidates anymore. The swapping of candidates is done multiple times, either until no further improvements can be made (it gets stuck to a local minimum) or a fixed number of iterations is achieved.

Steps for Tomáš' [13] solution for applying GRASP for automatic labeling:

1. Run the Advanced Greedy -algorithm
2. For each label check whether a better candidate is available considering the current label configuration and change the selected candidate to it
3. Continue from step 2 until no changes are made or enough iterations are achieved

The GRASP definitely improves the advanced greedy, but it is still inherits the same shortcoming of being greedy. It resolves the problem of devaluing popular places by placing the labels successively to their preferred places, but still suffers from the inability to escape local minimum as it only makes positive choices (i.e. it can only improve labels position). This is especially disadvantage for technical drawings as a label can take away quite many good positions from other labels by having its leader intersect with a lot of them. Having a poor choice in the beginning (but still best for the individual label) can ruin the whole rest of the labeling as the choice becomes fixed.

4.1.2 Discrete Form of Gradient Decent

Even though greedy algorithms can find decent solutions relatively quickly, the quality can be improved greatly with some more sophisticated approaches. Christensen et al. [1] discusses algorithms using *discrete form of gradient decent*. In this approach the labeling is improved by applying an operation from a set of possible operations (e.g. change the position of a label) that has the best possible impact on the objective-function that valuates the goodness of the labeling. This is quite similar to the GRASP method discussed in section 4.1.1 as it gradually improves the labeling by changing the label positions after the initial arrangement. The general outline of the discrete form of gradient decent goes as follows:

1. Select a random candidate for each label
2. Repeat until no more improvements can be made
   (a) For one label, calculate the change in objective function for repositioning it to each of its candidates
   (b) Set the label to be the candidate that results in the most improvement

One method that uses this approach is Hirsch's [18] algorithm that is quite different from the GRASP method. It uses infinite allowed positions for a label by using a circle around a point feature which the label must touch. The method then uses overlap vectors to determine how much and which direction the label should be moved in order to improve its position. Because this method forces the labels to be at a certain distance from the feature, this approach is not applicable for technical drawings as the labels should be allowed to reside anywhere on the drawing. It wouldn't be impossible to apply a similar approach to a setting with unrestricted label placement possibilities by moving the label in the actual drawing's 2D space instead of in a circular one dimensional line. However, trying out new, uncertain and difficult-to-implement methods is not an ideal approach in the scope of this thesis and this idea is not considered any further.

The discrete form of gradient descend seems to be better than the other greedy approaches. However, the problem with it is the same as with the greedy methods discussed earlier: it can get stuck to a local minimum. This could be improved by introducing randomness and probabilistic attributes to the algorithms. Even then finding global minimum is unlikely but at least then multiple different local minima can be traversed through improving the chance of finding a decent solution.

4.1.3 Simulated Annealing

The general problem of gradient decent methods not being able to escape local minima is improved in *simulated annealing* (Christensen et al. [1]), which is a stochastic gradient descend method. Stochastic meaning it's using randomness and probabilistic elements in its search. This makes it undeterministic and removes some systematic nature of the algorithm making it possible to jump out of local minima.

The simulated annealing is quite general and profilable approach. It is technically a gradient decent method but instead of choosing the best imminent choice, there is a chance for it to make a worse decision which is controlled by a *temperature* parameter T. As the algorithm progresses the temperature lowers thus making better choices more likely and the algorithm starts to approach a specific minimum.

Simulated annealing has four aspects to it that are independent of each other: initial configuration, objective function, configuration change method and annealing schedule.

The initial configuration is the starting setting. It can either be just a random solution or a solution created by some other method, which then the simulated annealing tries to improve. As pointed out by Christensen et al. [1], using some other method for the initial configuration makes almost no difference at all in the processing time or the quality of the final labeling, thus making random placement a viable option.

The objective function computes the change ΔE in the labeling quality of the whole drawing. It is used for determining whether a change was good or bad. For example it could be counting the amount of overlap a label has compared to its previous location. Since this objective function will be calculated several times during simulated annealing, it is important for it to be relatively simple and quick to compute in order to keep the algorithm performance at acceptable level.

The configuration change method determines how changes to the labeling will be performed. Christensen et al. [1] used two different strategies. Choose a label to change randomly from all labels and choose a label from all conflicting labels. The one which picks one from all labels is slower as it doesn't converge as fast but it is more likely to jump out of local minima.

The annealing schedule determines how the temperature T changes with the progression of the algorithm. Generally the temperature should lower faster when a lot of changes are made, slow down as the frequency of accepted changes drops and the algorithm should stop when the frequency of changes drop below certain threshold. These parameters are quite application-specific. For example the initial value of T is quite dependent on the scoring of labels and what value range ΔE is expected to have.

For general label placement problem the basic steps for simulated annealing goes as follows:

1. Select an initial solution (can be random placement)
2. Repeat until improvement rate drops below given threshold
   (a) Decrease the temperature T
   (b) Select a label and change its position
   (c) Calculate the change ΔE in objective function that was caused by the repositioning
   (d) If the labeling became worse, discard the change with probability $P = 1.0 - e^{-\Delta E/T}$

The simulated annealing approach seems really promising for automatic labeling of technical drawings. Christensen et al.[1] did an empirical study of different automatic point feature labeling algorithms and concluded that simulated annealing dominates the other tested methods in labeling quality. Those other methods include random placement, greedy depth-first placement, discrete gradient decent, Hirsch's algorithm[18] and Zoraster's algorithm[19]. However simulated annealing is quite a bit slower than for example gradient decent and greedy algorithms. Despite this drawback, simulated annealing is still very customizable. With certain parameters it can be made to basically function as a regular gradient descend method. It's difficult to say whether it is too slow for labeling technical drawings before actually seeing how fast it performs in real scenarios.

4.2 Comparison

As there are countless different heuristics for automatic labeling, surveying all of them is out of the scope of this thesis. The heuristics presented in this chapter are ones that seemed promising or are otherwise simple solutions to give a basic idea of the label placement methods. The greedy algorithms were mostly included for the reason of giving an idea of a simple solution. The rest of the algorithms presented are possible considerations for the final objective function to be used for the final solution.

It's difficult to compare how the algorithms would fare against each other in a real scenario without actually implementing and testing them. Christensen et al.[1] did an empirical study of different methods for automatic labeling. Even though the tests were done labeling maps, the results generally apply to the case of technical drawings as well. In the comparison, simulated annealing dominated in the terms of labeling quality (had lowest amount of total overlaps). Its only drawback compared to others was its slowness. It appears to be around ten times slower than gradient descend and around hundred times slower than a greedy algorithm. It overall seems to be a solid method to use in the solution developed in this thesis. The empirical study conducted in Chapter 7 evaluates the run time of the solution. If the method appears to be too slow for practical usage, a slower alternative, like the gradient descend, could be used instead.

Chapter 5

Generating Label Candidates

For technical drawings, generating label candidates is not as simple as just picking the four corner positions around the feature. A method for finding good places that satisfies the requirements (table 3.1) is required. Computing the whole search space for a single label to find a minimum is not feasible approach and some optimization methods needs to be used. Even though there exists a ton of research for automatic labeling, none of them really considers the case where the labels position is arbitrary and could be placed anywhere regardless of where the feature is. This chapter defines the problem and proposes a method for generating arbitrarily placed label candidates that uses heuristic introduced and called Luus-Jaakola [14] as a basis for the solution.

5.1 Problem definition

The problem is to find multiple different candidates for a single label to be passed to the objective function in order to avoid overlaps. Each label has unambiguous best candidate, which is the global minimum of the label's cost space. In addition, other potential candidates have to be found that reside in different positions. After all, the objective function requires different alternative candidates for the labels. For this, the found candidates should not be too close together so that there are more variety for the final configuration.

If the label's candidates are too close together, they may all conflict with a single other label.

The problem of finding multiple candidates for a label is defined as follows. Let w and h be the minimum horizontal and vertical distances between candidates, b_i's are the bounds of the drawing and the set

$$S = \{(x, y) \in \mathbb{R} \mid b_{xmin} \leq x \leq b_{xmax},\ b_{ymin} \leq y \leq b_{ymax}\} \in \mathbb{R}^2$$

represents all possible locations for label candidates. The cost space for each label is defined as a function

$$f : S \to \mathbb{R}.$$

A first candidate $a_0 \in S$ has to be found, so that

$$\min_{a \in S}(f(a)) = f(a_0).$$

Then consecutive candidates $a_n \in S$ have to be found so that

$$\min(f(b)) = f(a_n), \quad \text{where}$$

$$b \in S \setminus \big\{(x, y) \mid x_i - w < x < x_i + w,\ y_i - h < y < y_i + h, i \in \{0, ..., n-1\}\big\}.$$

In other words, first a global minimum has to be found, which is the first candidate. Then the consecutive candidates are the minima that are constrained to locate at some distance from the other candidates. This way multiple meaningful possible places for candidates are found.

The cost space f is arbitrary and it can be relatively heavy operation to calculate $f(a)$'s. Finding a solution that results the global minimum without fail can be difficult. As the goodness of a label placement is subjective and the cost space tries to be a concrete representation of that, the global minimum might not even represent the ideal placement according to some observers. That is one reason why finding the absolute global minimum is not crucial and so, the method for finding candidate positions doesn't have to be definite. A solution that quickly (only few calculations of f) finds decent candidates would be ideal.

5.2 Candidate Generation using Luus-Jaakola

Because the cost space $f : \mathbb{R}^2 \to \mathbb{R}$ is arbitrary, some of its properties are unknown (like whether it's differentiable). *Luus-Jaakola (LS)* [14] is an optimal method for finding decent minima. It is a general global optimization heuristic that uses random search to solve non-linear programming problems. The basic procedure for LS goes as follows:

Let $S \in \mathbb{R}^n$ be a bounded set, $f : S \to \mathbb{R}$ be the cost function and $a \in S$ is a candidate solution. To find an approximate minimum of f do the following procedure.

1. Pick k random points from S to form a set S_k and select the initial solution a that produces the smallest cost:
$$\min_{c \in S_k}(f(c)) = f(a)$$
2. Repeat until i iterations have been completed
   (a) Select k random points from S around a in the range r to form set S_k
   (b) Select point b that produces the smallest cost from the random points
   $$\min_{c \in S_k}(f(c)) = f(b)$$
   (c) if $f(b) < f(a)$ then select b as the new candidate solution a
   (d) Reduce search range by the amount e so that the new range $r = (1 - e)r$

The LS is quite profilable and can be tweaked to suit different functional and time requirements. Since the plain LS finds only one minimum and the label candidate generation problem requires the localization of multiple potential candidates, the basic procedure needs to be modified.

In the modified version the search is run multiple times, once for each candidate, and after each step the found candidate is added to a set that constrains the possible places for label candidates. And since drawings are two dimensional, the generality of n dimensions can be dropped to just two. Some further development of the heuristic can also be made for this particular case as the LS is meant to be very general. When a good candidate is found (i.e. after multiple iterations no better candidate is found) the procedure could either be stopped to save time or the range could be reduced even more so that a better approximation of a minimum could be achieved.

The following procedure describes the basic steps for generating label candidates using Luus-Jaakola.

Let $S \in \mathbb{R}^2$ be a bounded set, $f : S \to \mathbb{R}$ be the cost function, $a_n \in S$ are the candidate solutions and w and h are the minimum horizontal and vertical distances that candidates must be from each other. To find n candidates for a single label, the following has to be done for each candidate $a_c, c \in 0, ..., n$:

1. Pick k random points from S to form a set S_k and select the initial solution a_c that produces the smallest cost

   $$\min_{a \in S_k}(f(a)) = f(a_c)$$

   and is not too close to other candidates so that

   $$(x_d, y_d) = a_c - a_j, \quad |x_d| > w,\ |y_d| > h, \quad j \in 0, ..., c-1, \quad c \neq 0.$$

2. Repeat until i iterations have been completed or enough iterations have been made without finding new solution

   (a) Select k random points around a_c in the range r to form a set S_k

   (b) Select point b that produces the smallest cost from the random points

   $$\min_{a \in S_k}(f(a)) = f(b)$$

   and is not too close to other candidates so that

   $$(x_d, y_d) = b - a_j, \quad |x_d| > w,\ |y_d| > h, \quad j \in 0, ..., c-1, \quad c \neq 0$$

   (c) if $f(b) < f(a_c)$ then select b as the new candidate solution a_c

(d) Reduce search range by the amount e so that the new range $r = (1 - e)r$

The heuristic doesn't necessarily give the global minima as it relies on randomness and probability in finite number of iterations, but it finds a solution very quickly. The fact that the cost space is formed from the subjective opinions of experts what a good label placement should be, and those opinions are translated to mathematical constructs and rules, the optimal solution from the cost space might not be the best when asked from different experts. That's why a heuristic like Luus-Jaakola that is uncertain but fast is a good choice.

The choice of constraining the candidate positions to be minimum distance from each other is a bit artificial. However it is required since the candidates have to be different enough from each other. The more sophisticated automatic labeling heuristics divide the problem into search space and objective function. When labeling maps the search space is trivial while still the candidates being very different. Without artificially constraining the candidate search the candidates would likely be very similar thus making objective function's decision, what candidate to choose, meaningless. Making the candidates different while them still being relatively good makes the objective functions task to find the global minimum possible.

Chapter 6

Implementation

The previous chapters explored the theory and prior work of automatic labeling. This chapter collects the bits and pieces together, takes the heuristics that fits the requirements and ties them together into a working implementation. An implementation frame is provided that is evaluated with an empirical study in the next chapter. The implementation is discussed in very general level and many implementation specific details are intentionally left out. The implementation is made for CADMATIC, a commercial company, and application details including the source code are not public.

The implementation can be divided into five parts: *label scoring, label candidate generation, label selection, automatic labeling controller* and *application's interface connector* as shown in table 6.1. The implementation of the different parts reflect the concepts, methods and ideas discussed in the previous chapters. *Label scoring* is based on the requirements introduced in Chapter 3, *label candidate generation* is based on the method that uses Luus-Jaakola discussed in Chapter 5 and *label selection* bases its implementation on the findings of Chapter 4 which surveys automatic labeling algorithms from literature. The other two parts *automatic labeling controller* and *application's interface connector* are required components from software architecture's point of view. Their purpose is to assemble the individual solutions of sub problems to a usable software.

This chapter goes through each part of the implementation. The different possible methods and approaches are analyzed and chosen based on a brief analysis. A general idea of possible implementations are presented with some examples of the final outcome.

Implementation Parts	
Label Scoring	Functionality to calculate label candidate costs in different positions and configurations
Label Candidate Generation	Defines the search space for label selection. Finds multiple suitable positions for each label
Label Selection	The actual selection algorithm implementation. Tries to find the minimum of the search space (i.e. chooses a configuration of all the generated candidates)
Automatic Labeling Controller	Controls the overall flow of the automatic labeling procedure. Functions as a service provider for the different modules and also works as a public interface for the application to use automatic labeling
Application's interface connector	Application side implementation for using the automatic labeling

Table 6.1: Parts required for fully functional implementation of automatic labeling.

6.1 Scoring Labels

The label candidate generation tries to find good initial placements for all of the required labels and the objective functions purpose is to find a good combinatorial configuration from label candidates. In order to measure and rank the label candidates, some kind of scoring mechanism is required. Most of the algorithms don't depend on a specific implementation of the label candidate scoring. What they require is to measure a label's goodness in particular labeling configuration as well as a total measurement for a whole configuration. In most cases the scoring only considers the amount of overlaps a label candidate has and sometimes the labels have preferred positions, like the point feature should be at the down-left corner of the label. Because in technical drawings the labels can be placed anywhere on the drawing and they are connected to the feature with a leader, a more specific scoring system is required.

The scoring of the label candidates comes straight from the requirements (table 3.1). The first requirement R1 *readability* demands that a label shouldn't obscure underlying graphical features or other labels, thus the system should penalize candidates that overlap with other features. This also includes leaders overlapping with other graphics. The second requirement R2 *unambiguity* tells that labels should be connected to their feature with a leader and they should be close to their respective features. This means that the distance from the candidate to the feature should affect the score as well. The third requirement R3 *time* does not affect the scoring of the labels directly. It implies that the methods used to calculate the candidate scores should not take too long. Finally, the last requirement R4 *aesthetic considerations* is not as straight forward. It tells that labels shouldn't be too scattered or uniform and they should be aligned to groups. As this requirement considers the global arrangement of the labels, it shouldn't affect the score of a single label. The requirement could be taken into account when scoring the whole labeling configuration by evaluating the spatial distribution of all selected labels.

A label candidate has two different costs: *local* and *global cost*. The local cost only considers the label candidates overlap with underlying graphics and the distance from the feature. The local cost is invariant to the changing configuration as the properties that are used for its calculations don't change when the global configuration changes and thus can be calculated only once. The global cost is the local cost summed with properties that have global consequences like overlap with other labels, leader intersections and spatial distribution. Table 6.2 shows different metrics that are used to calculate costs for each candidate. The cost of different properties' weights can be adjusted case by case. For example if label proximity to its feature is a more preferred property, the cost of label distance can be weighted to be more compared to other properties.

Local Costs	
Label and leader overlap with graphics	The total area of overlap with graphical features weighted with the importance of the overlapping features
Label distance from its feature	Label's Euclidean distance from its associative feature
Global Costs	
Label overlap with other labels	Total area of overlap with other selected labels
Leader intersecting with other leaders	Number of intersections a label's leader has with other selected labels' leaders
Leader overlapping with other labels	Length of the leader's overlapping segment with other selected labels

Table 6.2: Metrics for label cost calculations.

The label scoring is a function that takes parameters like the label candidate size, position and reference point as an input and produces the candidate score as an output. If a single label is taken and the candidate score for each x and y coordinate of the drawing is plotted, a cost map is generated. Technically the label candidate generations mission is to find good enough minima from this cost map in order to figure out possible placements for a label. Figure 6.1 shows a plotted local cost map for a single label that uses a scoring implementation following the table 6.2's metrics. The plotted map shows the form of the function as well as different properties of the scoring implementation. For example, the overlap with underlying graphics can be seen to be weighted more than the distance from the feature.

Presenting the implementation details for each of the scoring metrics would be quite lengthy and out of the scope of this thesis. It's also heavily dependent on the underlying

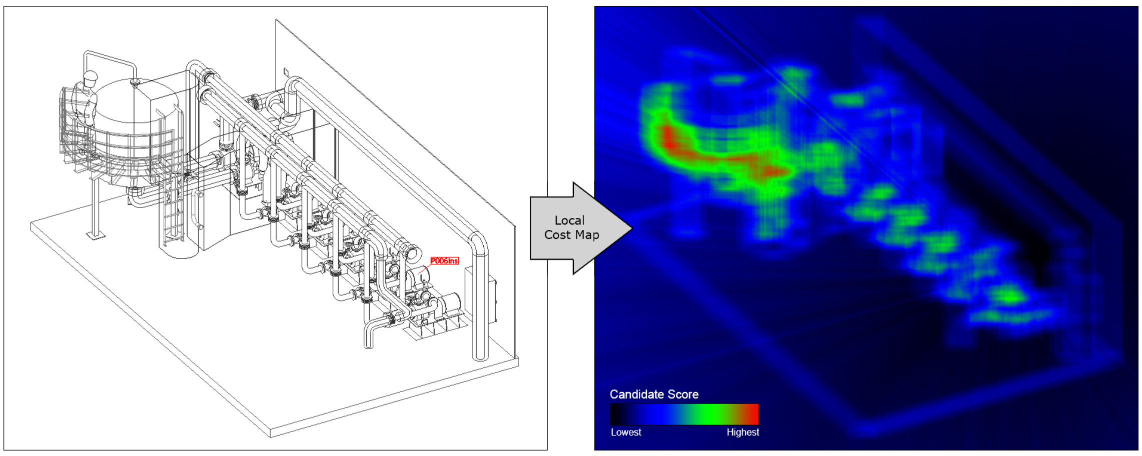

Figure 6.1: Local cost map plotted for a single label

system and how drawings are handled (i.e. are they raster images or vector graphics). For those reasons, the metrics are left relatively vague so that they can be applied in multiple different scenarios.

6.2 Label Candidate Generation

The goodness of a label placement is a subjective matter and it would be pointless to implement a time costly method that finds the absolutely best place when the ”best place” is ambiguous to begin with. That’s why the Luus-Jaakola heuristic described in section 5.2 is used as a random search method to find suitable places for labels. The section 5.2 introduces a modified version of Luus-Jaakola that was specifically tailored for this occasion and thus can be implemented straight as it is. The cost function sums up the local costs described in previous section 6.1. The pseudo code in listing 6.1 describes the basic steps for candidate generation using Luus-Jaakola.

```
/* Parameters */
candidates_amount            // The number of candidadates
random_points_amount         // The number of random points
search_radius                // Search radius
radius_reduction_amount;     // Radius reduction amount
horizontal_distance          // Horizontal distance between candidates
```

```
vertical_distance          // Vertical distance between candidates
max_iterations             // Maximum number of iterations

/* Returns */
candidate_positions[candidates_amount]   // Final label candidates

/* Label Candidate Genration for single label */
for c in (0,...,candidates_amount){
    // Get initial random points from whole drawing
    random_points = get_n_random_points(random_points_amount);
    // Initial solution for a candidate
    candidate_positions[c] = random_points.min(a => cost_function(a));
    for i in (0,...,max_iterations){
        // Get random points around currently selected position
        random_points = get_n_random_points(random_points_amount, search_radius,
            candidate_positions[c]);
        // Get the point with lowest score
        // that is not too close to other candidates
        for j in(0,...,c-1){
            for p in (0,...,random_points.size())
                d = candidate_positions[j] - random_points[p];
                if (abs(d.x) < horizontal_distance && abs(d.y) < vertical_distance){
                    random_points.remove(p);
                }
        }
        min_point = random_points.min(a => cost_function(a));
        // Set as selected point if produces smaller cost
        if (cost_function(min_point) < cost_function(candidate_positions[c])){
            candidate_position[c] = min_point;
        }
        // Reduce search radius
        search_radius = (1 - radius_reduction_amount) * search_radius;
    }
}
return candidate_positions;
```

Listing 6.1: Pseudo code for label candidate generation

The pseudo code gives a general idea on how to implement the modified version of Luus-Jaakola. However, the method could be further improved from what is shown here.

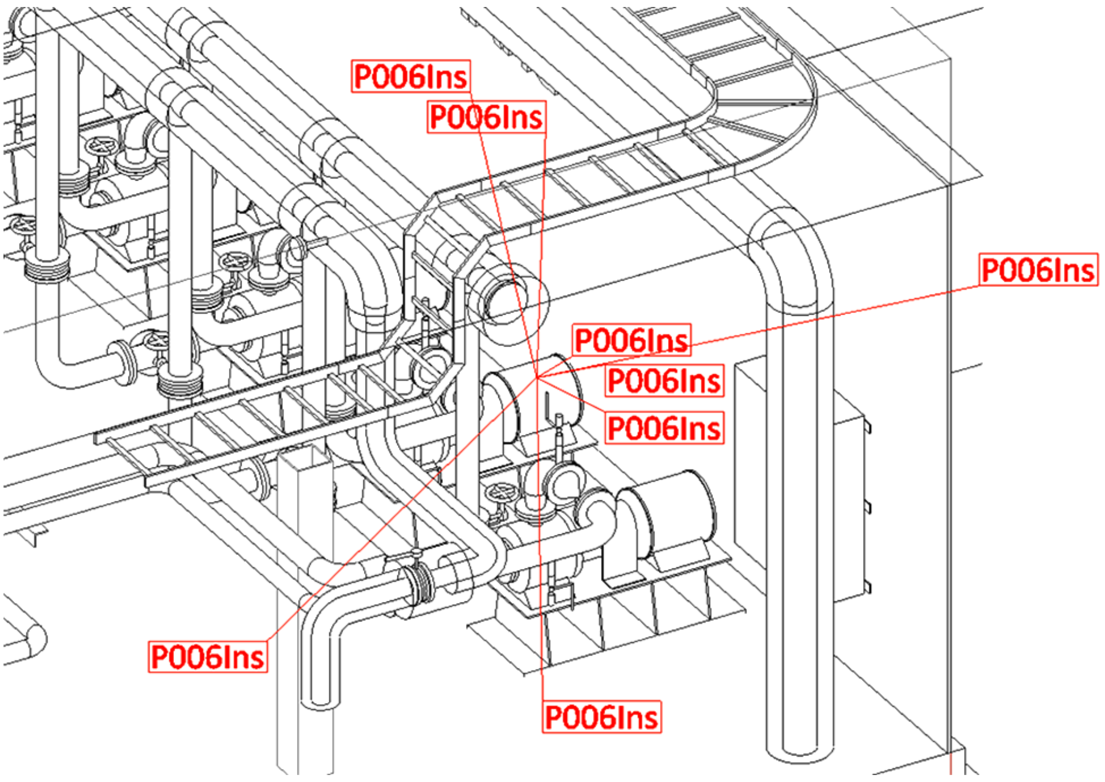

Figure 6.2: Example of eight candidates generated for a single label

The radius could be lowered more if better positions aren't found in a several rounds of iterations or the execution of the iteration step could be stopped entirely in such case.

From experimentation, good values for the parameters are 5 for the candidate amount, 200 for random point amount, 4-8 times the labels' width for the initial search radius, 0.05 for the radius reduction, the label width and height for the minimum distances between labels and around 100 maximum iterations. With these parameters the algorithm still finds good solutions and only takes a fraction of a second for each label. An example of a candidate generation using similar values can be seen in figure 6.2. In the example 8 candidates are generated for a single label. The candidates clearly find decent unoccupied positions even in this relatively difficult looking situation.

6.3 Label Selection

The label selection is probably the most important and significant part of the whole automatic labeling process. It is the objective function that determines the final label configuration by searching for the global minimum for the whole search space. The previous sections dealt with the label placement score and candidate generation which are the ones mostly shaped by the placement requirements. If there's only one label the objective function (i.e. label selection) is trivial, since the choice what candidate to pick is obviously the one with the lowest score. The label selection algorithm is only trying to minimize the total cost by finding an optimal configuration and thus, the selection of the algorithm doesn't depend on the details of the functional requirements. An optimal method would be one that finds the best global configuration in the window of the time requirement.

From the different algorithms introduced in Chapter 4 the simulated annealing (4.1.3) was chosen and implemented as it seemed most promising. It is the only introduced method that is stochastic and able to escape local minimum and thus is more likely to end up in the actual global minimum (or one that is relatively good). The method is also very flexible which is a big bonus due to the time requirement discussed in section 3.4. It's difficult to tell how long the algorithms takes to run until they are tested in real scenarios. Simulated annealing can easily be profiled to fall into a desired time scale.

As the simulated annealing is very profilable approach, not just by the amount and effect of the parameters but also by the implementation choices, it's important to consider all the aspects of the implementation i.e. how to determine the initial configuration, what configuration change method to use and what the annealing schedule looks like. For example choosing the candidates with the best local cost as the initial solution and then a very quick annealing schedule makes the solution fast, but it is less likely that the best minimum would be found.

The two main different initial configurations are random selection and selecting a candidate with the lowest local cost. The random placement is slower, but it avoids the problems of greediness that the best apparent initial solution might fall into. However as Christensen et al. [1] pointed out, different initial configurations don't seem to have significant impact on the final result nor the converge speed. Both of the proposed initial configurations are very simple to implement and could both be easily tested.

The objective function calculates the change (ΔE) in the total global cost after every change according to candidate scoring discussed in section 6.1. The candidates local costs can be calculated during the candidate generation stage as the local costs are invariant to the changing configuration. Only the global cost has to be re-calculated for each label on each step. Then the total global cost is simply the sum of all the selected candidates' global costs. As the changing of labels' selected candidate only affects labels that it was intersecting with and what the new label intersects with, the calculation of the change ΔE in the total global cost can be simplified to only consider these cases.

There are couple of different configuration change methods to consider. The simplest and perhaps the best in final quality is to randomly choose a label and a candidate to reposition. However it can take a bit longer time until all of the good choices are traversed through. Also the randomness nature can forsake a good candidate just by pure luck if it never was even considered. Another way would be to systematically go through the labels and candidates. This again can have similar problems to greedy algorithms because of it being systematic makes certain configurations more preferable just due to the order of selection. One way to avoid shortcomings of both methods is to combine them which also was chosen as the configuration change method for the final implementation. This can be done by first running the algorithm with random selection until a certain threshold of not making progress is achieved and then going through every labels' candidates systematically to make sure all possibilities are tried.

The annealing schedule is a bit more tricky to adjust. It should be decided what the initial temperature T should be as well as how often and how much to lower it. After some experimentation good value for the initial temperature is the difference between the highest and lowest global costs of all the candidates. This way the probability function $P = 1.0 - e^{-\Delta E/T}$ would produce a P of around 0.7 when the change ΔE is around third of the cost range between all candidates. How to lower the temperature depends on the trade-off between quality and performance. If the annealing schedule is fast, the quality also suffers from it. It's difficult to come up with ideal values as the whole method relies on randomness. A good annealing schedule from experiments is something similar to lowering the temperature by 10% of its current value every nth replacement where n is the total number of candidates (candidates per label *times* the number of labels). After no improvements is made in $3n$ iterations, the configuration method is changed from random selection to systematic selection. Otherwise the algorithm is kept running as normal until no improvements has been made for an additional $3n$ iterations.

```
/* Parameters */
labels[]            // Array of labels

/* Returns */
labels[]            // End results stored in labels[n].selected_candidate_index

/* Simulated annealing */
// Initial configuration is selected randomly
for i in (0,...,labels.size()){
    labels[i].selected_candidate_index = random(0, labels[i].candidates.size());
}
// Compute the total global cost
total_cost =  compute_total_global_cost(labels);

// Initial temperature is the difference of highest and lowest candidate costs
lowest_cost = labels.min(lbl => lbl.candidates.min(cnd => cnd.global_cost));
highest_cost = labels.max(lbl => lbl.candidates.max(cnd => cnd.global_cost));
temperature = highest_cost - lowest_cost;

// To determine whether random configuration change method is used
use_random_candidates = true;

// Iterate until improvements haven't been done in a while
max_iterations_without_improvement = labels.size() * labels[0].candidates.size() * 3;
iterations_from_last_improvement = 0;
```

```
while (iterations_from_last_improvement < max_iterations_without_improvement){
    // Start changing the selection
    for i in (0,...,labels.size()){
        for j in (0,...,labels[i].candidates.size()){
            label_index = i;
            candidate_index = j;
            if (use_random_candidates){
                label_index = random(0, labels.size());
                candidate_index = random(0, labels[i].candidates.size());
            }
            // Store the old selection
            old_selection = label[label_index].selected_candidate_index;
            // Select new candidate
            label[label_index].selected_candidate_index = candidate_index;
            // Compute the new total global cost (objective function)
            new_total_cost = compute_total_global_cost(labels);
            // Calculate the cost difference
            delta_E = total_cost − new_total_cost;
            if (delta_E < 0){
                // The change was better so accept it
                total_cost = new_total_cost;
                iterations_from_last_improvement = 0;
                continue;
            // If the change in objective function is basically zero, discard the change
            }else if (delta_E > 0.0001){
                // Otherwise discard the change with probability P
                P = 1.0 − e^(−delta_E/temperature);
                if (P < random(0,...,1)){
                    // accept the change regardless of it being worse
                    total_cost = new_total_cost;
                    iterations_from_last_improvement = 0;
                    continue;
                }
            }
            label[label_index].selected_candidate_index = old_selection;
            iterations_from_last_improvement++;
            // If no improvements has been made,
            // switch to systematic configuration change method
            if (iterations_from_last_improvement >= max_iterations_without_improvement
           && use_random_candidates){
                use_random_candidates = false;
                iterations_from_last_improvement = 0;
            }
        }
    }
    // Decrease the temperature
    temperature *= 0.9;
}
// The labels' selected candidates are now changed
return labels;
```

Listing 6.2: Pseudo code for simulated annealing

The presented pseudo code for simulated annealing in listing 6.2 is a simplified version of what a real implementation would look like. Still, the whole simulated annealing more or less fits into around 30 lines of code (if objective function is not included), which makes it relatively easy to understand, profile and implement. This makes it an attractive method to implement for the candidate selection process.

6.4 Assembling Everything Together

All the pieces of the automatic labeling puzzle are explained in the preceding sections. Remaining task is to combine them into a working solution. It's a good idea to keep the different parts (label scoring, candidate generation and label selection) separate as their own modules. This way it is easy to modify or completely change one part without affecting the others. For this reason another module is required that provides the functionality to the other modules, a service provider. The service provider works as a interface between the different parts to make them independent of each others' specific implementation. To keep things simple the service provider could also work as a master module that controls the flow of the automatic labeling process as well as handles the public interface for clients to use. This master module is called *automatic labeling controller*. The figure 6.3 shows a diagram of the relationships between the different modules.

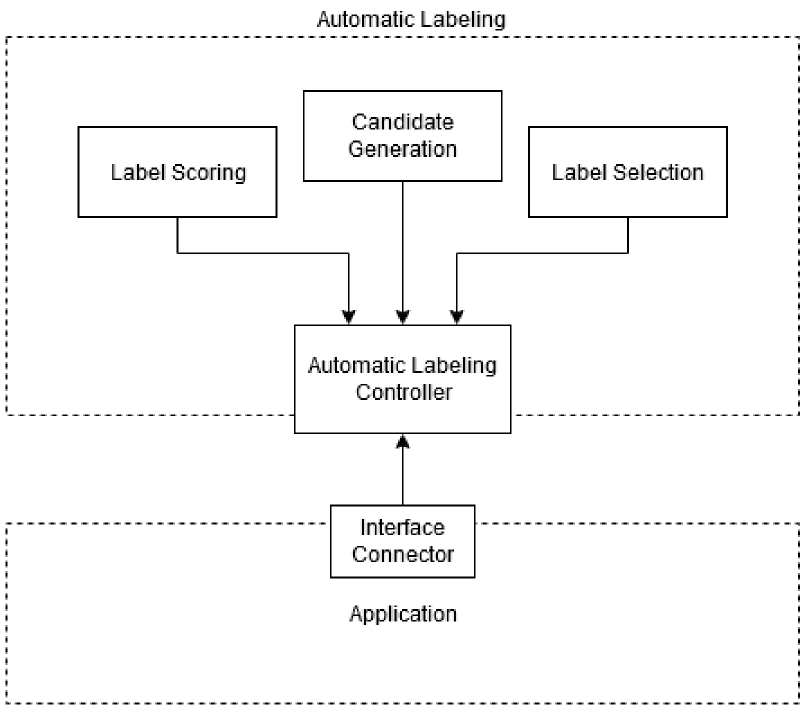

Figure 6.3: Diagram of the different components associated with the implementation

The basic control flow of the automatic labeling process is simple process with two steps: first generate label candidates for each label and then perform label selection to those labels. A bit of an improvement for a very complex drawings is to perform multiple passes of the whole process. In the first pass all of the labels are generated and selected as normal. All labels that don't have intersections or overlaps with other labels are marked fixed. For the rest of the conflicting labels, new candidates are generated taking the fixed labels into account and the candidate selection is performed again for all non-fixed labels while still taking the fixed labels into account in the cost calculations. This can be done multiple times until a configuration without conflicts is found or a maximum number of passes is performed. This way it's more likely that a non-conflicting final result is reached. The control flow for such case could be the following procedure.

1. Set all labels to be non-fixed.

2. Repeat until all labels are fixed or maximum number of iterations is passed.

   (a) Generate new candidates for non-fixed labels.
   (b) Perform label selection procedure without changing candidates for labels that are fixed.
   (c) Set all non-conflicting labels as fixed and conflicting labels as non-fixed.

One thing to consider when starting to implement automatic labeling is the generality of the final solution. A very general solution could be one that takes an image of a drawing, required label reference points and sizes as well as some settings as input and produces final label positions as output. This kind of image based implementation is quite general as it doesn't depend on external drawing formats, systems or protocols. However it has a downside common to general solutions. The applications has to convert its drawing first to an image which takes time and processing an image can be much slower than a specific drawing format. On the other hand this makes it quite flexible as it doesn't depend on specifics but then again, everything has to be done by the solution. That leads to the other possibility: an implementation that is based on a specific drawing format, engine and/or system. There are multiple advantages. The implementation can use already existing functions, there's no need for conversion and usually drawings are stored in some kind of vector format, which can be much faster to process, but it is tied to specific system and lacks generality.

The only module affected by the generality concern is the label scoring. The candidate generation and selection modules use the costs gained from the scoring system to find the optimal places for labels. They don't need any contextual understanding of the drawings content. It wouldn't be impossible to implement both image based and specific system based implementations as only the scoring module would be implemented twice. Though, the scoring system is arguably the most laborious to implement since there's a lot of sub problems to solve like label overlap costs, leader intersections, leader overlaps, etc.. The implementation created alongside this thesis uses only the image based scoring system.

An example of a labeling created with the implementation discussed in this chapter

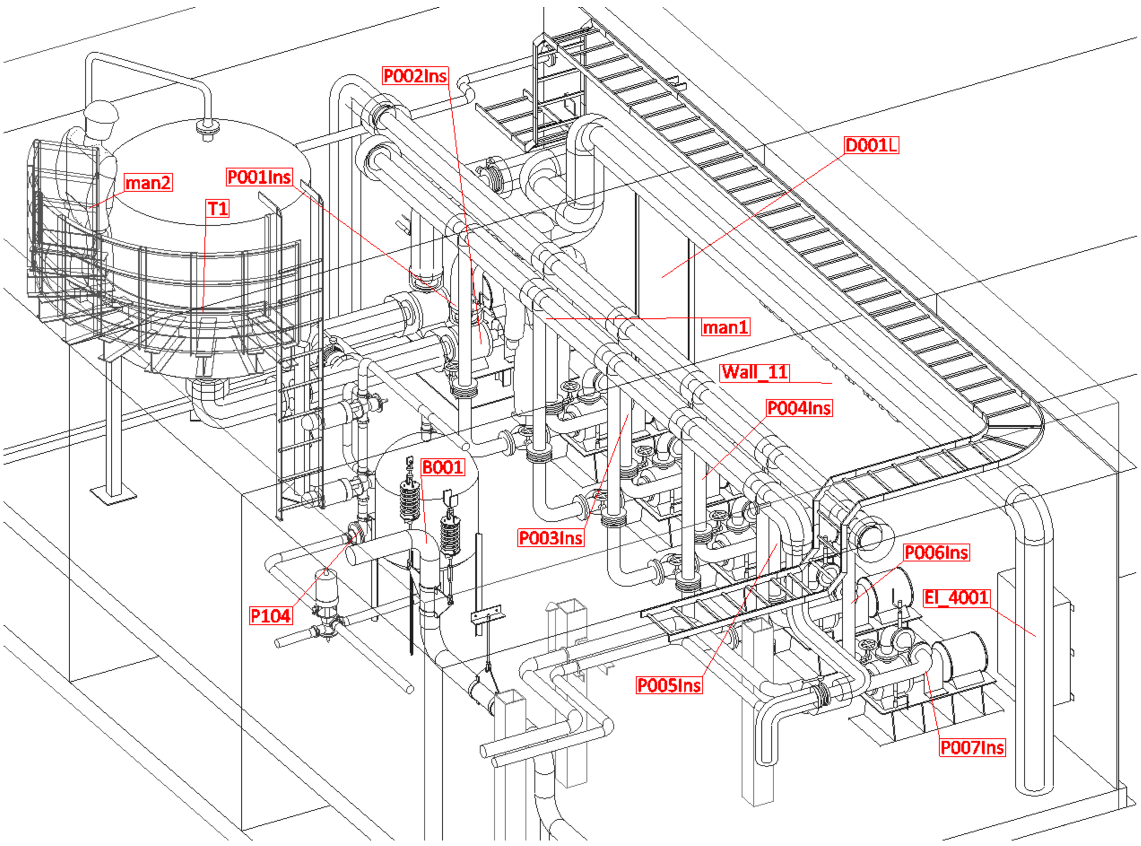

Figure 6.4: Example of a label placement produced by the automatic labeling implementation discussed in this chapter

is presented in figure 6.4. The example is a relatively complex axonometric drawing. It's not very typical drawing to require labeling, but it shows quite well the extents of the automatic labeling solution. Unfortunately, examples of real scenarios could not be provided as they're classified customer data. The next chapter presents the final results of the particular implementation made alongside this thesis in the form of an empirical study.

Chapter 7

Results and Evaluation

This chapter evaluates the results of the automatic labeling solution via an empirical study. In the study, a result of the implementation of the automatic label placement solution to CADMATIC Plant Modeller's drawing functionality is compared with the manually labeled drawings and the performance of the solution is measured. The implementation used in the study is completely based on the one presented in previous chapter. The ultimate goal of the thesis was to develop a solution to automatically label technical drawings that produces good enough results to be used in practice.

The purpose of the empirical study is to evaluate the success of the automatic label placement solution presented by this thesis to the extent that it can be marked as good enough for commercial use. The solution is evaluated by comparing it with a manual labeling. The comparisons purpose is to get an understanding of how close the automatic solution is to manual labeling in terms of quality and how the solution could be developed further. It is not expected to be superior to manual labeling. Because the study was issued by CADMATIC, a commercial company, the result should assess whether further development of an automatic labeling feature should be conducted. Due to label goodness being a subjective matter, a user questionnaire was chosen as the evaluation method. The performance of the solution is evaluated by measuring the run time of the algorithms with different amounts of labels.

7.1 Conducting the Empirical Study

Initial idea for evaluation method was to give the automatic labeling tool to professional user to test and play around with it. However, due to time limitations and the incompleteness of the application side of the feature, that approach was discarded. For the same reason an extensive survey of its capabilities was out of the question. In order to evaluate the solutions quality, a simple questionnaire was conducted. The performance of the solution was evaluated by a simple measurement of its running times.

In the questionnaire three pairs of drawings were presented to the participants. Each pair is the exact same drawing where the other is labeled manually by a field expert and the other one is labeled automatically. The drawings are taken from a real construction project and each pair has a different level of detail and varying amount of labels. The first drawing is a small inlet drawing with ten labels. The second drawing is a bigger inlet drawing with around a hundred labels and the third one is a piping arrangement drawing with around forty labels. As the drawings are from CADMATIC's customer, participants from outside the company were unable to attend the study. For that reason, only nine participants who were all CADMATIC employees completed the questionnaire. Despite the fact that the participants were CADMATIC (software company) employees, they all had at least some experience in working with similar drawings in the industry.

The questionnaire clearly stated that only the spatial position and arrangements of the labels should be evaluated. The questions themselves were quite simple. For each pair of drawings the participants were asked which of the two given drawings looks better when considering the spatial position and arrangement of the labels. A detailed justification was also requested from the participants. Comments about the readability and ambiguity of the labeling were specifically asked since those were the main requirements for the solution. The questionnaire also had a free-form comment field for general comments, ideas and concerns regarding automatic labeling.

For evaluating the performance of the solution for how long it takes to automatically label a drawing, automatic labeling was conducted multiple times to a single drawing with varying amounts of labels. The results were plotted by the amount of labels as a function of time to get a rough idea of the time complexity of the solution.

7.2 Results of the Empirical Study

This section analyses the results of the empirical study. The results of the questionnaire are reviewed and the solutions time performance profile is presented.

7.2.1 Time Complexity

The run time of the automatic labeling is one of the requirements discussed in Chapter 3. To evaluate the solution's success when considering the time requirement, a performance test was performed. In the test, a drawing was automatically labeled multiple times with varying amount of labels. The labels were placed semi-randomly in order to avoid systematic errors or errors related to clustering of labels. This of course is not a realistic scenario when considering actually running the algorithm in practice. But the actual performance should not differ that much from a random placement that it would deprive the credibility of the tests.

The range of label cont was from 2 to 150 and twenty runs of the automatic labeling was run with different amounts of labels. Only one pass of the labeling was run for each case and eight candidates was generated for each label. The total time of the candidate generation as well as the label selection was recorded for each case.

The end result of the performance tests can be seen plotted in figure 7.1. The plot shows that when label amount is below 35 the time for candidate generation is the most significant but after that the label selection takes over. The run time of the candidate generation seems to be linear as expected since the process takes the same amount of time for one label regardless of how many other labels there are. It takes only the local cost of

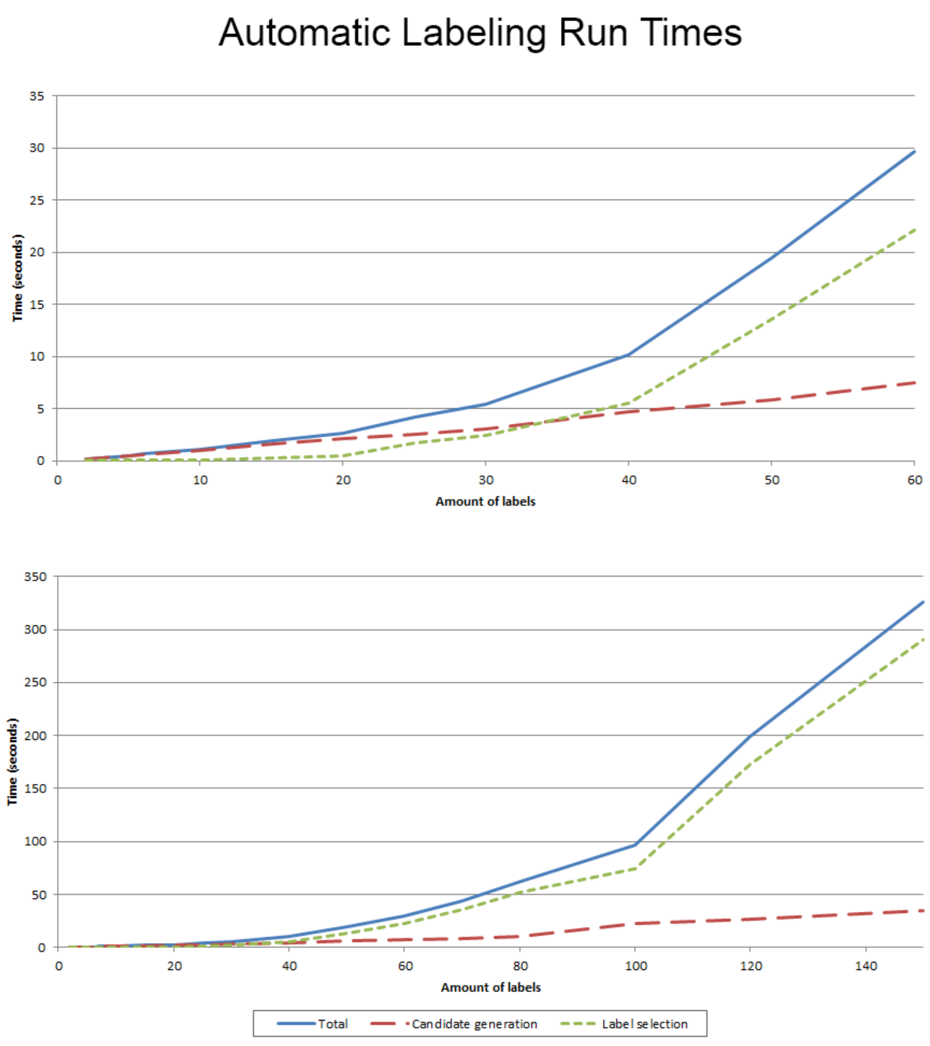

Figure 7.1: Running times of the automatic labeling for different amounts of labels

the candidate into account after all.

The actual label selection algorithm behaves a bit differently. Even though the complexity of the general label placement problem is proven to be NP-hard [3], the label selection is still polynomial. Although the solution is based on probabilities, the probability to make worse choice is lowered each iteration. There is a fixed amount of iteration until the probability drops low enough for its floating point representation to be considered zero. As otherwise the algorithm consist of just nested loops, the algorithm converges in polynomial time which the figure 7.1 seems to comply with.

The solution's run time is in the frame of the requirements. The requirement was that the label placement should take from few seconds to around a minute with reasonable amount of labels. The minute threshold is exceeded at around 80 labels, which is quite many for a single drawing. There is still a lot of optimization to be done on the application side but the algorithms themselves are more difficult to improve time-wise without making compromises.

Because the label selection is the most time consuming part of the solution when dealing with a lot of labels, the simulated annealing could be swapped with a greedy method to speed up the run time of the solution. This of course can affect the quality of the labeling, but if time is the more significant factor of the label placement, it can be a considerable alternative. After all the greedy methods discussed in Chapter 4 were reported to be 10-100 times faster by Christensen et al. [1] in their empirical study.

Noteworthy is that the performance test used only a single pass of the automatic labeling. In practice, especially in very cramped drawings, it would be better to run multiple passes of the algorithm if the result had overlaps or intersections of the labels. This would multiply the amount of time it takes to run the whole labeling process and the true run time would also be much higher. However, with moderate amount of labels (40) usually a single pass is enough to result in a non-conflicting labeling.

7.2.2 Questionnaire Results

Even though the questionnaire was quite short and there were only nine participants, it still gave a lot of valuable information of the success and shortcomings of the implementation. The figure 7.2 shows the multiple choice questions results about which of the two drawings, manually and automatically labeled, is better. As the purpose for the automatic solution is not to be better than manually performed labeling, the answers were not expected to indicate that either. The purpose was to compare the different labeling methods and to understand to which extent the automatic labeling is inferior. The participants did

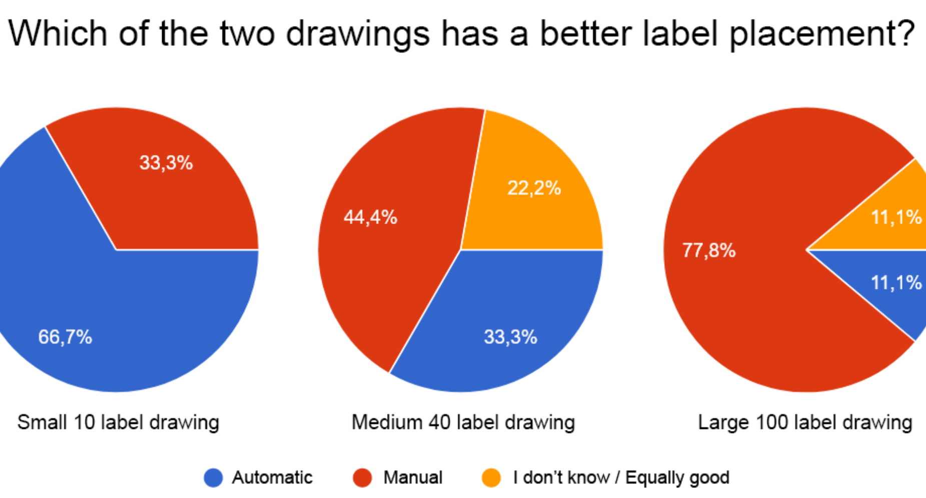

Figure 7.2: Answers of the questionnaire about which of the two drawings had better label placement.

not know which of the two drawings was done by the automatic labeling. This way the participants wouldn't have any assumptions and the comparison would be fair.

Because the drawings used in the questionnaire were from a customer, the particular comparisons cannot be shown. In order to demonstrate the differences, a separate comparison drawing was made that is similar to the medium-sized drawing used in the questionnaire. This example comparison can be seen in figure 7.3. The manual labeling was done by an inexperienced software developer, which might make the labeling to be worse than if it was made by an experienced designer from the industry. Two other examples are also shown (figures 7.4 and 7.5) that try to demonstrate the differences that appeared in the questionnaire drawings.

The results were quite positive. In the small ten label drawing, surprisingly two thirds of the answers favored the one produced by the automatic labeling. There were multiple detailed arguments for both of the drawings. The manual labeling had a little longer leader lines as it clearly tried to align and group the labels according to their feature's contextual content. Some participants valued the alignment more while others valued the

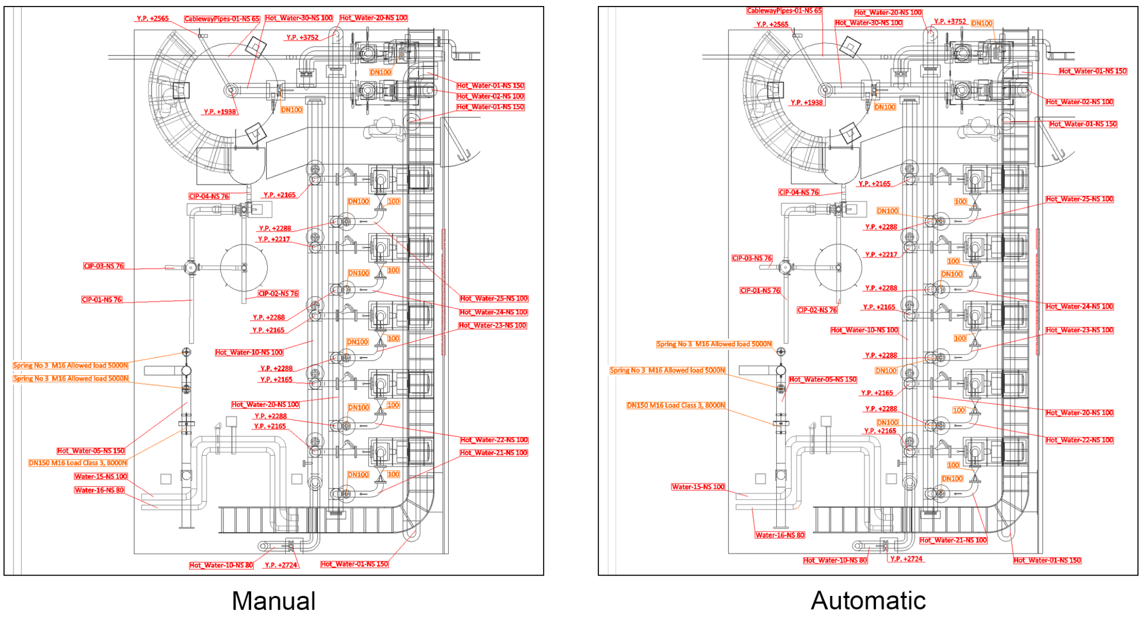

Figure 7.3: An example comparison of manual and automatic labeling.

closeness of the label. The automatic labeling left a bit wider margin between the labels which was valued by some participants as it made the labels more readable and a bit more ambiguous. The conflicting opinions shows the subjective nature of evaluating the goodness of a label placement. Even with such simple drawing and a small subject group, conflicting opinions emerge.

The bigger drawings however show that manual labeling is still superior. The manual labeling was slightly favored by the participants in the medium sized drawing. Four favored the manual, three were on the automatic's side and two thought they were equally good. The only argument against the automatic labeling was the leaders proximity or total overlap of dimension lines which the automatic algorithm does not prioritize any more than any other graphics. Figure 7.4 shows an example of labels overlapping with dimension lines. This little detail appeared to be quite significant as a lot of the answers commented on it saying it was the deciding factor between the two drawings. Otherwise the actual placement of the labels was quite on par with the manual labeling according to most of the answers. The usage of automatic labeling in this type of drawing seems very

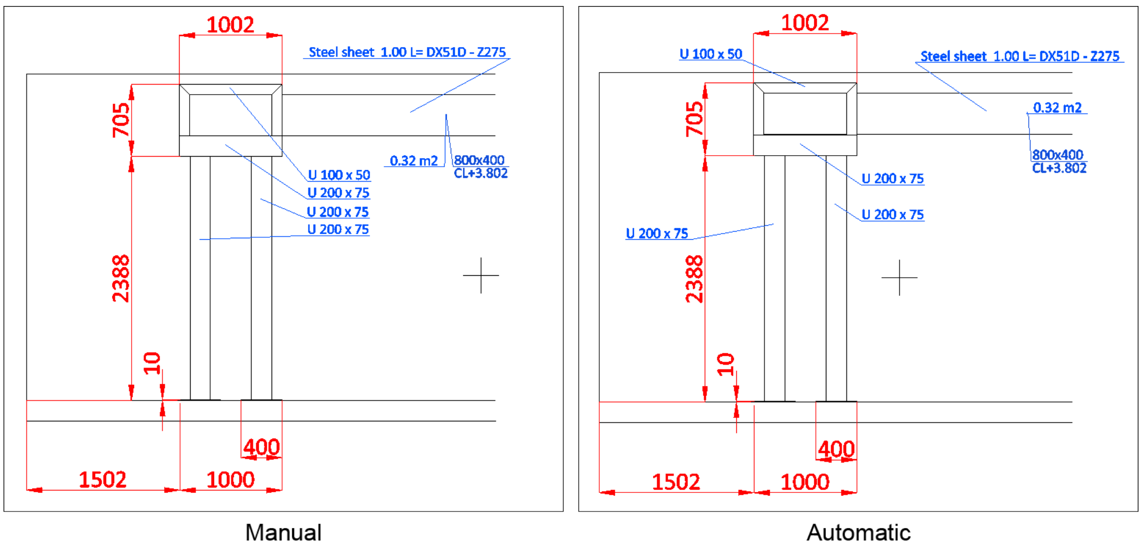

Figure 7.4: An example labels overlapping with dimension lines in automatic labeling

feasible if the dimension lines are taken more into account by the label scoring system.

The automatic labeling did not fare so well against the biggest drawing with over a hundred labels. Still, having one of the nine participants favour the automatic labeling and one thinking they're equally good indicates that the gap is not that big. The biggest difference of the drawing was the space available. The medium drawing was much more cramped with other graphics and the labels were quite far away from each other. The labels usually had quite ambiguous best position, which the automatic algorithm was able to find. In the biggest drawing there were much more space for each label. They had plenty of similarly good candidates to choose from. This made the candidate selection rather random, as some labels were placed above their feature while other were on the side or under them. This makes the label placement to look very fragmented and almost random. The manually labeled counterpart clearly favoured the north-eastern side of the feature to place the label to. When most of the labels are on the same side of their feature the drawing looks much more structured and uniform. An example of similar differences regarding the spatial distribution is shown in figure 7.5. The automatic labeling had similar drawback in this case as in the previous one. The labels overlapped with some graphics

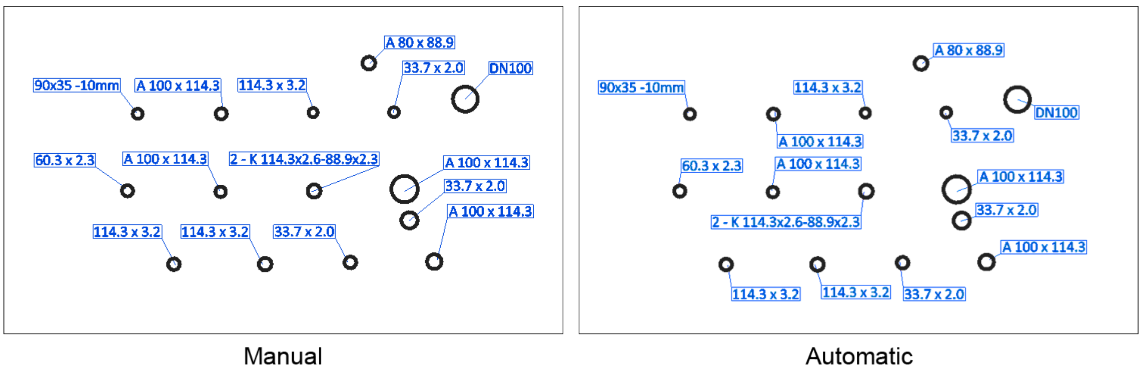

Figure 7.5: An example of differences in spatial distribution between manual and automatic labeling. The manually placed labels are all positioned on top of their features making the drawing look more structured and uniform.

that shouldn't be obscured. Another point regarding label ambiguity that was brought up was axis aligned leaders. A lot of the lines in the drawing are axis aligned and if a leader goes close to a line like that, it can be difficult to distinguish them from each other making it ambiguous where the leader points to. Making the labels favour positions with a leader closer to 45° from the axis's might help. Also better grouping of labels in the manual labeling was brought up in multiple answers.

Overall, a decent amount of information to compare the manual and automatic labeling was received from the questionnaire. Even though such small-scale study cannot be considered very accurate, it can definitely be used to improve upon the first iteration of the automatic label placement implementation. The results also indicate that the automatic label placement solution is a viable option to manual labeling, especially with a bit of improving.

7.3 Final Results

The empirical study's purpose was to verify if the presented solution is practical enough to be used in real environment to replace at least some of the manual labeling done to technical drawings. The results indicate that the solution already is decent but with few improvements to the requirements and implementation specific details, the solution could prove to be very useful. The performance of the solution has potential for improvement but those improvements are implementation specific details and not direct improvements to the presented heuristic frames. As the time complexity test showed, the label selection is the most time consuming part when a lot of labels needs to be placed. If the slowness of the solution becomes an issue, there are viable alternative greedy algorithms that can be much faster with a compromise in quality.

According to the empirical study, the automatic label placement requirements could be revised to include the issues brought up. The labels could have a bit of a margin in order to not be too close to other graphical features or each other. The requirements indicate that labels should not obscure other graphical features of the drawing. While this is true, it's often impossible to avoid all other graphical features and a compromise is made what features to obscure. The implementation should take into account not only the labels readability but other objects as well. Thus, the requirement should be revised to indicate that different graphical objects should be prioritized. For example overlap with text reduces the readability of the drawing more than with a structural line. Another thing brought up in the questionnaire was the spatial distribution of the labels. Especially favoring one side of the feature in a relatively sparse drawing makes the overall labeling to look more structured and uniform. Also labels with axis aligned leaders were disliked. A requirement to favor a certain direction that is not axis-aligned could improve the label placement in certain situations. This kind of requirement is specific to situations with more space and it should not be prioritized over any other presented requirements. The grouping of labels was also mentioned which already was part of the requirements but was regarded as a bonus due to its complexity. It was not implemented in the final solution because of its ambiguous description, but it is something to consider when improving the solution in the future. The other issues and suggestions are simple enough to include in the updated requirements. The new revised requirements are presented in table 7.1.

Improved Requirements	
R1. Readability	Labels should not obscure other graphical features according to their prioritization factor and they should have a decent margin around them.
R2. Unambiguity	Labels are connected to their features with leaders and they should be as close together as possible. The labels should avoid places which results in axis-aligned leaders.
R3. Time	The run time of the algorithm shouldn't take more than few seconds for reasonable amount of labels.
R4. Aesthetic considerations (bonus)	The labels distribution shouldn't be too scattered or uniform and they should be aligned to groups. Labels should favor a uniform direction from their feature.

Table 7.1: Improved requirements for automatic labeling in technical drawings

The section 6.1 discussed that the requirements are mostly reflected in the label scoring. The changes in the requirements should also be reflected in the scoring metrics. The table 7.2 presents the improved label scoring metrics.

Note that by expanding the metrics for the cost calculation, the run time of the solution is also affected by the new calculations it has to do on each iteration. The local costs don't affect the run time nearly as much as they are only computed in the candidate generation phase which was proven to be the least significant time consumer in section 7.2.1. The extension of global cost slow the solution down the most, but if the change improves the labeling quality enough, it is justified.

Local Costs	
Label and leader overlap with graphics	The total area of overlap with graphical features weighted with the importance of the overlapping features.
Label distance from its feature	Label's Euclidean distance from its associative feature.
Label margin overlap with graphics	The total area of margin overlap with other graphics.
Avoid axis aligned leaders	Angle between label's leader and main axis's.
Global Costs	
Label overlap with other labels	Total area of overlap with other selected labels.
Leader intersecting with other leaders	Number of intersections a label's leader has with other selected labels' leaders.
Leader overlapping with other labels	Length of the leader's overlapping segment with other selected labels.
Label's margin overlap with other labels	Area of label's margin overlapping with other labels.
Uniformity of labels' directions	Either leader's angle closeness to a predefined direction (can be considered local cost) or the total dispersion of the labels' directions.

Table 7.2: Improved metrics for label cost calculations.

A robust solution to reliably group labels into structured units could improve the labeling drastically. It was after all a requirement discovered in Chapter 3 and it was brought up multiple times in the questionnaire of the empirical study. However, the grouping would require large and quite fundamental changes to the whole solution, as the grouping should be taken into account in the label candidate generation phase. The labels should be aware of each other so they could pick candidates that would be aligned with each other. One idea to incorporate grouping into the solution could be to first analyse the drawing for possible places for such groups. Then in the candidate generation phase the labels would be offered placements in those groups as an alternative for a regular candidate position. The groups would work as their own entities that can decide the order of the labels included. The grouping would then somehow be run at the same time as the label selection. This is a bit of speculation though and the subject requires another study to research the possibilities further.

Chapter 8

Conclusion

This thesis aimed to develop a robust solution to automatically label technical drawings. By collecting a set of general requirements a solution was developed that fulfills them. As the prior research has mainly concentrated on automatic map labeling, a new method for generating label candidates had to be created. Surprisingly, the functional requirements only affected the implementation of label scoring which drives the label candidate generation to the right direction. By using the developed candidate generation algorithm, simulated annealing and a label scoring based on the requirements, an implementation frame of an automatic labeling solution for technical drawings was presented. The solution was evaluated by conducting an empirical study on the implementation made for CADMATIC's CAD-software. The study aimed to verify the success of the solution in terms of labeling quality and run time performance. With the results, the research questions set up by this thesis can be answered.

1. What are the requirements for a practical label placement solution for technical drawings?
2. What kind of automatic label placement solution fulfils those requirements?

The initial answer for the first question was formed in Chapter 3 which studied general label placement metrics and researched possible industry standards and customs that could affect the requirements for the label placement. The initial requirements were collected to table 3.1 in Chapter 3 that summarises all the rules the final solution should fulfill. The requirements were further improved with the results from the empirical study discussed in Chapter 7. The requirements table was expanded to include the issues and ideas that the study revealed. The final requirements and the answer to the first question were summarised in the new requirements table 7.1 in Chapter 7.

The rest of the thesis strove to answer the second question. To develop a solution that fulfills the requirements and is practical in industrial use. The Chapter 6 presents the solution as an implementation frame and its success was evaluated and the design improved in the results Chapter 7. The empirical study verified that the solution complies with the requirements and is practical enough for industrial use.

The solution fulfills the time requirement barely (from few seconds to around a minute of run time for moderate amount of labels). A lot of optimization could be done in the application side in terms of performance, but also the label selection algorithm could be swapped with a greedy one. That would reduce the quality the solution produces but make it considerably faster. The quality of the labeling the solution produces was deemed relatively good, though a few notable issues and ideas for improvement were brought up:

- The labels' readability would benefit from a margin around the labels.
- The labels should avoid axis aligned leaders.
- A uniform distribution of the labels direction from their features would make the labeling more structured.
- Aligning the labels into groups would make the drawing aesthetically more pleasing.

The table 7.2 takes these issues into account as a new and improved guideline for label cost calculation (excluding the grouping).

As this thesis specifically targeted technical drawings in plant and marine industry, the generality of the requirements, solution and results might be a concern. The Chapter 3 discussed the requirements and discovered that there really aren't any industry standards or practises that the solution should consider other than the fact that labels should be connected to their features with leaders. Also the time requirement is specific to the application. Therefore, the solution is general to label any drawings or images that require labels that are connected to their features with leaders and where the performance of the solution is enough.

The grouping of labels is a subject that could definitely be researched more in the future. The grouping was an item in the original requirements but was discarded as being too complex for the scope of this thesis. The issue was also brought up in multiple answers in the empirical study's questionnaire. Thus, incorporating a grouping functionality to the presented solution could improve the labeling quality drastically.

Another issue that could be studied in the future is replacing of the simulated annealing as the objective function. Even though it produces labeling with high quality, it is a bit slow. A method that is faster but produces quality comparable to the simulated annealing would benefit the presented solution.

The empirical study's questionnaire showed that the automatic labeling is not on par with manual labeling when dealing with a lot of labels (100+). When the features are cramped into a small area, the close neighbourhood simply doesn't have enough space for all the labels. The solution fails to consider placement far away from the cluster of features which would reduce the amount of conflicts between labels and improve the overall labeling quality. These scenarios could be studied in the future as they are not uncommon in plant and marine industry. Being able to automatically label drawings that are extremely laborious would considerably reduce the amount of time designers take to produce drawings.

References

[1] Jon Christensen, Joe Marks, and Stuart Shiebe. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphicsn*, 14(3):203–232, 1995.

[2] Alexander Wolff. The map-labeling bibliography. `https://i11www.iti.kit.edu/map-labeling/bibliography/` (accessed: 06 / 2020).

[3] Alexander Wolff. *Automated Label Placement in Theory and Practice*. PhD thesis, Free University of Berlinn, 1999.

[4] Frank Wagner, Alexander Wolff, Vikas Kapoor, and Tycho Strijk. Three rules suffice for good label placement. *Algorithmica*, 30:334–349, 2001.

[5] Konstantinos Kakoulis and Ioannis Tollis. Algorithms for the multiple label placement problem. *Computational Geometry*, 35:143–161, 10 2006.

[6] Konstantinos G. Kakoulis and Ioannis G. Tollis. Labeling algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 489–515. Chapman and Hall/CRC, 2013.

[7] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215 – 236, 2007.

[8] Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. pages 310–319, 01 2010.

[9] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13:289–317, 2009.

[10] Missae Yamamoto, Gilberto Câmara, and Luiz Lorena. Fast point-feature label placement algorithm for real time screen maps. *GEOINFO 2005 - 7th Brazilian Symposium on GeoInformatics*, 01 2005.

[11] H. Schumann, M. Luboschik, and H. Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization & Computer Graphics*, 14(06):1237–1244, nov 2008.

[12] Gregory Cipriano and Michael Gleicher. Text scaffolds for effective surface labeling. *IEEE transactions on visualization and computer graphics*, 14:1675–82, 01 2009.

[13] Tomáš Chamra. Algorithms for automatic label placement. Master's thesis, Czech Technical University in Praguen, 2017.

[14] Rein Luus and T.H.I. Jaakola. Optimization by direct search and systematic reduction of the size of search region. *AIChE J.*, 19(4):760–766, 1973.

[15] Konstantinos G. Kakoulis and Ioannis G. Tollis. Labeling algorithms. *Handbook of Graph Drawing and Visualization*, pages 489–513, 2013.

[16] Knut Hartmann, Timo Götzelmann, Kamran Ali, and Thomas Strothotte. Metrics for functional and aesthetic label layouts. In *Smart Graphics*, 2005.

[17] Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. On the readability of leaders in boundary labeling. *Information Visualization*, 18(1):110–132, 2019.

[18] Stephen A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[19] Steven Zoraster. Integer programming applied to the map label placement problem. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 23:16–27, 10 1986.