Cancer Type Classification Using a Logic Formulation

Hendrik Schmitt

A thesis presented for the degree of
Bachelor of Science

Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany
9th January, 2022

Acknowledgments

I would like to express my gratitude towards Prof. Dr. Gunnar Klau for giving me the opportunity to write about this interesting topic. Next, i would like to thank Prof. Dr. Stefan Harmeling for being my second assessor. Furthermore, a special thanks goes to Nguyen Khoa Tran for his advice in the weekly meetings, his patience with my many questions and for giving me fast and extensive feedback on my writing. Finally, i would like to thank the Center for Information and Media Technology at Heinrich Heine University Düsseldorf for providing the computational infrastructure.
Thanks.

Abstract

Cancer type classification based on tumor DNA can aid in cancer research and diagnostics. In modern medicine machine learning algorithms are used to create models that can be used for predicting the cancer type of a tumor DNA sample. Unfortunately, most of these machine learning algorithms produce black box models that do not explain why a classification was made. To overcome this issue the machine learning algorithms "LOCATOR" and "LOBICO" produce explainable logic models with high credibility and interpretability. The logic models consist of Boolean logic formulas in disjunctive normal form (DNF), with each literal standing for the presence or absence of a specific gene mutation. Both algorithms produce a DNF formula for every cancer type and if the DNF formula is true for a specific tumor DNA sample, the sample gets classified as the corresponding cancer type.

For training both models we use data from the online genom database cBioPortal. The dataset includes 6640 tumor DNA samples, each having 13150 features. To reduce the dimensionality of the dataset we use recursive feature elimination for linear support vector machines (SVM-RFE).

LOCATOR and LOBICO use an integer linear program (ILP) for finding the best DNF formula for each class. In this thesis we reformulated the ILP as a Boolean satisfiability problem (SAT) which can produce the same DNF formulas as the ILP. SAT solvers have become faster over the last years and thus with the SAT formulation we might be able to create the logic models faster and more efficiently.

In the course of this thesis we introduce five different SAT formulations each having a different complexity. Compared to the first SAT formulation, we were able to reduce the formulation's complexity by more than 85 % in our best formulation. For LOCATOR even with this reduction we were not able to produce logic models for the whole dataset (including all 6640 DNA samples) in a reasonable time compared to the ILP. As LOBICO's formulation is more complex by its base structure, the ILP and SAT formulations were not able to produce models for the whole dataset as well.

In experiments with a reduced number of DNA samples and limited computational resources we observe that the LOCATOR SAT formulation has a higher runtime than the LOCATOR ILP formulation, with the exception of some special cases. For LOBICO we observe that the SAT formulation constructs the models quicker across all tested cases, but as the ILP can use multiple CPUs, the advantage is only present when using limited computational resources.

In conclusion, the LOCATOR ILP formulation is more useful than the SAT formulation as it can handle bigger datasets and needs less time, whereas the LOBICO SAT formulation can be more useful than the ILP, especially when having only limited computational resources available.

Contents

1 Introduction

Cancer is one of the biggest challenges modern medicine has to face. With almost 20 million cases in 2020 and approximately 10 million deaths [1], cancer is one of the most frequent diseases in the world with a high mortality. Since tumors often develop from mutations in the DNA, the diagnosis and therapy can be done by evaluating the patient's gene mutations. However, DNA is very long and contains a lot of information. For that machine learning techniques can be used to evaluate the patient's DNA. These algorithms are good at classifying tumor DNA but in most cases they are like a black box, not explaining why a classification was made. A machine learning algorithm that produces more explainable models could help, so that we can understand the complex patterns and make progress in cancer diagnosis and therapy.
The two algorithms, "Logical Cancer-Type Predictor" (LOCATOR) [2] and "Logic Optimization for Binary Input to Continuous Output" (LOBICO) [3] are two examples of new machine learning approaches that construct more explainable logic models with a high credibility. LOCATOR and LOBICO require a dataset where each row stores one DNA sample, one column the samples class label and all other columns the absence (0) or presence (1) of a specific gene mutation as input (training dataset). The generated logic models consist of a Boolean logic formula in disjunctive normal form (DNF) for each cancer type. Furthermore, the DNF formulas consist of literals, each representing a somatic mutation in a certain gene. After LOCATOR or LOBICO computed the logic model based on known data it can be used to classify unknown data. A small example DNF formula for the cancer type "Breast invasive carcinoma" is given in the following:

$$\text{LMO7DN.n} \wedge \text{SAP30BP.p} \wedge \neg\text{DUXA.n} \tag{1}$$

The formula (1) is just one DNF formula from the generated logic model and can be used to classify unknown DNA samples based on their gene mutations.
In their current implementation both algorithms use integer linear programming (ILP) for finding the best DNF formulas for each cancer type. In the scope of this thesis we want to propose a new approach of finding the best DNF formulas. Over the last years the Boolean satisfiability problem (SAT) has become more popular as SAT solvers have become more efficient at solving SAT instances. Because of this we want to reformulate the ILP formulation as a SAT formulation. As LOCATOR and LOBICO also include Boolean logic, the idea is that a SAT formulation can improve processing time and computational efficiency. This might make the two algorithms even more useful and practical. The ILP formulations of both algorithms lead to good results regarding their classification performance such that we want to construct the SAT formulations as close as possible to the current ILP formulations.
In Section 2, we give a short introduction to the theoretical background knowledge that is needed to understand this thesis.
Section 3 starts by giving an insight into the workflow which we use together with the algo-

rithms. Furthermore, we explain the ILP formulations and their corresponding SAT formulations for LOCATOR and LOBICO.

We present our results in Section 4, where we evaluate the ILP and SAT formulations regarding their runtimes using a dataset from cBioPortal [4]. Additionally we take a close look at how the runtimes of different SAT solvers compare and how the runtimes of our SAT formulations are composed.

We give a critical reflection on our research in Section 5 and summarize our achievements in the last section.

2 Preliminaries

In the following sections we will give an introduction to the background knowledge that is needed to understand the thesis.

2.1 Boolean Logic Formulas

Boolean logic formulas consist of variables $x_1, x_2, ..., x_n$, which can either be True or False. Each variable itself is defined as a formula inducing that the set of variables is a subset of the set of all formulas. Given two formulas ϕ_1 and ϕ_2, the expressions $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$ and $\neg\phi_1$ are formulas as well. The symbols $\wedge$(conjunction), $\vee$(disjunction), $\neg$(negation) and $\Longrightarrow$ (implication) are logical operators with the following properties. Given variables ψ_1 and ψ_2:

ψ_1	ψ_2	$(\psi_1 \wedge \psi_2)$	$(\psi_1 \vee \psi_2)$	$\neg\psi_1$	$\psi_1 \Longrightarrow \psi_2$
F	F	F	F	T	T
F	T	F	T	T	T
T	F	F	T	F	F
T	T	T	T	F	T

Table 1: Properties of logical operators in boolean formulas. F equals False and T equals True.

More than these four logical operators exist but they are out of scope for this thesis.

2.2 Disjunctive Normal Form and Conjunctive Normal Form

The disjunctive normal form (DNF) and conjunctive normal form (CNF) are special forms of how boolean variables are connected inside a Boolean logic formula. Boolean variables or the negation of a variable are called literals. We define a minterm as a formula, where all literals are connected with conjunctions. The minterm evaluates to True, when all of its literals evaluate to True. A formula in DNF combines the minterms with disjunctions and is True, when one of its minterms evaluates to True.
The following example is in DNF.

$$(x_1 \wedge \neg x_2 \wedge x_3) \vee (x_4 \wedge x_1 \wedge \neg x_5)$$

A possible solution for the formula is:

$$x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{True}, x_4 = \text{True}, x_5 = \text{False}$$

For the CNF, we define a clause as the disjunction of literals. A clause evaluates to True if one of its literals equals True. A formula is in CNF if all clauses are connected with conjunctions. The CNF formula evaluates to True if all of its clauses evaluate to True. The following formula is in CNF.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_5 \vee x_3)$$

One possible solution of the formula is:

$$x_1 = \text{True}, x_2 = \text{False}, x_3 = \text{False}, x_4 = \text{False}, x_5 = \text{True}$$

Both special forms are important since every Boolean logic formula can be written as an equivalent DNF or CNF formula.

2.3 Data

The dataset that was used for this thesis was also used by Nguyen Khoa Tran [2] in his masters thesis and originally provided by Soh et al. [5]. It can be downloaded from cBioPortal [4]. The dataset has 6640 rows, each representing a tumor DNA sample and 13152 columns. The first column of every row stores the tumor DNA sample‘s name and the last column a number ranging from 1 to 28, which labels the cancer type. All other columns either store a zero or a one for the absence or the presence of a somatic mutation. As in the thesis by Tran [2] the class labels were changed from 1, 2, 3, . . . , 28 to 0,1, 2, . . . , 27. The following table shows an extract from the dataset:

A1CF	A2M	...	ZSWIM7.p	ZSWIM7.n	classLabel
0	0	...	0	1	0
0	0	...	0	0	0
⋮	⋮	⋱	⋮	⋮	⋮
0	0	...	1	0	27
0	0	...	0	0	27

Table 2: Extract of the dataset.

The dataset includes two types of somatic mutations. The somatic point mutations (SPMs) and the copy-number alterations (CNAs). As the name infers SPMs are mutations of a single base in the DNA sample whereas CNAs are amplifications or deletions of parts in the DNA sample [2]. Furthermore, one can see that the first row of the dataset stores an identification for the column. Soh et al. [5] named the columns after a specific schema. All column names store the genes name, the ones that are extended with a ".p" or ".n" are CNAs, where the ".p" refers to an amplification and the ".n" to a deletion in the gene. Additionally, columns without extension store the genes name and provide the absence or presence of SPMs.

2.4 Support-Vector Machine

Support-Vector Machine (SVM) is a supervised machine learning algorithm used for classification or regression problems. The idea is to train the model on a dataset with known class labels and then use the model to classify data with unknown class labels. For SVMs each data point is n-dimensional, where n refers to the number of features. While training the model

a hyperplane is placed which tries to separate the data points according to their class labels. This means that the number of correctly classified data points by the separating hyperplane should be maximized. The hyperplane is described by $(n-1)$ linearly independent direction vectors and a support vector, which is the closest point to the hyperplane. A SVM can be configured with a regularization parameter C, which for larger values increases the distance from the hyperplane to its closest point. Increasing or decreasing the distance between the support vector and the hyperplane can influence the classification of known and unknown data points, thus there is no general rule for setting the parameter. As described in Section 2.3 our data has 28 different class labels which means one hyperplane is not able to classify all the data correctly. To overcome this issue we use the same strategy as Soh et al. [5] called one-vs-all, which fits a hyperplane for each class and separates it from all other classes. Having trained the SVM model, unknown data points can be classified by their location in the vector space. A linear SVM is defined as a SVM, where all hyperplanes are linear. In this thesis only linear SVMs will be used,see Figure 1.

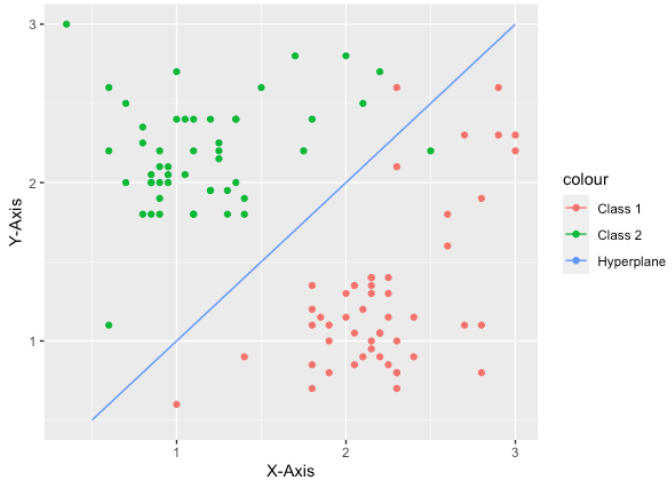


Figure 1: Example of a linear SVM with two different classes and one separating hyperplane. The data points have 2 features x and y, thus are 2-dimensional.

2.5 Recursive Feature Elimination

Recursive Feature Elimination for linear Support-Vector Machines (SVM-RFE) is a technique which can be used for dimensionality reduction. The technique consist of three steps. In a first step a linear SVM is trained on the given dataset. In the second step a freely chosen number of less significant features are removed from the dataset. One can identify less significant features by evaluating the direction vectors of the hyperplanes. A dimension where all direction vectors have a value that is close to zero corresponds to a dimension that has less impact on the classification and thus is less significant. After steps one and two, in step three one can repeat these two steps until the desired number of dimension has been reached or some other stop criterion is reached.

2.6 Integer Linear Programming

Integer linear programming (ILP) is used for linear optimization problems. The program consists of a linear objective function subject to linear constraints. The linear objective function can either be maximized or minimized and the linear constraints define the solution space. Furthermore, the program consists of variables, which are constrained to be non negative integers.A general definition of an ILP can be written as:

$$\max \ c^T x \tag{2}$$

$$\text{s.t. } Ax \leq b \tag{3}$$

$$x \in \mathbb{N}_0^n \tag{4}$$

Note that there are different definitions of an ILP but they can all be converted into one another. In this definition the linear objective function is defined in (2), the linear constraints can be found in (3) and the variables are constrained to be non-negative integers in (4). Matrix A and vector b in (3), as well as vector c in (2) are the inputs to the ILP. Solvers optimize the linear objective function while staying in the solution space, which is defined by the constraints. Optimization in this context means finding suitable values for the variables in vector x, while maximizing the value from (2).

2.7 Boolean Satisfiability Problem

The Boolean Satisfiability Problem (SAT) defines the problem of determining whether a Boolean formula can be satisfied. A Boolean formula is satisfiable if there is a way to set each variable to True or False, so that the formula evaluates to True. In Section 2.2 one could see two satisfiable boolean formulas, an example for an unsatisfiable boolean formula is given by:

$$\phi \wedge \neg\phi$$

The variable ϕ can not be set to either True or False so that the formula evaluates to True.
SAT was the first problem proven as NP-complete [6]. NP-complete means SAT has no polynomial time algorithm if NP $\neq$ P, which neither has been mathematically proven nor disproven. Moreover, all other problems in the complexity class NP are at most as difficult as solving SAT.

3 Methods

In the following sections we want to give an introduction to the workflow of LOCATOR and LOBICO, after that introduce the different methods used in LOCATOR and LOBICO and moreover give a detailed insight into the ILP and SAT formulations of the two methods.

3.1 Workflow

The workflow of the whole model can be divided into three steps, see Figure 2.

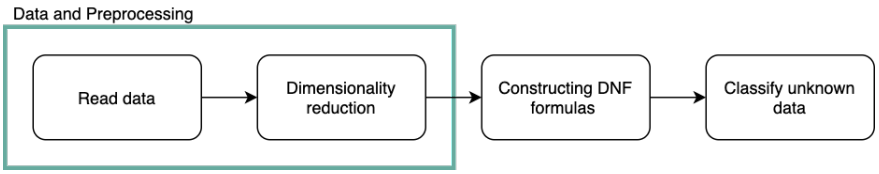


Figure 2: Model workflow.

The first step in the workflow is further divided into two smaller steps. First the program needs to read the configured data and then it starts with the dimensionality reduction as described in Section 2.5. After reducing the features of the data, a next step transforms the data into the correct format. Detailed information regarding the Data and Preprocessing steps are explained in [2].
In a next step the model constructs the DNF formulas. Calculation of these formulas for each class of the data can be done with two different methods. The first is LOCATOR, which tries finding an optimal minterm with up to L literals in each iteration. LOCATOR executes P iterations and after each excludes datapoints, to not construct the same minterm in each iteration. LOCATOR has several configuration options for excluding datapoints and features after each iteration, further information can be found in [2]. The second method is called LOBICO which in contrast to LOCATOR constructs the whole DNF formula including all P minterms with up to L literals per minterm in one iteration. For calculating the DNF formula both methods originally use an ILP formulation, which will be introduced in Section 3.2.1 and 3.3.1. In the scope of this thesis we implemented a SAT formulation for LOCATOR and LOBICO which will be introduced in Sections 3.2.2 and 3.3.2.
In the last step the DNF formulas from before are used to classify new unknown datapoints by plugging the new datapoint in the DNF formula for each class i. If one formula evaluates to True the model classifies the datapoint as the corresponding class. If more than one or none of the DNF formulas evaluate to True the model provides additional configuration options on how to classify the datapoint. All configurations for this case can be found in [2].

3.2 LOCATOR

As described in Section 3.1 LOCATOR constructs a DNF formula for each class c by constructing up to P optimal minterms with up to L literals each. For finding the optimal minterm in each

step LOCATOR uses an ILP or SAT formulation, which can be solved by an ILP or SAT solver. Both formulations take the data matrix X, the class label vector y and the parameter L as inputs. Furthermore, we define three binary variables l_m, l'_m and y'_n, which have the following properties:

$$l_m = \begin{cases} 1 & \text{if feature } X_m \text{ is used as a literal in the minterm,} \\ 0 & \text{else.} \end{cases} \tag{5}$$

$$l'_m = \begin{cases} 1 & \text{if feature } X_m \text{ is used as a negated literal in the minterm,} \\ 0 & \text{else.} \end{cases} \tag{6}$$

$$y'_n = \begin{cases} 1 & \text{if the minterm is true for data point } X_n, \\ 0 & \text{else.} \end{cases} \tag{7}$$

Note that we use the same variables for the SAT formulation in Section 3.2.2 but use interchangeably 1 as True and 0 as False.

3.2.1 ILP Formulation

Given the input and binary variables we define the ILP formulation of LOCATOR as follows.

$$\min \frac{N+1}{N} \cdot \sum_{\forall n: y_n = 0} y'_n - \sum_{\forall n: y_n = 1} y'_n \tag{8}$$

subject to:

$$l_m + l'_m \leq 1 \qquad m = 1, \ldots, M \tag{9}$$

$$\sum_{m=1}^{M} l_m + l'_m \leq L \tag{10}$$

$$\sum_{m=1}^{M} l_m \geq 1 \tag{11}$$

$$M \cdot y'_n \leq M - \sum_{\forall m: X_{nm} = 1} l'_m - \sum_{\forall m: X_{nm} = 0} l_m \leq y'_n + M - 1 \qquad n = 1, \ldots, N \tag{12}$$

In (8) we define the objective function of the ILP formulation. In this case the objective function aims at minimizing false positive and maximizing true positives. Furthermore false positives get slightly more penalized than true positives. A false positive classification is defined as a classification where $y_n = 0$ and $y'_n = 1$. A true positive classification is defined as a classification where $y_n = 1$ and $y'_n = 1$.

The inequality constraints in (9) define that l_m and l'_m can not be equal to one at the same time. This inequality speeds up processing time since if the minterm would include the feature as a positive and negative literal the minterm would always be false for every datapoint. The

constraint in inequality (10) limits the number of literals in the minterm with the help of the configured parameter L. In (11) we define that at least one feature has to have a positive literal, since cancer does not develop without any mutations. Next, the ILP formulation defines the connection between l_m, l'_m and y'_n. As defined in (7) y'_n should be equal to one, if the minterm is true for the datapoint X_n. This means the minterm can not have a conflict with the datapoint. We define a conflict within a minterm for datapoint X_n as the situation when the feature X_m is present in the datapoint X_n but the feature is used as a negative literal in the minterm, thus $l'_m = 1$, or when the feature X_m is absent in the datapoint X_n but the feature is used as a positive literal in the minterm, thus $l_m = 1$. For example, a conflict is present if datapoint n has a mutation in gene "A2M" and the gene is used as a negative literal in the minterm which means that the minterm is like $(\ldots \wedge \neg \text{A2M} \wedge \ldots)$. "A2M" would be set to True but as it is negated in the minterm the negated literal would be equal to False and thus the whole minterm equals False. If the minterm has a conflict with the datapoint X_n we enforce $y'_n = 0$ and otherwise $y'_n = 1$.

3.2.2 SAT Formulation

Given the existing ILP formulation for LOCATOR we want to rewrite the ILP as a SAT formulation. We will do that by translating each constraint of the ILP to a Boolean logic formula and at the end translate the objective function. The SAT formulation is solved using existing SAT solvers and therefore all Boolean formulas have to be in CNF, otherwise the solver can not use them.

We start by translating the constraints given in inequality (9), which allow only l_m or l'_m to be equal to one, not both at the same time. For the SAT formulation this means that only l_m or l'_m is allowed to be true, thus we add the following clauses to our SAT formulation.

$$(\neg l_m \vee \neg l'_m) \qquad\qquad m = 1, \ldots, M \tag{13}$$

Constraint (11) in the ILP formulation forces at least one literal in the minterm to be not negated. This translates into the following Boolean formula:

$$\bigvee_{m=1}^{M} l_m \tag{14}$$

Beginning with the next constraint (10), we observe the difficulty in translating ILP formulations to SAT formulations because counting with Boolean variables is complex and can produce a lot of additional variables or clauses. The inequality constrains the minterm, such that it consist of maximum L literals. In a first step we need to create a set Π, which includes all subsets of length $L+1$ of the set $\zeta = \{l_1, l_2, \ldots, l_M, l'_1, l'_2, \ldots, l'_M\}$, or more formally $\Pi = \{\pi \mid |\pi| = L+1 \ \wedge \ \pi \subseteq \zeta\}$. Next we add the following clauses to the SAT formulation, which are false when all variables of the clause are equal to True, otherwise the clauses evaluate to True, thus not more than L literals are allowed in the minterm.

$$\bigvee_{\pi_i \in \pi} \neg \pi_i \qquad \forall \pi \in \Pi \tag{15}$$

Constraint (12) enforces $y'_n = \text{False}$, if there is a conflict within the minterm for datapoint X_n, otherwise $y'_n = \text{True}$. This constraint requires a series of clauses to realize all dependencies. For each datapoint n we add the following clauses:

$$y'_n \vee (\bigvee_{\forall m: X_{n,m}=0} l_m) \vee (\bigvee_{\forall m: X_{n,m}=1} l'_m) \tag{16}$$

$$(\neg y'_n \vee \neg l_m) \qquad \forall m : X_{n,m} = 0 \tag{17}$$

$$(\neg y'_n \vee \neg l'_m) \qquad \forall m : X_{n,m} = 1 \tag{18}$$

The clause (16) enforces $y'_n = \text{True}$ if no literal produces a conflict within the minterm for datapoint X_n. Furthermore, clauses (17) and (18) enforce $y'_n = \text{False}$, if at least one literal has a conflict within the minterm for datapoint X_n.

In the next step we want to translate the objective function of the ILP formulation. As already mentioned, counting in SAT formulations needs a lot of additional clauses and variables. We start by reformulating the objective function of the ILP because in its base form (8), the function minimizes the number of false positives and maximizes the number of true positives. Furthermore, it penalizes false positives by factor $\frac{N+1}{N}$. Since including this penalizing factor in our SAT formulation is too difficult to realize and goes beyond the scope of this thesis, we will ignore it. Moreover, for the SAT formulation we want to use only one counting direction, meaning we either want to minimize or maximize, thus we change the objective function to the following:

$$\max \sum_{\forall n: y_n=0} (1 - y'_n) + \sum_{\forall n: y_n=1} y'_n \tag{19}$$

The changed objective function maximizes the number of true negatives while also maximizing the number of true positives. We want the ILP and SAT formulation to have the exact same objective values (disregarding the penalizing factor), thus we have to subtract the number of datapoints with $y_n = 0$ from the results of function (19). The objective values will be the same because of the following equation:

$$\#\text{true negatives} + \#\text{true positives} - |\{n : y_n = 0\}| = -\#\text{false positives} + \#\text{true positives} \tag{20}$$

For counting the number of true positives and true negatives we introduce new boolean variables f_n. We define the value of the variables $f_n, n = 1, \ldots, N$, by adding the following clauses to the SAT formulation:

$$(y'_n \vee f_n) \qquad \forall n : y_n = 0 \tag{21}$$

$$(\neg y'_n \vee \neg f_n) \qquad \forall n : y_n = 0 \tag{22}$$

$$(\neg y'_n \vee f_n) \qquad \forall n : y_n = 1 \tag{23}$$

$$(y'_n \vee \neg f_n) \qquad \forall n : y_n = 1 \tag{24}$$

The clauses added by equation (21) and (22) set for all datapoints which are not from the current class f_n = True, if the datapoint was classified as true negative.Furthermore clauses (23) and (24) set for all datapoints which belong to the current class f_n = True, if the datapoint was classified as true positive. The number of true variables f_n corresponds to our desired value from the objective function.

In a next step we need to implement a structure of clauses, which allows us to count the number of true f_n variables. The structure of clauses, which we are going to use, was provided by Dan Gusfield [7] and consists of four steps. Before starting with the steps we introduce a new set of boolean variables $t_{x,z}$ for $x = 1, \ldots, N+1$ and $z = 0, \ldots, x-1$. We interpret each variable $t_{x,z}$ as true if at least z variables in the set $\{f_n : \ n < x\}$ are set True, or more formally:

$$t_{x,z} = \textit{True, if } |\{f_n \mid f_n = \textit{True}, \ n < x\}| \geq z$$

As the following clauses do not give a good introduction into how the structure works one can see in Figure 3 two examples for counting the true function variables $F_0, F_1, F_2, \ldots, F_5$.

As one can see in the examples the variables $T_{x,0}$ are always true, this can be achieved by adding the following clauses:

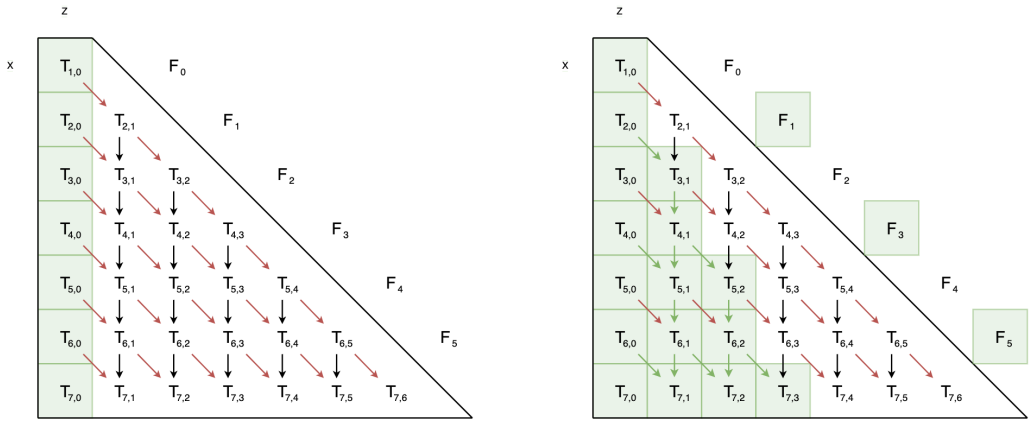
$$(t_{x,0}) \qquad x = 1, \ldots, N+1 \tag{25}$$

As the clauses only consist of a single literal, the corresponding variable is always equal to True. According to the interpretation of $t_{x,z}$ this means that at least zero f_n are true. In the second step we introduce clauses $t_{x,z} \implies t_{x+1,z}$, which means taking one more variable f_n into the set $\{f_n : \ n < x\}$ can not decrease the total number of true variables f_n. The clauses correspond to the black arrows in Figure 3 and read as follows:

$$(\neg t_{x,z} \vee t_{x+1,z}) \qquad x = 1, \ldots, N+1; \ z = 1, \ldots, x-1 \tag{26}$$

Note that z starts at index one and not at zero since we already set all $t_{x,0}$ = True in clauses (25), thus $t_{x,0} \implies t_{x+1,0}$ is not needed.

Starting with step three we add the clauses, which include the function variables f_n. We add the following clauses to our SAT formulation which correspond to the red arrows in Figure 3:



(a) objective value 0 (b) objective value 3

Figure 3: Counting Structure with examples: In the figure the background of variables $T_{x,z}$ and F_{x-1} is set to green, if their value is equal to True. The arrows connecting the variables $T_{x,z}$ each represent one clause added to the SAT formulation. The red arrows set its destination to True, if the outgoing $T_{x,z}$ variable is equal to True and the right hand function variable F_{x-1} is equal to True. The black arrows set its destination to True, if the outgoing $T_{x,z}$ variable is true. Example (a) shows what happens, if none of the function variables are equal to True. We observe that in this example only the $T_{x,z}$ variables in the most left column are equal to True. Looking at variable $T_{7,1}$ we know that zero of the F_{x-1} variables are equal to true since it is equal to False. In example (b) one can see that function variables F_1, F_3 and F_5 are equal to True. This situation leads to more true $T_{x,z}$ variables because we now have green arrows which set their destination variables to True. Looking at the last row we observe that this time $T_{7,0}$ until $T_{7,3}$ have a green background and thus are equal to True. Because of the fact that $T_{7,3}$ is true and $T_{7,4}$ is false we know that exactly three function variables are equal to True. In both examples the figure does not show all clauses/arrows added by the structure because it would make the figure too messy. The missing arrows restrict variables $T_{x,z}$ from being equal to True even if none of the incoming arrows are green.

$$(t_{x+1,z+1} \vee \neg t_{x,z} \vee \neg f_x) \qquad x = 1,\ldots,N;\ z := 0,\ldots,x-1 \tag{27}$$

These clauses represent the case that z variables of $f_n,\ n < x$ are equal to True and the variable f_x = True, which then implicates that $t_{x+1,z+1}$ = True. In other words adding one true variable to the set of true variables implicates that the set has become larger by one and the number of true variables has also increased by one.

The three steps created a structure of clauses, such that for at least z variables being equal to true the variable $t_{N+1,z}$ will be true. Looking at the clauses added in the last three steps we have implemented all red and black arrows from Figure 3 but as already stated in the description of the figure we need to restrict variables $t_{x,z}$ from being equal to True even if none of the arrows set the variable True. We need to do that because the arrows are all implications and as shown in Table 2, the right side of an implication can be true even if the left side is false. In Figure 3 we observe that all $T_{x,z}$ variables except the outer diagonal and the most left column have two incoming arrows. Consequently we have to add clauses that only allow the inner $t_{x,z}$ variables to be equal to True if they have been set to True by the red and/or black arrow. For the $t_{x,z}$ variables in the outer diagonal we only have add this dependency for the red arrow

and for variables in the most left column we can ignore the dependency completely as they are true because of (25). The following implications achieve this dependency.

$$t_{x,z} \implies t_{x-1,z} \vee [t_{x-1,z-1} \wedge f_{x-1}] \qquad x = 2,\dots,N+1;\ z = 1,\dots,x-2 \tag{28}$$

$$t_{x,z} \implies [t_{x-1,z-1} \wedge f_{x-1}] \qquad x = 2,\dots,N+1;\ z = x-1 \tag{29}$$

The implication translates into the following boolean formula:

$$(t_{x-1,z} \vee [t_{x-1,z-1} \wedge f_{x-1}] \vee \neg t_{x,z}) \qquad x = 2,\dots,N+1;\ z = 1,\dots,x-2 \tag{30}$$

$$[t_{x-1,z-1} \wedge f_{x-1}] \vee \neg t_{x,z} \qquad x = 2,\dots,N+1;\ z = x-1 \tag{31}$$

Unfortunately, the Boolean formulas above are in DNF and not in CNF. Since the SAT solver only accepts formulas in CNF, (30) and (31) will be transformed to CNF with the Tseytin encoding explained in [7]. The idea is to first set variables $q_{x-1,z-1}$ equivalent to the clauses $[t_{x-1,z-1} \wedge f_{x-1}]$ while only using CNF formulas. The equivalence is achieved by adding the following clauses:

$$(t_{x-1,z-1} \vee \neg q_{x-1,z-1}) \qquad x = 1,\dots,N+1;\ z = 1,\dots,x-1 \tag{32}$$

$$(f_{x-1} \vee \neg q_{x-1,z-1}) \qquad x = 1,\dots,N+1;\ z = 1,\dots,x-1 \tag{33}$$

$$(\neg t_{x-1,z-1} \vee \neg f_{x-1} \vee q_{x-1,z-1}) \qquad x = 1,\dots,N+1;\ z = 1,\dots,x-1 \tag{34}$$

As $q_{x-1,z-1}$ is equivalent to $[t_{x-1,z-1} \wedge f_{x-1}]$ we can replace it in formula (30) and (31).

$$(t_{x-1,z} \vee q_{x-1,z-1} \vee \neg t_{x,z}) \qquad x = 1,\dots,N+1;\ z = 1,\dots,x-2 \tag{35}$$

$$(q_{x-1,z-1} \vee \neg t_{x,z}) \qquad x = 2,\dots,N+1;\ z = x-1 \tag{36}$$

The variables $t_{N+1,z}$ are now only equal to True if at least z variables f_n are equal to True. When solving the SAT formulation we can assume $t_{N+1,z}$ to be equal to True and if the formulation is satisfiable then we can increase z by one. If at some point the solver proves that the formulation is not satisfiable then $z-1$ is our maximum objective value.

3.2.3 SAT Modifications

The SAT formulation given in Section 3.2.2 can be further improved by applying the following heuristic and three modifications. We define an improvement to the SAT formulation as a modification where the number of variables or clauses added to the SAT formulation decreases.
Starting with the heuristic, the idea is that we decrease the number of iterations the SAT solver has to do in order to find the maximum objective value. At the moment we would start the iterations at $t_{N+1,1}$ by assuming that $t_{N+1,1} =$ True. If an instance with $t_{N+1,z} =$ True is solvable

then our objective value (without the correction from (20)) is at least equal to z. As we also count true negative classifications in our objective value an empty minterm has an objective value of $g = |\{n : y_n = 0\}|$, as all datapoints from other classes are predicted correctly. We now want the minterm to be at least as good as an empty minterm and thus start the iterations at $t_{N+1,g}$. This heuristic was used in all modifications of the SAT formulation.
The first modification reduces the number of variables by N because all variables f_n are removed from the formulation. Furthermore all clauses in equations (21) to (24) are removed. Instead of using the variables f_n in the counting clauses we directly use the variables y'_n and therefore change (27) to the following clauses:

$$\forall n : y_n = 1 :$$
$$(t_{x+1,z+1} \vee \neg t_{x,z} \vee \neg y'_n) \qquad x = n;\ z := 0, \ldots, x-1 \tag{37}$$
$$\forall n : y_n = 0 :$$
$$(t_{x+1,z+1} \vee \neg t_{x,z} \vee y'_n) \qquad x = n;\ z := 0, \ldots, x-1 \tag{38}$$

In the new clauses we have to decide between samples with $y_n = 1$ and samples with $y_n = 0$ because for samples from the current class we want to count variables $y'_n = \text{True}$ and for samples from other classes we want to count variables where $y'_n = \text{False}$. Furthermore we have to change the clauses (32) to (34) and apply the distinction as well:

$$\forall n : y_n = 1 :$$
$$(t_{x-1,z-1} \vee \neg q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{39}$$
$$(y'_{x-1} \vee \neg q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{40}$$
$$(\neg t_{x-1,z-1} \vee \neg y'_{x-1} \vee q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{41}$$

$$\forall n : y_n = 0 :$$
$$(t_{x-1,z-1} \vee \neg q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{42}$$
$$(\neg y'_{x-1} \vee \neg q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{43}$$
$$(\neg t_{x-1,z-1} \vee y'_{x-1} \vee q_{x-1,z-1}) \qquad x = n;\ z = 1, \ldots, x-1 \tag{44}$$

The second modification to the SAT formulation reduces clauses added by equation (15). In the equation we add clauses such that the number of literals per minterm is less than or equal to the input variable L. The equation (15) adds a clause for each combination of positive and negative literals of length $L+1$, but as we already defined that l_m and l'_m can not be both equal to true $\forall m = 1, \ldots, M$, some combinations where the negative and positive literal of a feature is present can be ignored. Only generating the need combinations of literals reduces the number of clauses added from (15) by approximately 7.36 %. Detailed information on how to exclude the combinations can be found in the source code.

The last modification removes all clauses added by equation (15). Instead of adding a clause for each subset of positive and negative literals of length $L+1$, the third modification counts the number of false variables in the set of all literals $\zeta = (l_1, l'_1, \ldots, l_M, l'_M)$. We use the same technique as for counting the function variables f_n and add the variables $r_{x,z}$ and $q'_{x,z}$ to the SAT formulation. Furthermore, we add the following clauses to the formulation for counting the literals:

$$(r_{x,0}) \qquad x = 1, \ldots, 2M+1 \tag{45}$$

$$(\neg r_{x,z} \vee r_{x+1,z}) \qquad x = 1, \ldots, 2M;\ z = 1, \ldots, x-1 \tag{46}$$

$$(r_{x+1,z+1} \vee \neg r_{x,z} \vee \zeta_x) \qquad x = 1, \ldots, 2M;\ z = 1, \ldots, x-1 \tag{47}$$

$$(r_{x-1,z-1} \vee \neg q'_{x-1,z-1}) \qquad x = 2, \ldots, 2M+1;\ z = 1, \ldots, x-1 \tag{48}$$

$$(\neg \zeta_{x-1} \vee \neg q'_{x-1,z-1}) \qquad x = 2, \ldots, 2M+1;\ z = 1, \ldots, x-1 \tag{49}$$

$$(\neg r_{x-1,z-1} \vee \zeta_{x-1} \vee q'_{x-1,z-1}) \qquad x = 2, \ldots, 2M+1;\ z = 1, \ldots, x-1 \tag{50}$$

$$(r_{x-1,z} \vee q'_{x-1,z-1} \vee \neg r_{x,z}) \qquad x = 2, \ldots, 2M+1;\ z = 1, \ldots, x-2 \tag{51}$$

$$(q'_{x-1,z-1} \vee \neg r_{x,z}) \qquad x = 2, \ldots, 2M+1;\ z = x-1 \tag{52}$$

When solving the SAT formulation we can configure the solver to set $r_{(2M+1),(2M-L)}$ to True and thus restrict the number of literals used in the final minterm.

The three modifications lead to five different configurations of the LOCATOR SAT formulation:

1. "LOCATOR SAT no mod" no modification.
2. "LOCATOR SAT modA" only the first modification.
3. "LOCATOR SAT modB" only the second modification.
4. "LOCATOR SAT modAB" the first and second modification.
5. "LOCATOR SAT modAC" the first and third modification.

3.3 LOBICO

The modeling framework LOBICO was introduced by Knijnenburg et al. [3] and originally used to explain drug response of cancer cell lines using gene mutation data. Since LOBICO is a modeling framework we use it in this thesis for the same purpose as we use LOCATOR and classify cancer types based on binary mutation data. In the original version LOBICO uses an ILP formulation for calculating the DNF formulas. Contrary to LOCATOR, LOBICO directly calculates the whole DNF formula including all P minterms in one step. In the following sections we want to give an introduction into the ILP formulation and the SAT formulation. Both the ILP and SAT formulation get the data matrix X, the vector storing the class labels y, L, and P as inputs and use the following variables:

$$s_{mp} = \begin{cases} 1 & \text{if feature } X_m \text{ is used as a positive literal in the } p\text{th minterm,} \\ 0 & \text{else.} \end{cases} \tag{53}$$

$$s'_{mp} = \begin{cases} 1 & \text{if feature } X_m \text{ is used as a negative literal in the } p\text{th minterm,} \\ 0 & \text{else.} \end{cases} \tag{54}$$

$$y'_n = \begin{cases} 1 & \text{if the DNF formula is true for data point } X_n, \\ 0 & \text{else.} \end{cases} \tag{55}$$

$$t_{np} = \begin{cases} 1 & \text{if the } p\text{th minterm is true for data point } X_n, \\ 0 & \text{else.} \end{cases} \tag{56}$$

$$e_p = \begin{cases} 1 & \text{if the the } p\text{th minterm is not empty,} \\ 0 & \text{else.} \end{cases} \tag{57}$$

Note that we use Boolean values for the variables in the SAT formulation.

3.3.1 ILP Formulation

The following ILP formulation was changed from the original ILP formulation introduced in [3]. The first difference is that the original ILP formulation uses variables w_n for each datapoint n as misclassification weight. As theses variables are not binary we omit them due to the fact that it would be difficult to realize such weights with a SAT formulation and this is out of scope for this thesis. Furthermore, constraint (7) in [3] was changed to the constraints (61), (62) and (63) because in the original version of the ILP the constraint (7) defines an empty minterm as True and forces the generated DNF formula to have exactly P minterms. However, in our version we want to define an empty minterm as False and the generated DNF formulas should consist of maximum P minterms. The changed LOBICO ILP formulation reads as follows:

$$\min \sum_{\forall n: y_n = 0} y'_n - \sum_{\forall n: y_n = 1} y'_n \tag{58}$$

subject to:

$$s_{mp} + s'_{mp} \leq 1 \qquad m = 1, \ldots, M \; ; \; p = 1, \ldots, P \tag{59}$$

$$\sum_{p=1}^{P} (s_{mp} + s'_{mp}) \leq L \qquad m = 1, \ldots, M \tag{60}$$

$$e_p \leq \sum_{m=1}^{M} (s_{mp} + s'_{mp}) \leq M \cdot e_p \qquad p = 1, \ldots, P \tag{61}$$

$$M \cdot t_{np} \leq M - \sum_{\forall m: X_{nm} = 0} s_{mp} - \sum_{\forall m: X_{nm} = 1} s'_{mp} \qquad p = 1, \ldots, P \; ; \; n = 1, \ldots, N \tag{62}$$

$$M \geq M - e_p + t_{np} \geq M - \sum_{\forall m: X_{nm}=0} s_{mp} - \sum_{\forall m: X_{nm}=1} s'_{mp} \quad p = 1,\ldots,P\ ;\ n = 1,\ldots,N \quad (63)$$

$$y'_n \leq \sum_{p=1}^{P} t_{np} \leq P \cdot y'_n \qquad n = 1,\ldots,N \quad (64)$$

The objective function (58) of the LOBICO ILP formulation aims at minimizing false positive classifications while also maximizing true positive classifications. (59) restricts all minterms from 1 to P, such that they do not include a feature as positive and negative literal because that would cause the minterm to be false for all datapoints and it speeds up computation. Next, constraints (60) limit the number of literals per minterm to the input variable L. Furthermore, inequalities (61) set the variable e_p based on the number of literals in the minterm p. If the minterm is empty, meaning that $\sum_{m=1}^{M}(s_{mp} + s'_{mp}) = 0$, then $e_p = 0$ and else $e_p = 1$. Variables e_p are necessary since we define an empty minterm as false for all datapoints. In constraints (62) the first dependency between the literals in the minterms and the variables t_{np} is defined. As t_{np} should be equal to zero if the minterm p is false for datapoint n, the constraint forces $t_{np} = 0$ if there is at least one conflict within the minterm p for datapoint n. Constraints (63) define the other direction of this dependency, meaning that $t_{np} = 1$, if the minterm p is true for datapoint n. Furthermore the inequalities (63) force $t_{np} = 0$ with the help of the variables e_p, even if there is no conflict within the minterm because we defined an empty minterm as false for all datapoints. Lastly the constraints (64) define the connection between y'_n and the variables t_{np}, such that $y'_n = 1$, if at least one $t_{np} = 1$, else $y'_n = 0$.

3.3.2 SAT Formulation

In this section we want to introduce the SAT formulation for LOBICO. We will start by translating each constraint of the ILP formulation step by step and at the end translate the objective function.

We start with constraints (59), which define that only the positive literal or the negative literal of a feature can be present at the same time in the same minterm, thus s_{mp} and s'_{mp} can not be both true. We achieve this by adding the following clauses to the SAT formulation:

$$(\neg s_{mp} \vee \neg s'_{mp}) \qquad m = 1,\ldots,M;\ p = 1,\ldots,P \quad (65)$$

Furthermore with the next clauses we want to limit the number of literals per minterm to be less or equal to L, thus we apply the following method $\forall p = 1,\ldots,P$:

1. Calculate the set Π, which includes all subsets of length $L+1$ of elements: $\{s_{1,p}, s'_{1,p}, s_{2,p}, s'_{2,p}, \ldots, s_{M,p}, s'_{M,p}\}$.
2. Add the following clauses to the SAT formulation:

$$\bigvee_{\pi_i \in \pi} \neg \pi_i \qquad \forall \pi \in \Pi \tag{66}$$

Next, we want to add the dependency between e_p and the number of literals in the minterm p to the SAT formulation. As defined in (57) e_p should be equal to False if the minterm is empty and if not it should be equal to True. We can achieve this dependency by adding the following clauses to the SAT formulation:

$$\neg e_p \vee (\bigvee_{m=1}^{M} (s_{mp} \vee s'_{mp})) \qquad \forall p = 1, \ldots, P \tag{67}$$

$$(e_p \vee s_{mp}) \qquad p = 1, \ldots, P;\ m = 1, \ldots, M \tag{68}$$

$$(e_p \vee s'_{mp}) \qquad p = 1, \ldots, P;\ m = 1, \ldots, M \tag{69}$$

The next clauses refer to constraints (62) and (63) of the LOBICO ILP formulation and define that t_{np} is equal to False, when there is a conflict between the literals of the minterm p and the features of the datapoint n or if the minterm p is empty (e_p = False). Furthermore, the clauses define that t_{np} = True if there is no conflict and the minterm is not empty. The following clauses are added to the formulation:

$$\forall m : X_{nm} = 0 :$$

$$(\neg t_{np} \vee \neg s_{mp}) \qquad p = 1, \ldots, P;\ n = 1, \ldots, N \tag{70}$$

$$\forall m : X_{nm} = 1 :$$

$$(\neg t_{np} \vee \neg s'_{mp}) \qquad p = 1, \ldots, P;\ n = 1, \ldots, N \tag{71}$$

$$(e_p \vee \neg t_{np}) \qquad p = 1, \ldots, P;\ n = 1, \ldots, N \tag{72}$$

$$t_{np} \vee (\bigvee_{\forall m : X_{nm} = 0} s_{mp}) \vee (\bigvee_{\forall m : X_{nm} = 1} s'_{mp}) \vee \neg e_p \qquad p = 1, \ldots, P;\ n = 1, \ldots, N \tag{73}$$

In the last step of the constraints we connect the values of the variables t_{np} with the variables y'_n, such that y'_n = True, if at least one t_{np} = True and else y'_n = False, thus we add the following clauses to the SAT formulation:

$$\neg y'_n \vee (\bigvee_{p=1}^{P} t_{np}) \qquad n = 1, \ldots, N \tag{74}$$

$$(y'_n \vee \neg t_{np}) \qquad p = 1, \ldots, P;\ n = 1, \ldots, N \tag{75}$$

The SAT formulation now includes all constraints of the ILP formulation. In the last step of the SAT formulation the objective function is implemented with the variables y'_n, which have been set by clauses (74) and (75). The objective function introduced in section 3.3.1 is

the same objective function as in the ILP formulation of LOCATOR but the penalization factor $\frac{N+1}{N}$ is not included. Since we did not implement this factor it was not included in the SAT formulation of LOCATOR and thus we use the exact same objective function for the LOBICO SAT formulation as for the LOCATOR SAT formulation. In the objective function we want to maximize true negative classifications while also maximizing true positive classifications. The function reads as follows:

$$\max \sum_{\forall n: y_n = 0} (1 - y'_n) + \sum_{\forall n: y_n = 1} y'_n \tag{76}$$

Having a different objective function than the ILP formulation enforces another calculation step as the objective values should be equivalent in the end. After the solver has maximized the objective function in the SAT formulation we subtract the number of datapoints with $y_n = 0$ from the objective value. Because of equation (20) we get the same objective value as the in the ILP formulation.

As the LOCATOR SAT and LOBICO SAT objective function are identical we do not get into detail on the following clauses again as they are the same as in Section 3.2.2. We state them here again for completeness of the formulation. The implementation requires adding the variables $f_n : n = 1,\dots,N$, $k_{x,z} : x = 1,\dots,N+1;\ z = 0,\dots,x-1$ and $q_{x,z} : x = 1,\dots,N+1;\ z = 0,\dots,x-1$ to the model and the following clauses:

$$(y'_n \vee f_n) \qquad \forall n : y_n = 0 \tag{77}$$
$$(\neg y'_n \vee \neg f_n) \qquad \forall n : y_n = 0 \tag{78}$$
$$(\neg y'_n \vee f_n) \qquad \forall n : y_n = 1 \tag{79}$$
$$(y'_n \vee \neg f_n) \qquad \forall n : y_n = 1 \tag{80}$$

$$(k_{x,z} \vee \neg q_{x,z}) \qquad x = 1,\dots,N+1;\ z = 0,\dots,x-1 \tag{81}$$
$$(f_x \vee \neg q_{x,z}) \qquad x = 1,\dots,N+1;\ z = 0,\dots,x-1 \tag{82}$$
$$(\neg k_{x,z} \vee \neg f_x \vee q_{x,z}) \qquad x = 1,\dots,N+1;\ z = 0,\dots,x-1 \tag{83}$$

$$k_{x,0} \qquad x = 1,\dots,N+1 \tag{84}$$
$$(\neg k_{x,z} \vee k_{x+1,z}) \qquad x = 1,\dots,N;\ z = 0,\dots,x-1 \tag{85}$$
$$(k_{x+1,z+1} \vee \neg k_{x,z} \vee \neg f_x) \qquad x = 1,\dots,N;\ z = 0,\dots,x-2 \tag{86}$$
$$(q_{x-1,z-1} \vee \neg k_{x,z}) \qquad x = 2,\dots,N+1;\ z = 1,\dots,x-2 \tag{87}$$
$$(k_{x-1,x-1} \vee q_{x-1,x-2} \vee \neg k_{x,x-1}) \qquad x = 2,\dots,N+1 \tag{88}$$

In an iterative approach the variable $k_{N+1,z}$ is assumed to be equal to true and if the SAT formulation is still solvable by the solver than z is increased by one. If at some point the solver

proves the formulation as unsatisfiable, then $z-1$ is the maximum objective value. In the end the value is transformed according to (20) to match with the objective value of the ILP formulation.

3.3.3 SAT Modifications

Section 3.2.3 introduced three modifications and one heuristic that optimize the SAT formulation of LOCATOR. We can apply these improvements to LOBICO as well to reduce the number of iterations, variables and clauses of the SAT formulation.
The heuristic described in 3.2.3 was applied without any changes to all LOBICO SAT modifications.
The first modification removes variables f_n and clauses (77) to (80) from the formulation. Furthermore we use variables y'_n instead of f_n and change clauses (81) to (83) to the following:

$$(k_{x,z} \vee \neg q_{x,z}) \qquad x = 1,\dots,N+1;\ z = 0,\dots,x-1 \tag{89}$$
$$(y'_x \vee \neg q_{x,z}) \qquad \forall x : y_x = 1;\ z = 0,\dots,x-1 \tag{90}$$
$$(\neg y'_x \vee \neg q_{x,z}) \qquad \forall x : y_x = 0;\ z = 0,\dots,x-1 \tag{91}$$
$$(\neg k_{x,z} \vee \neg y'_x \vee q_{x,z}) \qquad \forall x : y_x = 1;\ z = 0,\dots,x-1 \tag{92}$$
$$(\neg k_{x,z} \vee y'_x \vee q_{x,z}) \qquad \forall x : y_x = 0;\ z = 0,\dots,x-1 \tag{93}$$

Additionally we change equation (86) to the following two equations:

$$(k_{x+1,z+1} \vee \neg k_{x,z} \vee \neg y'_x) \qquad \forall x : y_x = 1;\ z = 0,\dots,x-2 \tag{94}$$
$$(k_{x+1,z+1} \vee \neg k_{x,z} \vee y'_x) \qquad \forall x : y_x = 0;\ z = 0,\dots,x-2 \tag{95}$$

The second modification removes not needed clauses added by equation (66). The base equation stays the same but the set Π has to be changed. This can be done as explained in Section 3.2.3 but for LOBICO we need to do the step for each minterm.

The third modification removes all clauses added by equation (66). Instead of adding all combinations of literals of length $L+1$, this modification counts the number of false variables for all minterms p in the set $\zeta^p = (s_{1,p}, s'_{1,p}, \dots, s_{M,p}, s'_{M,p})$. A detailed explanation for one minterm p can be found in Section 3.2.3. We add new variables $r_{p,x,z}$ and $Q_{p,x,z}$ to the SAT formulation and add the following clauses $\forall p = 1,\dots,P$:

$$(r_{p,x,0}) \qquad x = 1,\dots,2M+1 \tag{96}$$
$$(\neg r_{p,x,z} \vee r_{p,x+1,z}) \qquad x = 1,\dots,2M;\ z = 1,\dots,x-1 \tag{97}$$
$$(r_{p,x+1,z+1} \vee \neg r_{p,x,z} \vee \zeta^p_x) \qquad x = 1,\dots,2M;\ z = 1,\dots,x-1 \tag{98}$$

$$(r_{p,x-1,z-1} \vee \neg Q_{p,x-1,z-1}) \qquad x = 2,\ldots,2M+1;\ z = 1,\ldots,x-1 \tag{99}$$

$$(\neg \zeta^{p}_{x-1} \vee \neg Q_{p,x-1,z-1}) \qquad x = 2,\ldots,2M+1;\ z = 1,\ldots,x-1 \tag{100}$$

$$(\neg r_{p,x-1,z-1} \vee \zeta^{p}_{x-1} \vee Q_{p,x-1,z-1}) \qquad x = 2,\ldots,2M+1;\ z = 1,\ldots,x-1 \tag{101}$$

$$(r_{p,x-1,z} \vee Q_{p,x-1,z-1} \vee \neg r_{p,x,z}) \qquad x = 2,\ldots,2M+1;\ z = 1,\ldots,x-2 \tag{102}$$

$$(Q_{p,x-1,z-1} \vee \neg r_{p,x,z}) \qquad x = 2,\ldots,2M+1;\ z = x-1 \tag{103}$$

Setting $\forall p = 1,\ldots,P : r_{p,(2M+1),(2M-L)} =$ True restricts the final minterms to include less or equal to L literals.

As for LOCATOR we define five different configurations of the LOBICO SAT formulation based on the three modifications above:

1. "LOBICO SAT no mod" no modification.
2. "LOBICO SAT modA" only the first modification.
3. "LOBICO SAT modB" only the second modification.
4. "LOBICO SAT modAB" the first and second modification.
5. "LOBICO SAT modAC" the first and third modification.

4 Results

The evaluation of the methods LOCATOR and LOBICO regarding their classification performance was done by Tran [2] for LOCATOR and Knijnenburg et al. [3] for LOBICO. The LOBICO SAT formulation introduced in this thesis yields the same results as the LOBICO ILP formulation (disregarding the weights in the objective function from [3]), thus the classification performance of the SAT formulation is the same as for the ILP formulation. The LOCATOR SAT and LOCATOR ILP results can be different due to the missing penalizing factor in the objective function of the SAT formulation, but the difference in classification performance is small as expected and thus will not be further discussed. The evaluation focuses on comparing the time consumption of the SAT and ILP formulations. The SAT and ILP formulations were always executed inside the same environment such that the experiments produce comparable results. All experiments were made in the High Performance Computing (HPC) environment of the Heinrich Heine University [8]. Each computation for one class was done with one CPU and 8 GB of RAM. Both formulations were implemented with Python 3 and Snakemake [9], which is a workflow management system for creating scalable and reproducible data analyses. Furthermore, the ILP formulations were solved using the Python interface of Gurobi [10] and the SAT formulations were solved using three different solvers: the Python interface of cryptominisat called pycryptosat [11], Glucose 4.1 [12] via the PySAT interface [13] and Lingeling [14] also via the PySAT interface.
The datasets used in all experiments are subsets of the dataset described in Section 2.3. The number of features per dataset was reduced with the SVM-RFE from Section 2.5 until the accuracy of the SVM model decreased, and in some cases the features were further decreased with a fixed number using the SVM-RFE. The samples per dataset were randomly selected such that the percentage of samples per class is the same as in the whole dataset.

4.1 LOCATOR

In the following section we give an introduction into the runtime differences between the different SAT formulations and furthermore compare them to the runtime of the ILP formulation. The first dataset for our experiment includes 255 datapoints and 40 features. The experiment gives an overview over the runtimes of the three different solvers across different SAT configurations while $L = 3$ and $P = 3$. Figure 4 shows the results of the experiment.

We can see that cryptominisat [11] is the fastest solver across all instances, followed by Glucose 4.1 [12]. Lingeling [14] has by far the worst performance with more than double the runtime compared to cryptominisat. Additionally, one can see in Figure 4 that "no mod", "modA", "modB", and "modAB" have approximately the same runtime, whereas "LOCATOR SAT modAC" has a much lower runtime. Because of these results, in the following experiments with larger datasets we only use the configuration "LOCATOR SAT modAC" together with the cryptominisat solver.
Next, we tested the "LOCATOR ILP" and "LOCATOR SAT modAC" formulations for datasets with

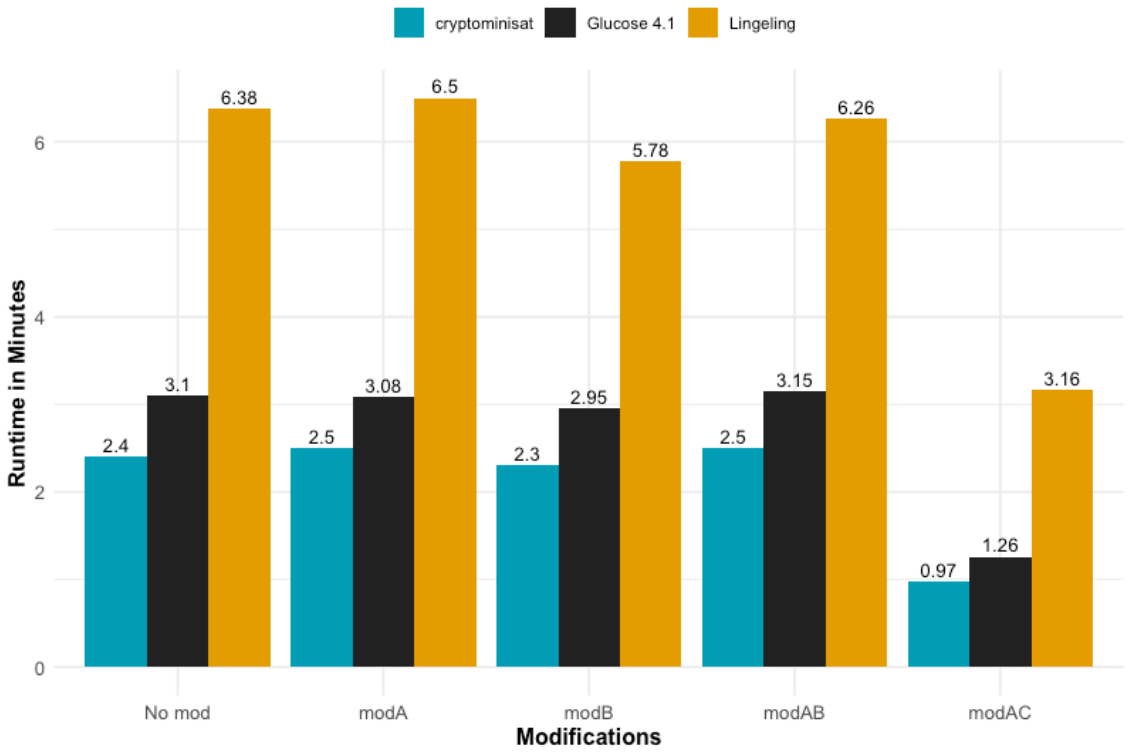
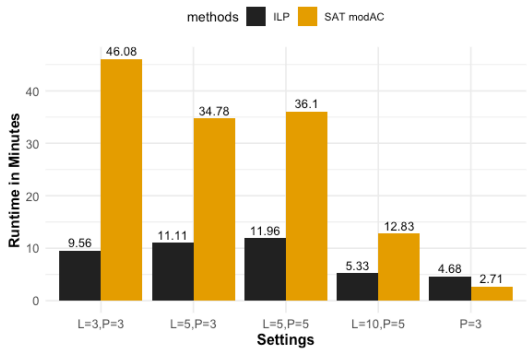


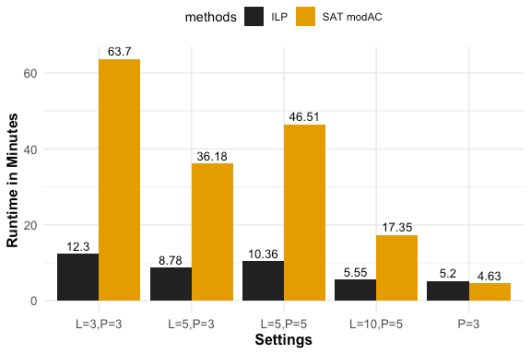
Figure 4: Comparison of all LOCATOR SAT modifications tested with all solvers. $L = 3$, $P = 3$, $N = 255$, and $M = 40$

$N = 415$ ($\frac{6640}{16}$), $N = 475$ ($\frac{6640}{14}$), $N = 554$ ($\frac{6640}{12}$), and $N = 664$ ($\frac{6640}{10}$) while always using the same 250 features. Both methods needed at most 3 GB of RAM. One can see the results in Figure 5.

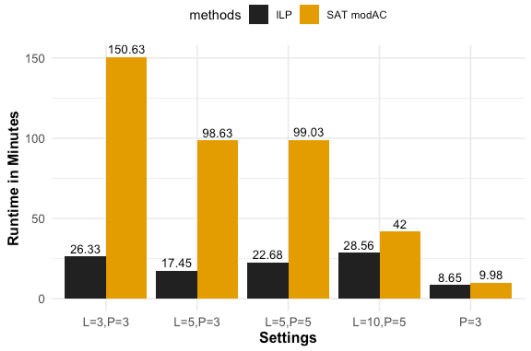
The bar chart shows that the runtime of the "LOCATOR ILP" formulations has a tendency to decrease when increasing L. Furthermore, if we do not specify L the runtime is the lowest in all cases. For the "LOCATOR SAT modAC" formulation we can make the same observation that the runtime decreases when increasing L and that an unlimited L has the lowest runtime. The reduction of the runtime when L is set to unlimited is very large for the "LOCATOR SAT modAC" formulation. Comparing the ILP and SAT formulations runtimes, we can see that the ILP formulation is faster in almost all cases. However, when we configure L as unlimited the SAT formulation can be faster as one can see in (a) and (b). However, the runtime advantage of the SAT formulation decreases when increasing the number of samples in the dataset and thus in (c) the two methods are almost equal and in (d) the ILP formulation is faster again. Often the problem with SAT formulations is that it takes a lot of time to prove an instance to be unsatisfiable (UNSAT) [7] thus Table 3 gives an insight into the exact runtimes of the "LOCATOR SAT modAC" formulation. The following results are based on the same experiments as Figure 5. For all classes we recorded how long the solver needed to prove an instance as unsatisfiable. When we configure $P = 3$ we get exactly three runtimes for each class as the solver has to prove UNSAT exactly three times per class. Summing all these runtimes up and dividing by the total runtime of that configuration shown in Figure 5 we get the runtime



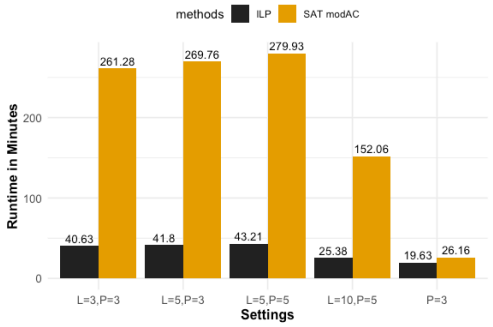
(a) $N = 415$, $M = 250$



(b) $N = 475$, $M = 250$



(c) $N = 554$, $M = 250$



(d) $N = 664$, $M = 250$

Figure 5: Comparisons between "LOCATOR ILP" and "LOCATOR SAT modAC" for different sizes of datasets and different configurations for L and P. If L is not specified then the number of features per minterm is not limited.

percentage of UNSAT iterations. Additionally, we counted the number of iterations such that we can also include how many iterations out of all iterations were UNSAT iterations. The left values in Table 3 show the runtime percentage of UNSAT iterations and the right values the number of UNSAT iterations divided by the total number of iterations.

N	$L = 3, P = 3$	$L = 5, P = 3$	$L = 5, P = 5$	$L = 10, P = 5$	$P = 3$
415	17% | 18%	23% | 16%	23% | 20%	12% | 19%	11% | 15%
475	16% | 17%	9% | 15%	10% | 18%	3% | 17%	6% | 14%
554	35% | 16%	7% | 13%	8% | 17%	11% | 16%	13% | 13%
664	28% | 14%	9% | 12%	10% | 15%	5% | 14%	5% | 11%

Table 3: Each row shows the results for a different dataset with N samples. The columns represent the results for one configuration of the formulation. Additionally, the left values of an entry show the summed UNSAT iteration runtimes divided by the total runtime and the right values of an entry show the number of UNSAT iterations divided by the total number of iterations. Note that the values are all rounded.

In Table 3 one can see that in most cases the UNSAT iterations runtimes are below or equivalent to the average expected runtime per iteration, since the left percentage is lower than the right percentage.

4.2 LOBICO

In this section we want to present the results achieved with the LOBICO SAT and LOBICO ILP formulations. LOCATOR has been tested with at least $N = 415$ samples and $M = 250$ features per dataset (except Figure 4). However, the LOBICO instances are more complex since they are calculating the optimal DNF formula instead of single minterms such that the number of samples and features per dataset had to be further decreased to achieve results in reasonable time.

As for LOCATOR, we first compare different solvers. We tested the solvers cryptominisat and Glucose 4.1 on a dataset with $N = 255$ samples, $M = 40$ features, $L = 3$, and $P = 3$. Solver Lingeling was not included in the results as the solver was too slow.

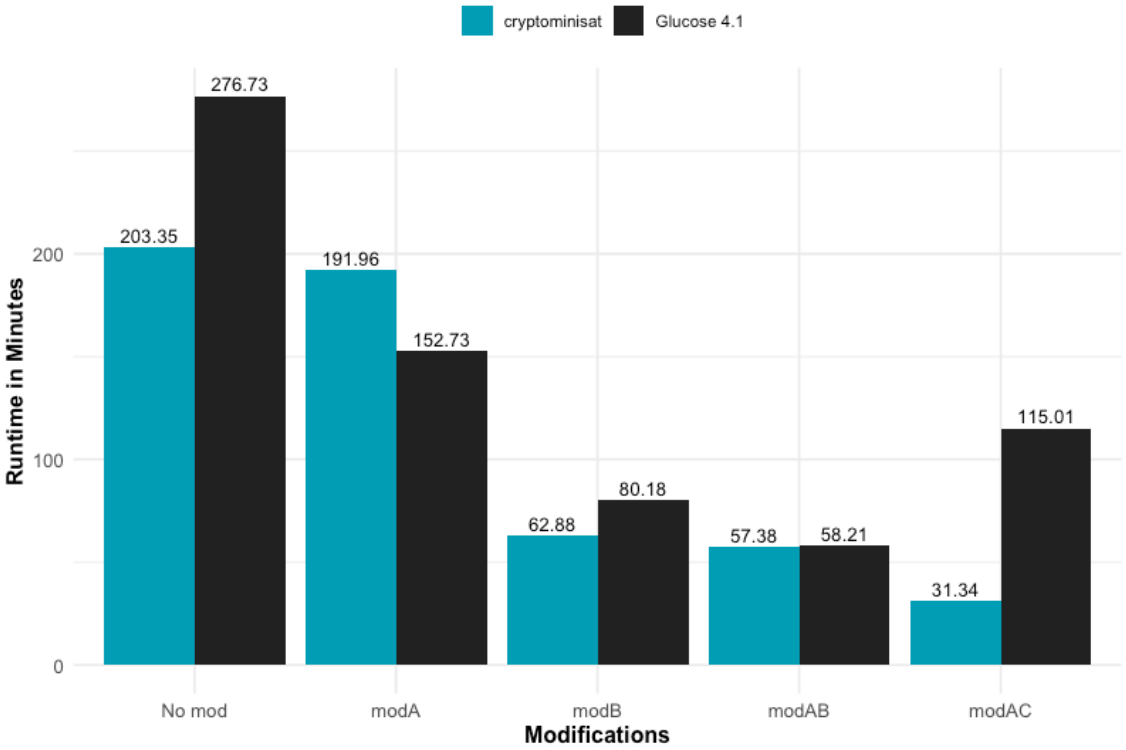
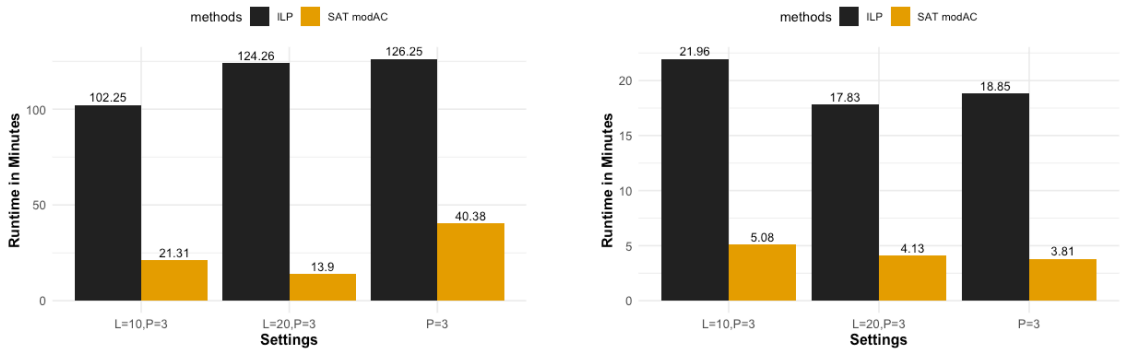


Figure 6: Comparison of all LOBICO SAT modifications tested with solvers cryptominisat and Glucose 4.1.

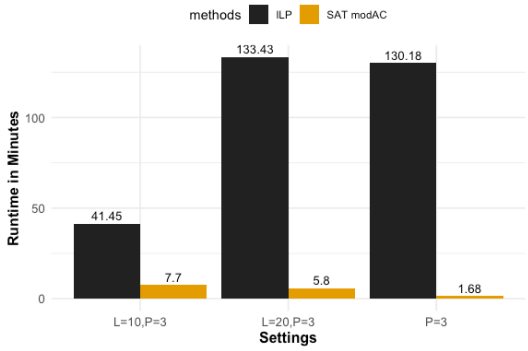
The bar chart in Figure 6 shows that LOBICO in general has a higher runtime than LOCATOR. Besides, we can see that the runtimes of the Glucose4.1 solver decrease from "No mod" to "modAB" but for "modAC" the runtime increases again. This result is very unexpected as the configuration "modAC" has the smallest instance size. For solver cryptominisat we observe that the runtime decreases with each configuration from "No mod" to "modAC". As expected we see that the "modAC" formulation is the fastest. Comparing the two different solvers one can see that in most cases the cryptominisat solver can solve the instances much faster but interestingly for "modA" the Glucose4.1 solver is faster. Overall the "LOBICO SAT modAC" formulation with the cryptominisat solver seems to be the fastest configuration. Because of that we will be using the "LOBICO SAT modAC" formulation together with the cryptominisat solver in the

following. In Figure 7 one can see a runtime comparison between the ILP formulation and the "LOBICO SAT modAC" formulation for different configurations of LOBICO and different sizes of datasets. The datasets have size $N = 277; M = 50$, $N = 302; M = 50$, $N = 302; M = 150$, and $N = 332; M = 150$. While the ILP formulation used at least 7 GB of memory, the SAT formulation only needed 3 GB. It was difficult to find good configurations for LOBICO as the method produces large instances for the ILP and the SAT formulation thus the parameter P was not changed and the parameter L had to be greater than 10. The features in the datasets were also manually decreased by configuring a fixed number in the SVM-RFE method.

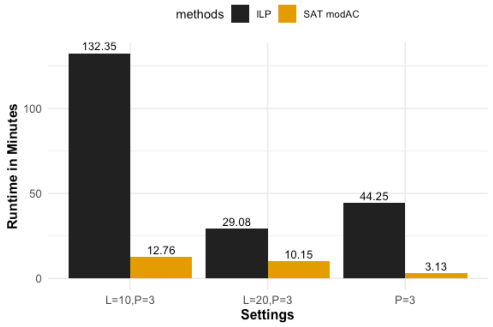


(a) $N = 277, M = 50$

(b) $N = 302, M = 50$



(c) $N = 302, M = 150$



(d) $N = 332, M = 150$

Figure 7: Comparisons between "LOBICO ILP" and "LOBICO SAT modAC" for different sizes of datasets and different configurations for L and P. If L is not specified then the number of features per minterm is not limited.

On the fist glance the results in Figure 7 seem to be very unexpected. Looking at the results of the ILP formulation one can see that in every bar chart the runtime increases or decreases differently when increasing the parameter L. Furthermore, in chart (b) we observe much lower runtimes than in chart (a) while (b) has a larger dataset. Comparing chart (a) with chart (d) we can see that the runtime in chart (d) and configurations $L = 20, P = 3$ and $P = 3$ is much lower even though the dataset has 100 features and 55 samples more. For the results from the SAT formulation one can recognize that the results are more expected because besides chart (a) and configuration $P = 3$ the runtime of the SAT formulation decreases when increasing the parameter L. However, also the results for the SAT formulation are a bit unexpected as the highest runtime can be found in chart (a) which is based on the smallest dataset. Comparing the ILP runtimes to the SAT runtimes we can see that in all experiments the SAT formulation

is much faster than the ILP.
Additionally, we want to give a small insight into the runtime of the SAT formulations. As for LOCATOR, we will be looking at the runtime percentage of UNSAT iterations and the number of UNSAT iterations divided by the total number of iteration. Table 4 shows the results.

M, N	$L = 10, P = 3$	$L = 20, P = 3$	$P = 3$
50, 277	94% | 11%	87% | 11%	97% | 11%
50, 302	49% | 10%	58% | 10%	17% | 10%
150, 302	16% | 9%	17% | 9%	21% | 9%
150, 332	39% | 9%	52% | 9%	55% | 9%

Table 4: UNSAT runtime percentages and UNSAT iteration percentages. Each row shows the results for a different dataset with N samples and M features. The columns represent the results for one configuration of the formulation. Additionally, the left values of an entry show the percentage of the summed UNSAT iteration runtimes out of the whole runtime and the right values of an entry show the percentage of the number of UNSAT iterations out of the number of all iterations. Note that the values are all rounded.

We observe that for LOBICO the runtime of UNSAT iterations is way higher than an average iteration runtime. In all cases the left value showing the runtime percentage of UNSAT iterations is higher than the right showing the relation of the number of UNSAT iterations to the number of all iterations. For our smallest dataset with 50 features and 277 samples we can see that UNSAT iterations had the largest share of the total runtime compared to the other experiments. The dataset with 150 features and 302 samples shows that the runtime percentage of UNSAT iterations can also be smaller and closer to the results from LOCATOR. All in all Table 4 shows that the solver needs a lot more time to prove that a LOBICO SAT instance is not solvable than finding a solution for an instance.

5 Discussion

In this section we want to critically analyze the SAT formulations and their results presented in this thesis.

5.1 SAT Formulations

First we want to have a look at the different LOCATOR and LOBICO SAT formulations regarding their complexity and thus their usability. In Sections 3.2.3 and 3.3.3 we introduced five different configuration of the SAT formulation for both LOCATOR and LOBICO. As seen in the results in Section 4 for larger datasets only the formulation "SAT modAC" was used. This was done as the number of clauses and variables in the other modifications were too high such that also the runtimes increased drastically. For LOCATOR and modifications "No mod" and "modA" we can calculate a lower bound for the number of clauses with $\binom{2M}{L+1} \in O(M^L)$. Already for small M and L we observe that the number of clauses gets really big. For example with $M = 40$ and $L = 3$ we have more than 1.5 million clauses for LOCATOR. If we want to have a lower bound for the number of clauses for LOBICO we need to multiply the lower bound of LOCATOR with P and get $P\binom{2M}{L+1} \in O(PM^L)$. Given $M = 40$, $L = 3$ and $P = 3$ we get more than 4.7 million clauses for LOBICO. The dataset used in this thesis had 13150 features which could be reduced by the SVM-RFE to about 250 without losing accuracy, still the number of clauses for LOCATOR and a small L of 3 would be more than 7.7 billion. As we want to use P and L to adjust our DNF formulas and counteract overfitting or underfitting of these formulas, the modifications "No mod" and "modA" are not usable for larger datasets with a lot of features. Overfitting means that a model is trained so precisely on the available data such that new data cannot be reliably classified anymore, whereas underfitting means that the model is not trained precisely enough on the data to reliably classify new data.
With "modB" and "modAB" we introduced two new modifications which mainly reduce the number of clauses used in the formulations by removing not needed clauses. Compared to the first two modifications we decreased the number of clauses by about 7 %. For our example with 250 features this means that the formulation consist of approximately 500 million clauses less for LOCATOR. As the total number of clauses is still more than 7.2 billion, these two modifications are also not feasible for large datasets.
Modification "modAC" now uses a completely different approach to limit the number of literals per minterm and thus can reduce the number of clauses a lot. Instead of the binomial coefficient as lower bound, the number of clauses can be calculated with a second degree polynomial in N and M and furthermore is independent of the configuration parameter L. Formulation "modAC" for our LOCATOR example with 40 features uses less than 15% of the number of clauses than the modifications before. Additionally, for larger examples this number increases as the number of clauses for "modAC" grows slower than in the other modifications.
The number of variables and clauses generated by "modAC" can be calculated with the following formulas (if L < M):

LOCATOR:

$$V(N,M) = 4M^2 + N^2 + 6M + 3N + 2 \tag{104}$$

$$C(N,M) = 12M^2 + 3N^2 + NM + 7M + 4N + 3 \tag{105}$$

LOBICO:

$$V_2(P,N,M) = 4M^2P + N^2 + 6PM + NP + 3N + 2P + 1 \tag{106}$$

$$C_2(P,N,M) = 12M^2P + 3N^2 + NMP + 9PM + 3PN + 4N + 2P + 1 \tag{107}$$

Modification "modAC" makes the SAT formulations for LOCATOR and LOBICO more usable for smaller datasets as Figure 5 and 7 show, but it is still not usable for bigger datasets. The main dataset was reduced to $M = 250$ features and includes $N = 6640$ samples, this then makes the SAT formulation consist of 44.361.022 variables and 134.707.113 clauses for LOCATOR. These SAT formulations are still quite large and thus we were not able to solve them for the whole dataset and had to at least reduce the number of samples for LOCATOR and for LOBICO we also had to reduce the number of features.
If we do not limit the number of literals per minterm our SAT formulations further decrease in size, as we do not need to worry about limiting the minterm to at most L literals. This makes modifications "No mod" and "modB" to be equal and modifications "modA", "modAB" and "modAC" to be equal. We lose control over the complexity of the generated DNF formulas but at the same time the formulation can be solved a lot quicker. Setting $L = M$, the formulation's size for "modA" can be calculated with the following formulas:

LOCATOR:

$$V_3(N,M) = N^2 + 2M + 3N + 1 \tag{108}$$

$$C_3(N,M) = 3N^2 + NM + 4N + M + 2 \tag{109}$$

LOBICO:

$$V_4(P,N,M) = N^2 + 2PM + NP + 3N + P + 1 \tag{110}$$

$$C_4(P,N,M) = 3N^2 + NPM + 3PM + 3PN + 4N + P + 1 \tag{111}$$

For the given dataset, this leads to an instance size for "LOCATOR modA" of 44.110.021 million variables and 133.955.612 clauses. We can see that the number of variables and clauses reduces a bit but the number of samples in the dataset makes the instance still very large. Even though the difference in instance size is not that big for datasets with more samples, we can see in the results that the runtime decreases a lot for datasets with less samples. This is because for less samples the percentage of clauses and variables added by limiting the number literals in each minterm is higher.

5.2 Comparison of ILP and SAT

From the results in Section 4.1 we can see that the LOCATOR SAT formulations "No mod", "modA", "modB", and "modAB" are not even worth comparing to the LOCATOR ILP formulation since they produce too large instances and thus do not even come close to the results of the LOCATOR ILP formulation. LOCATOR modification "modAC" is much more competitive against the ILP formulation, especially when using large values for L or if we do not specify L at all. We observe that for $P = 3$ in (a) and (b) in Figure 5 the SAT formulation is faster than the ILP formulation and for (c) they are almost equal. However, as we want to use the parameter L to control the complexity of the DNF formulas these results are not as important. When comparing the SAT and ILP formulation of LOCATOR with the complete dataset and configuration $L = 3, P = 3$ we observe that for the ILP each class can be calculated in under two hours using 24 CPUs and 32 GB of RAM per class. For the SAT formulation we can only use one CPU as using k CPUs would cause the memory usage to be at least k times the size of the single CPU run, which is at about 25 GB if we specify $L = M$ and higher if $L < M$. Furthermore, using more CPUs meant we had to recompile the SAT solver (cryptominisat) and set certain compiler flags for enabling the program to use more memory. Still, even with more CPUs and the new compiled solver we were not able to achieve any results with a wall time of 24 hours. Additionally, we tested multi-CPU usage for smaller instances to test the behavior and observed that the runtime was always worse than the runtime with only one CPU. Finally, we tested the LOCATOR SAT formulation for $L = M$ and $P = 3$ with one CPU and 32 GB of RAM per class and got no results after 24 hours. From our results we know that if "LOCATOR SAT modAC" does not produce DNF formulas for $L = M$ in 24 hours then it will not produce formulas for $L < M$ in 24 hours as well. All in all we can say that in some special cases the SAT formulation "modAC" can be faster but generally the ILP seems to have a better runtime. Looking at the results of the LOBICO SAT and ILP formulations in Section 4.2 one can see completely different results. The LOBICO SAT formulation was much faster than the ILP formulation throughout all datasets and all configurations. However, because of the generally increased complexity of LOBICO we only used small datasets and thus these results might not be really feasible as datasets are almost always larger. Besides, we have to keep in mind that the results were achieved with only one CPU per class and thus the CPUs for the ILP formulation could be further increased to speed up processing time without increasing RAM usage. We did not tested the multi-CPU runtime of the LOBICO SAT formulation because of the results from LOCATOR. Still, in a scenario with a small dataset and only very limited computation power we can definitely say that the runtime of the SAT formulation is less than the runtime of the ILP formulation for LOBICO.

Another problem is that proving a SAT instance to be unsatisfiable is in most cases the slowest part of using SAT formulations [7]. One can already see that in Table 4 where in most cases an UNSAT iteration for LOBICO took a lot longer than a normal iteration where we found a solution. Although this is not a problem for LOCATOR with the reduced datasets, it might

become a problem when the datasets are larger and the complexity of the LOCATOR formulation increases. We can observe this trend in an experiment we did with approximately 1000 samples and 250 features for configuration $L = 3, P = 3$. In this experiment the runtime of UNSAT iterations already made up 35 % of the whole runtime while the number of UNSAT iterations was at only 11 %. Note that class 1 was not included in this experiment as it did not finish under 5 hours.

5.3 Further Improvements to SAT Formulations

In Sections 3.2.3 and 3.3.3 we already introduced some improvements to our SAT formulations. While implementing the different SAT formulations, we have found some further improvements that are not directly reflected in the formulations. First we found out that instead of just adding the variable $t_{N+1,z}$ into the assumptions we can add all variables from $t_{N+1,1}$ to $t_{N+1,z-1}$ into the assumptions as well. We can do that because the counting structure sets all values $\{t_{N+1,z-x} \mid x < z\}$ also to True if $t_{N+1,z} =$ True. For the experiment with configuration "LOCATOR SAT modAC" in Figure 4 this improvement reduced the runtime by approximately 20 %. In SAT formulation "No mod", "modA", "modB" and "modAB" we can only use this improvement for counting the objective value but for "modAC" we can also use it in the structure that counts the number of literals in the minterm. All results from Section 4 already include this improvement.

Furthermore, there might be some more improvements possible, to at least speed up the runtime for LOCATOR and smaller datasets. As we can see in Table 3 proving UNSAT did not increase the runtime at all such that methods like binary search could find the maximum value quicker than our current approach. This improvement was beyond the scope of this thesis and thus not implemented and tested.

Another improvement that decreases the number of iterations could be implemented by evaluating the solution of a satisfiable iteration. If the solution has variables $t_{N+1,z+x} : x > 0$ that are equal to True then the solver found a solution that has a higher objective value than we were searching for and thus we could increase z by the greatest x value. If we do not find such a variable $t_{N+1,z+x} : x > 0$ then we can still just increase z by one. This improvement was also not implemented and tested.

6 Conclusions

In this thesis, we introduced new formulations for LOCATOR and LOBICO which construct logic models for aiding in cancer research and diagnostics. The SAT formulations construct the same logic models as the existing ILP formulations, disregarding minor changes in the objective functions.
Comparing the base SAT formulation to the best modification "SAT modAC" we achieved good results in reducing the complexity and consequently the runtime of the SAT formulation. Still, "SAT modAC" could not construct logic models for larger datasets. For LOCATOR we observed that in most cases the ILP formulation was able to construct the logic models faster than our best SAT formulation even with very limited computational resources.
For LOBICO the SAT formulation was much faster on the tested datasets when using limited computational resources. However, this runtime advantage of the SAT formulation becomes less significant when running the LOBICO ILP formulation with more CPUs.
For future work there might be room to improve the SAT formulation's runtime, especially by excluding iterations with heuristics as described in the discussion or by using another SAT solver that is able to use more CPUs at the same time without the need of additional memory.
As a conclusion, the SAT formulations introduced in this thesis can be useful in special cases, but in most cases the ILP formulations are better as they construct the same models in less time.

Data and code availability.
Data and code for LOCATOR and LOBICO is available at https://gitlab.cs.uni-duesseldorf.de/albi/albi-students/ba-hendrik-schmitt.

7 References

[1] *Cancer Statistics WHO*. URL: https://gco.iarc.fr/today/fact-sheets-cancers.

[2] Nguyen Khoa Tran. "Logic Models to Classify Cancer Types based on Tumor DNA Samples". MA thesis. Heinrich Heine University Düsseldorf, 2021.

[3] Theo A Knijnenburg et al. "Logic models to predict continuous outputs based on binary inputs with an application to personalized cancer therapy". In: *Sci Rep 6, 36812* (2016). URL: https://rdcu.be/cyaC6.

[4] *cBioPortal for Cancer Genomics*. URL: https://www.cbioportal.org/.

[5] K. P. Soh et al. "Predicting cancer type from tumour DNA signatures". In: *Genome Medizine 9* (2017).

[6] Stephen A. Cook. *The complexity of theorem-proving procedures*. ACM, 1971, pp. 151–158.

[7] Dan Gusfield. "From Integer Linear Programming to SAT-Solving in Computational Biology". In: The 2020 ALGO/WABI Conference, Sept. 2020.

[8] *High Performance Computing*. URL: https://www.zim.hhu.de/forschung/high-performance-computing.

[9] Johannes Köster and Sven Rahmann. "Snakemake a scalable bioinformatics workflow engine". In: *Bioinformatics* 28.19 (2012). eprint: https://academic.oup.com/bioinformatics/article-pdf/28/19/2520/819790/bts480.pdf.

[10] *Gurobi Optimizer*. URL: https://www.gurobi.com.

[11] *Cryptominisat Solver*. URL: https://github.com/msoos/cryptominisat.

[12] *Glucose 4.1 SAT Solver*. URL: https://www.labri.fr/perso/lsimon/glucose/.

[13] *PySAT interface*. URL: https://pysathq.github.io.

[14] *Lingeling SAT Solver*. URL: http://fmv.jku.at/lingeling/.