Exploring factors that affect performance on introductory programming courses

Krista Longi

Master's Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, September 26, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Krista Longi

Työn nimi — Arbetets titel — Title

Exploring factors that affect performance on introductory programming courses

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's Thesis	September 26, 2016	70

Tiivistelmä — Referat — Abstract

Researchers have long tried to identify factors that could explain why programming is easier for some than the others or that can be used to predict programming performance. The motivation behind most studies has been identifying students who are at risk to fail and improving passing rates on introductory courses as these have a direct impact on retention rates. Various potential factors have been identified, and these include factors related to students' background, programming behavior or psychological and cognitive characteristics. However, the results have been inconsistent.

This thesis replicates some of these previous studies in a new context, and pairwise analyses of various factors and performance are performed. We have data collected from 3 different cohorts of an introductory Java programming course that contains a large number of exercises and where personal assistance is available. In addition, this thesis contributes to the topic by modeling the dependencies between several of these factors. This is done by learning a Bayesian network from the data. We will then evaluate these networks by trying to predict whether students will pass or fail the course. The focus is on factors related to students' background and psychological and cognitive characteristics.

No clear predictors were identified in this study. We were able to find weak correlations between some of the factors and programming performance. However, in general, the correlations we found were smaller than in previous studies or nonexistent. In addition, finding just one optimal network that describes the domain is not straight-forward, and the classification rates obtained were poor. Thus, the results suggest that factors related to students' background and psychological and cognitive characteristics that were included in this study are not good predictors of programming performance in our context.

ACM Computing Classification System (CCS):
Social and professional topics ∼ Computer science education
Computing methodologies ∼ Bayesian network models

Avainsanat — Nyckelord — Keywords
Computer science education, CS1, Bayesian networks

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Researchers have long attempted to identify factors that could explain why learning to program is easier for some than the others. That is, the goal has been to determine what contributes to the ability to learn to program, *programming aptitude*. Also, being able to predict students' performance on programming courses provides an opportunity for early interventions. Several factors that potentially affect programming aptitude or performance have been identified in many previous studies. These include factors that are related to students' background, programming behavior and psychological and cognitive characteristics. However, there is little consensus on these factors as the results have often been inconsistent, and many studies need further verification of the results in different contexts. This thesis focuses on both background and psychological and cognitive factors and takes steps towards understanding their relationship with programming aptitude and performance.

The research on predicting performance on introductory programming courses dates back over 40 years [21]. It initially started with the need to recognize good employees [14], but the focus has since shifted to predicting student's performance when studying programming [2, 21, 112, 113, 115, 119]. Special interest has been put into studying the introductory programming course at universities, often referred to as CS1. This course is especially important as success in the beginning can determine whether the student continues with the major or not. Many Computer Science programs have reported poor retention rates [7], and it can be possible to improve these rates through the introductory courses.

Introductory programming courses have struggled with high failing rates with estimated one-third of the students failing on average worldwide [10,114]. Being able to recognize students who struggle with their studies allows us to offer additional help and support to these students. Introductory courses often have a lot of participants, and thus it is not always possible for the lecturer to know how the class is doing before the final exam. Early interventions can be beneficial for many students, so having a method to recognize the need for help early on is important. When recognizing the students at risk,

interventions can also be targeted only to this particular group.

Identifying the factors that contribute to programming aptitude can also help us to understand how students learn to program, which in turn can help to plan the needed intervention. This knowledge can also be used to improve teaching in general. Same teaching methods do not work for everyone and identifying students individual needs can be very beneficial. For example, the same teaching techniques that help inexperienced students might not work for experienced learners [55].

In addition, many countries have recently introduced or are planning to bring programming as a part of the curriculum in primary or secondary school [9, 39]. This creates additional challenges as the students are even younger, and on the other hand, the teachers might not have any previous experience in teaching programming. Thus, a better understanding of the learning process is especially important when planning these new curriculums.

However, in many countries programming is not a part of the curriculum even in secondary school yet [39]. This can be a problem when considering admissions to programming related degrees. As the students have no previous knowledge of the field, the admission criteria need to be based on success in other fields and other factors. Knowing what kind of background and characteristics help students to succeed in their programming courses helps to plan these admission criteria.

Previous studies have been able to identify numerous potential predictors of performance, including variables related to students' background, psychological and cognitive characteristics and programming behavior. This thesis focuses on factors that do not change or change slowly; that is the demographic factors as well as different psychological and cognitive factors. Most previous studies have only focused on one or few factors, and thus many of them have only been studied in one or two contexts, and further replication of these results is needed. As variables related to programming behavior model students' current knowledge and skills, these variables have been excluded. Factors related to programming behavior have already been found to be good predictors in different contexts [2, 22, 80, 113]. The results with other variables are less clear, but better knowledge could help to improve introductory programming courses.

Most of the studies to date have been studying simple correlations between different predictors and the performance measure [11–13, 18, 113, 118], but other methods such as t-test [11, 113] or linear regression [88] have also been used. However, the studies also have many limitations, and the results between different studies have been inconsistent and varied depending on the context. Thus replication of the previous studies in different contexts is needed, and computing the correlations for several different variables also in our context is one of the contributions of this thesis.

The method of using simple correlations has many limitations. For example, correlation does not always imply causality but the two correlating variables can be coincident effects of a common cause. Correlations can also only find linear relationships. In addition, these predictors are most likely not independent of each other, so understanding the relationships between the predictors can also bring valuable information. Modeling the domain as a whole allows us to flexibly solve different questions from classification to prediction. Speculating about causalities is also possible using the dependencies. Constructing a causal network that describes the domain can help in answering questions like what will happen, if we can, for example, improve the learning strategies of the students. Ideally, including more predictors can also improve the prediction accuracy. Thus, another contribution of this thesis is to simultaneously study several different variables, their relationships, and effect on performance.

We have chosen to use Bayesian networks [78] to model the domain. Bayesian networks are graphical models that describe a joint probability distribution for the whole domain in an intuitive and compact way. They have been widely used in different fields such as medicine, finance, and industry [84]. They are a tempting tool for studying student performance, as they are able to model uncertainty, and the graphical model is intuitive to read. Also, as Bayesian networks have gained a lot of interest in recent years, several tools and tutorials for practical applications exist [4, 71, 98]. Bayesian networks have already been applied in the context of predicting educational performance [8, 48, 99], just not in the context of programming.

This thesis is organized as follows. In Sec. 2 we explore the factors that have been used to predict success previously in the literature. Section 3

provides an introduction to the method used in this study, Bayesian networks. The methodology and research questions are presented in Sec. 4 and the results in Sec. 5. Finally, the results are discussed in Sec. 6, and Sec. 7 provides conclusions as well as some suggestions for future work.

2 Performance on introductory programming courses

Predicting performance on introductory programming courses is a widely studied problem, and the motivation behind these studies is usually the high failing rates [10, 114]. Ideally, we want to be able to recognize the students who are struggling early on during the course, so that those students can then be offered additional help and support. On the other hand, successful students could be offered additional challenges to also improve their learning experience. Understanding the reasons behind failing or succeeding can help to plan these interventions and teaching methods in general.

The work started with the need to recognize good employees and to predict their success in training [14]. Since then, the main focus of research has shifted towards studying programming courses, especially introductory programming courses. Programmer Aptitude tests (PAT) were a common tool at the beginning for companies to find employees, but often the test results did not strongly relate to performance on the job [70]. Later, studies have proposed various predictors. These include factors related to students' background, such as previous academic success [11,93,113,119] and previous programming experience [11, 45, 49, 93, 107, 113, 117, 119] as well as psychological and cognitive factors, such as self-efficacy [85,116,119] and self-esteem [12,113]. Some studies have also included demographics like gender, age and major [11, 93].

More recently, as it has become more and more common to collect log data on introductory programming courses [51, 89], newer studies have also included variables based on this data [22,52,112,113]. These variables try to capture students' behavior while they are solving exercises, for example by taking into account how much time they spend dealing with errors.

When predicting performance, and important thing to consider is of course how to measure it. As most studies have focused on predicting performance

on introductory programming courses, the final grade [12, 13, 85, 93, 116] or midterm grade [80, 118, 119] is a natural choice. Often this means a number between 0 and 100 [12, 13, 18, 119] or on a smaller scale such as 1 to 5 or similar [93, 117]. In most cases, the grade consists of performance in exercises and the final exam, though final exam usually makes up for most of the grade [12, 13, 116, 118]. Exam and lab performance have also been examined separately [107], and other more specific performance measures have also been used [2, 62].

This chapter introduces some of the most common predictors used in literature collecting results from several different studies. These can roughly be divided into three categories: student's background, psychological and cognitive factors and data-driven factors. All these three categories are discussed below. Section 2.4 then describes the methods used in these studies. Most of the studies to date have examined correlations between these factors and a performance measure, but there are also some studies that have aimed to examine the dependencies between different predictors.

2.1 Student's background

Different factors related to students' backgrounds are attractive as predictors as they are usually easy to collect, or they might be already available. For example, grades from secondary school have often already been collected during the admission process. Also, variables related to students' backgrounds are usually used as admission criteria to universities. If we can better understand which of these characteristics allow students to succeed in their studies in computer science, we can perhaps improve the admission processes. However, some information, like previous programming experience, can often be only obtained through questionnaires and therefore requires more effort to collect.

The effect of especially programming experience on programming performance has been studied by many researchers [45, 49, 113, 117–119]. According to many studies, previous programming experience does seem to have an impact on the success of the students on introductory programming courses [45, 49, 113, 118, 119], but some studies did not find any significant

relationships between these two variables [11, 107]. Watson et al. discovered that while previous programming experience can predict the success of students, specific variables such as the number of languages known had only weak correlations with the performance [113]. In addition, while Holden and Weeden found a difference in the results of experienced and non-experienced students on the first introductory course, this difference disappeared in the later courses [49].

Some studies have found that also non-programming related computer experience can be related to programming performance [118, 119], but this has been studied much less than previous programming experience. For example, playing computer games has been shown to have a negative effect on the course grade [118, 119].

In addition to previous experience, researchers have studied the effect of previous academic success. Researchers have been especially interested in the influence of mathematics background on success in programming, and it has been found to be a significant predictor in many studies [11, 18, 118, 119]. Wilson et al. even found mathematics background to be a stronger predictor than previous programming experience [119]. However, as with previous programming experience, there are also studies that did not find any link between previous math background and success on the introductory programming course [107, 113].

Though mathematics background has been most extensively studied, some studies have included success in other subjects. Bergin and Reilly found that previous physics and biology grades have moderate but significant correlations with performance in the final exam, but surprisingly there was no relationship between chemistry grade and the final exam score [11]. Byrne and Lyons also explored the predictive power of grade achieved in the native language or second language, but they found no significant correlations [18]. Watson et al. found no significant correlations between college grade point average (GPA) and success on an introductory programming course, but high school GPA was a moderate predictor [113].

Many computer science programs also suffer from having only a small minority of women as students [32, 87]. This is perhaps one reason why many researchers have also been interested in the effect of gender in success in

introductory programming courses. Perhaps males have better programming aptitude or females are discouraged by the male-oriented study environment. However, most studies have found that gender has no effect on programming performance [18, 81, 107, 113, 115, 118], though different results also exist [63].

2.2 Psychological and cognitive factors

There has been a lot of interest in using different psychological and cognitive factors extracted from questionnaire data to predict students' performance on a programming course. These are known to affect success, as good performance is not only dependent on skills, but also individuals ability to use those skills effectively. For example, self-efficacy affects many aspects of learning like persistence and use of cognitive strategies [5]. Weak self-efficacy can be improved with positive personal experiences [5], and this can, in turn, be a way to improve students' performance.

Studying psychological factors as predictors can be helpful in understanding the process that students go through when learning. Thus, the information can be useful when designing the teaching methods. However, the problem with psychological and cognitive factors is that they often require questionnaires or tests, that can sometimes be lengthy and require time to process. Therefore, these factors might not be helpful predictors if the goal is to intervene early on during the course. However, the variables can still bring valuable information about what kind of students are at risk to fail. This section presents the most common psychological and cognitive factors used in computer education studies. Most of these have been applied in other fields as well as in computer science.

Different students have different ways to approach new material and learning tasks as well as to process information, and these traits are called students' *learning styles*. Several theories of learning styles exist, but two of the most common ones are *Kolb's Learning Style Inventory* (LSI) [57] and *Gregorc style delineator* (GSD) [44]. LSI has a standardized questionnaire, where individuals rank potential endings for 12 sentences on a scale of 1 to 4. The result is a score for the individual's predisposition toward concrete experience, reflective observation, abstract conceptualization, and active

experimentation. However, the correlations between the four dimensions and performance in programming courses were found to be either not significant or weak [19, 23, 30, 113]. GSD uses four dimensions similarly to LSI: concrete random, concrete sequential, abstract random, and abstract sequential [44]. The dimensions have shown to have weak or moderate correlations with programming performance [62, 63, 113].

Motivated Strategies for Learning Questionnaire (MSLQ) [82, 83] is a questionnaire designed to assess students' motivations and learning strategies. It consists of two parts, a motivation section, and a learning strategies section, that can be administered individually. The learning strategies section measures nine different scales regarding students' cognitive, meta-cognitive, and resource management strategies [83]. The motivation section consists of six scales that can be divided into three categories: value, expectancy, and affect [83]. Not many studies have applied MSLQ in the context of programming yet. For the motivation section, two studies have found significant correlations between success and total score for MSLQ, intrinsic goal orientation, and self-efficacy for learning and performance. [12, 113]. One of these studies also found correlations with task value and control of learning beliefs [12]. The results with learning strategies have been more inconsistent. While Bergin et al. found significant, strong correlations for seven out of nine scales [13], Watson and Godwin found significant, moderate correlations only for critical thinking and effort regulation [113].

Self-efficacy is a person's judgment of their ability perform tasks and reach goals [5]. Self-efficacy is related to certain domains, which means that person's self-efficacy can vary depending on the task. Self-efficacy for learning and performance is also one of the subcategories ins MSLQ motivation questionnaire. As mentioned above, it has been found to correlate with success [12,113]. Self-efficacy has also been studied with other questionnaires, for example by using the Computer Programming Self-Efficacy Scale developed by Ramalingam and Wiedenbeck [86]. It has been shown to correlate with success in introductory programming courses [107]. However, Wiedenbeck et al. found no significant correlations between pre-self-efficacy, that is the self-efficacy measured at the beginning of the course, and performance [117]. Self-efficacy measured at the end of the course, post-self-efficacy, correlated

with the performance also in their study [117]. On the other hand, pre-self-efficacy has been shown to even have a negative correlation with performance among non-majors [116].

Another factor that has been studied is the explanations that students give to their success, *attribution of success*. The students were asked to rank four reasons for their success: attribution to ability, attribution to task difficulty, attribution to luck, and attribution to effort. Again, the results have been somewhat inconsistent. Attribution of success to luck has been shown to correlate with performance on introductory programming courses by three studies [107,113,119], but only two studies found correlations with attribution to task difficulty and efforts [112, 119] and one study with attribution to ability [119].

Self-esteem is a person's evaluation of their worth. Rosenberg's self-esteem scale (RSE) [92] has been widely used as a measure for self-esteem. It contains ten statements that the students evaluate on a four-point scale. Though several studies have linked self-esteem to achievement at school [50] and RSE is a widely used measure in general, only two studies have measured RSE in the context of programming. These studies had questions that were modified to relate them to programming, and while Bergin et al. found a moderate correlation between the score and performance [11], Watson et al. found no significant correlation [113].

The variables mentioned above are perhaps the most studied ones, but researchers have also been interested in predictors like mental models [85,117], comfort level [118] and achievement goals [124]. For example, Wilson found that comfort level was the strongest predictor out of 12 variables in their context [118].

2.3 Data-driven variables

As the amount of data collected on introductory programming courses constantly increases, there has been more and more interest in using this data to predict success in introductory programming courses. Several systems that collect data while students solve their exercises have been developed [17, 109]. These systems collect data with different granularities: on submission level,

snapshot level or keystroke level [110].

Data-driven variables can offer information fast without requiring any extra effort from the students. Many traditional predictors presented in Sec. 2.1 and Sec. 2.2 require the students to complete questionnaires or tests that can sometimes be lengthy. Also, processing these tests may require a lot of time and make early interventions difficult. With data-driven variables, the collection of the data is continuous, and thus it can also reflect the progress that students make during the course. Tests taken once during the course do not offer this possibility.

Data-driven variables aim to model students' programming behavior, and even variables that seem simple can predict student's success. Watson and Godwin extracted ten measures that were based on analyzing event pairings [113]. As they had previously discovered, that just counts of events were not good predictors [112], they measured the percentage of analyzed pairings out of all pairings. Nine of the tested measures had significant and strong or moderate correlations with performance [113]. Good prediction accuracy was also achieved by using features based on total amount of time spent and the correctness of the solution achieved on specific exercises [2]. Piech et al. constructed a graphical model that describes students' programming process, and this model was used to predict whether the student will struggle later on during the course [80].

There are also measures that have aimed to quantify students' overall programming behavior. The *Error Quotient* (EQ) was first introduced by Jadud to quantify students' struggle with syntax errors [52]. It examines consecutive pairs of compilation events, adds to the score if both compilation events in the pair end to an error, and adds extra points if the events end to the same type of error [52]. Jadud himself observed only weak correlations between the EQ and performance on the course, but his data only included programming data from times that students were working in computer labs, and therefore he was missing an unknown amount of data [52]. Other studies have found that EQ explains about 10 – 30% of the variance in students performance when using Java programming language [103, 112, 113] and about 3% when using C++ [22].

Watson et al. sought to improve the work started by Jadud [52] and

introduced their own measure, the *Watwin score* [112]. The Watwin score, like the EQ, is also formed by examining pairs of compilation events, but it introduces several new qualities. The method that Watson et al. use for pairing the events is different, and they also account the time that student's take between these events into the scores [112]. The idea of the measure is to give a bigger score to students who spend more time to resolve a specific type of error than their peers. The results with Watwin score have in general been better than with EQ [22, 112, 113], but still inconsistent. Watson et al. themselves found that Watwin score accounts for about 36–42% [112, 113] of the variance in students performance when using Java as the programming language on the course. However, according to another study, Watwin score accounts only for 12% of the variance in final grades, though the language used in this study was C++ and the course studied CS2 instead of CS1 [22].

While EQ and Watwin score focus on the compilation behavior of the students, Carter et al. [22] wanted a measure that also considers other aspects of programming behavior, like debugging and eliminating semantic errors. They introduced a new measure, a *Normalized Programming State Model* (NPSM), that aims to measure the syntactical and semantical correctness of students' programs. The NPSM is based on the time that a student spends in each of 11 possible states while programming relative to the total time spent programming [22]. Carter et al. themselves found that NPSM accounted for 36% of the variance in students' final grades [22]. The language that the students used in their study was C++ and the data was collected on a CS 2 course instead of a CS 1 course.

Also, different kinds of data-driven variables that are not based on programming data have been used. For example, students' web usage and participation in online discussions on a Moodle course have been used to predict performance [90, 91].

Using data-driven variables in predicting students' success in introductory programming courses is still relatively new, and fewer studies have focused on them than on background or psychological and cognitive factors. However, the results have been promising. Watson et al. found that data-driven variables, in general, are stronger predictors of performance than traditional variables [113]. They tested 50 different variables, and all the strongest

predictors were based on programming data, except for self-efficacy [113]. However, there are still issues. For example, these measures seem to yield somewhat inconsistent results depending on the context [79], for example when using different programming languages [22, 79, 112, 113].

2.4 Methods used for modeling effect on performance

Several studies have attempted to identify factors that affect success in introductory programming courses. The methods in most studies are very similar focusing on pairwise analyses between the variables and the performance measure. However, some studies have also used multiple variables to predict performance.

Most studies that have tried to identify good predictors have focused on studying the relationships between one predictor and the and course or exam grade independently by using either correlations [11, 12, 18, 107, 113, 115, 119] or simple linear regression [22, 88]. For example, Watson and Godwin analyzed the correlations between 50 different variables and course performance [113]. However, computing this may correlations raises the risk of obtaining statistically significant results by chance. This is known as the multiple comparisons problem. Many researchers have also performed t-tests to compare the mean scores of students grouped by some of the variables [11, 18, 81, 107, 118].

In addition, some studies have also attempted to model several factors at the same time. Wilson et al. utilized a general linear model to examine the effect of multiple variables. Rountree at al. used decision trees in predicting whether a student will fail or pass a CS1 course [93]. The data was collected with a questionnaire including questions about the students' background, working status and expectations. They were able to identify some features that put the students at risk of failing rather than explicitly identifying the students who will fail. Their results indicated that students who are likely to fail had similar answers to questions on academic background, math experience, year of study, age, and expectation of a grade [93].

Some studies have constructed a graphical model based on previous research and then used path analysis to test the model. For example, the

relationships between previous programming experience, pre self-efficacy, post self-efficacy, mental models and the final grade of the course have been studied this way [85, 117]. The unexpected finding here was that previous programming experience does not seem to have on effect on students' mental models unlike the researchers expected [85, 117]

Lau and Yu used Partial Least Squares (PLS) modeling to study relationships between six different variables: gender, learning styles, mental models, Band, MOI, and programming performance [63]. PLS modeling can be used to explore relationships between variables. It combines the previously mentioned multiple regression and path analysis among other similar methods, and again, in this case, the hypothesis of relationships was based on previous studies. All the studied variables were found to relate to performance except for gender, which only was related to mental models [63].

More recently, some studies have compared various machine learning methods in predicting performance [2, 80, 91]. For example, Ahadi et al. [2] tested the performance of different machine learning methods in predicting students' performance. They aimed to recognize whether the student would be in the high- or low-performing half of the class, and the variables used in the study were based on students' background and key log data. They achieved classifier accuracies as high as 93%, and the best results were obtained by using random forests [2]. Random forests learn multiple decisions trees and use averaging to get better results and to reduce over-fitting. Also Romero et al. experimented with different classification methods to predict performance based on participation in online discussions [91]. Piech et al. used Hidden Markov Models to learn a high-level representation of students' progress when programming, and then k-means clustering to find patterns in these representations [80].

3 Bayesian networks

The method we have chosen here to model the relationships between factors affecting programming performance is Bayesian networks. Bayesian networks [78] are probabilistic graphical models that can be used to represent uncertain domains. They can represent probabilistic dependencies in a set of

variables efficiently and naturally, and also help us to understand the domain as well as to predict future events based on collected data.

Bayesian networks have many features that make them useful for real-life situations such as student modeling. Because of the wide applicability, the interest in Bayesian networks has grown in different fields, and the methods have advanced rapidly. As a result, several commercial and open source packages are now available for researchers to use and apply in their context [4, 71, 98]. Fields, where Bayesian networks have been applied, are numerous, including finding relationships between genes, environment, and disease [102], analyzing gene expression data [43], environmental modeling [106] and much more [84]. Bayesian networks have also already been utilized in student modeling and cognitive assessment in general [8, 99].

One of the advantages of Bayesian networks is that they represent the joint distribution of all the variables, and thus allow modeling the domain as a whole instead of just focusing on one variable. Therefore Bayesian networks can flexibly be used for any inference task. Moreover, they represent the joint distribution in a compact and interpretable way. In addition, Bayesian networks can model uncertainty, which is inevitably present in student modeling. Bayesian networks can also be used to model relationships that are not linear. Many traditional models, such as regression, are not sufficient when the dependencies in the data are non-linear. Also, handling missing data can be naturally incorporated into the analysis, and when it comes to real-life data, especially data collected with questionnaires, missing values can be common. Bayesian networks can also make good predictions even when trained with a limited amount of data [58], and with student analysis the number of students can sometimes be small.

In addition, Bayesian networks can be learned just from the data without defining any initial structure. That means there is no need for initial model constructed by humans, which removes a certain risk to errors. Previous studies that have examined the relationships between variables that affect success on introductory programming courses have tested networks that have been solely defined by humans [85]. However, it is still possible to incorporate previous knowledge in the structure when it is available.

Finally, Bayesian networks are a tempting solution when we want to

visualize cause-effect relationships in a natural way. As the networks are directed, the direction of the causation can be seen intuitively from the structure. However, the edges between variables do not always indicate causality, and certain assumptions need to be made before cause-effect relationships can be examined. Causalities can provide valuable information about the domain, but the directions of the arrows are not relevant when it comes to probabilistic inference. Even without assuming any causal relationships, it is still possible to get meaningful information from the structure.

This section introduces Bayesian networks and the notation used in this thesis. It starts with describing the needed background information and important definitions in Sec. 3.1. An important part is learning the model, which is presented in Sec. 3.2. Section 3.3 then describes how to extract information from the model. Finally, Sec. 3.4 presents some of the previous work that has used Bayesian networks to model student performance.

3.1 Basics

Bayesian networks define a joint distribution for a set of variables in an intuitive and compact way. They are graphical models and consist of two components: a directed acyclic graph (DAG) and a set of conditional probability distributions. In the DAG, the nodes represent a set of random variables, and the directed arcs represent the conditional dependencies between these variables. Each variable also has a conditional probability distribution associated with it. An example of a Bayesian network can be seen in Figure 1.

More formally, a Bayesian network defines a joint distribution for a set of n random variables $X = \{X_1, ..., X_n\}$. As it contains two components, it is formally defined as a pair (G, θ_G), where G is the graphical representation, a directed acyclic graph (DAG), and θ_G the parameters associated with it. The DAG G is also defined as a pair (N, A), where $N = \{1, ..., n\}$ is a set of nodes, where each node v corresponds to one random variable X_v in the data, and A is a set of arcs between the nodes N. The arcs in G are directed, and by definition, the graph cannot contain any directed cycles. This means, that when starting from any node v there is no way to loop back to that

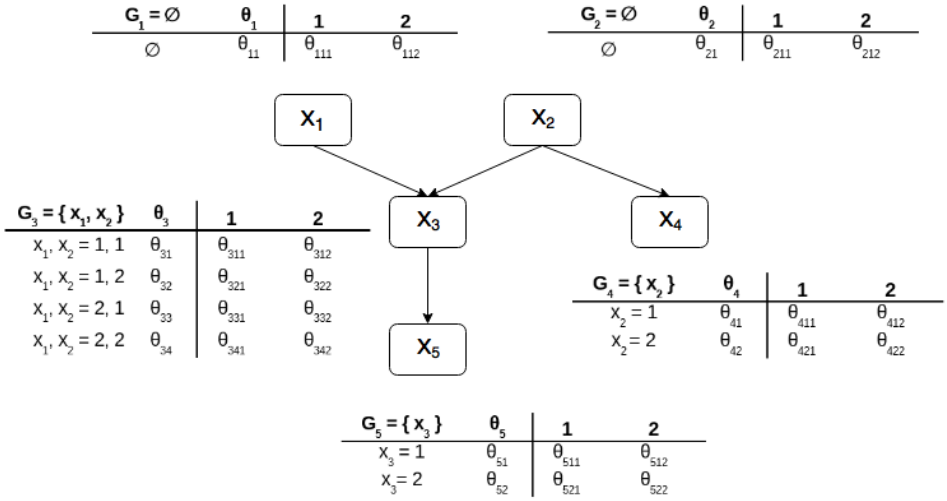

Figure 1: Example of a Bayesian network for variables $X = \{X_1, X_2, X_3, X_4, X_5\}$ that each can take two possible values. The tables represent the conditional probability distributions $P(X_i|G_i, \theta)$.

same node again by following the directed edges.

In the DAG, a node u is defined as the *parent* of a node v if there is an arc from u to v in the graph. Then, the node v is a *child* of the node u. For example, in Figure 1 node X_1 is a parent of node X_2. Bayesian network structure can be expressed as a vector $G = \{G_1, ..., G_n\}$, where G_i is the parents of variable X_i. For example, the network structure in Figure 1 can be represented as $G = (\{\}, \{\}, \{X_1, X_2\}, \{X_2\}, \{X_3\})$. Similarly, u is defined as an *ancestor* of v if there is a directed path from u to v, and v is then a *descendant* of u.

Each variable X_i has also table of parameters θ_i associated with it. θ_i defines the conditional probability distribution $P(X_i|G_i, \theta)$. The paths between the nodes in the graph express the probabilistic dependencies between the variables. An important feature, the *Markov condition*, states that each random variable is conditionally independent of all of its non-descendants given its parents. Thus, as the topology of the graph already holds information about the dependencies, the Markov condition allows storing joint probability

distributions of even large amount of random variables efficiently if there are many independences. The joint probability distribution for the set of variables $X = (X_1, ..., X_n)$ represented by a Bayesian network $M = (G, \theta_G)$ is given by

$$P_M(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|G_i, \theta_i) \quad (1)$$

For example, by taking advantage of the conditional independence relationships seen in the network presented in Figure 1, the conditional probability of the five variables can be written according to Equation 1 as:

$$P_M(X_1, ..., X_5) = P(X_1) \times P(X_2) \times P(X_3|X_1, X_2) \times P(X_4|X_2) \times P(X_5|X_3)$$

3.2 Learning Bayesian networks

This section gives a short introduction to learning Bayesian networks. When constructing a Bayesian network, we need to determine its' structure and the parameters. Out of these two, structure learning is often the harder case, and in this description, more emphasis is put on learning the structure.

Learning the structure can be done from prior knowledge, data, or from a combination of these two. One way to construct a network is to design it based on what we know about the domain. However, most of the time this is not possible as we might not have enough information about the domain, or the task would just be too time-consuming for a human. Therefore, several methods for learning the network from data exists. If available, prior knowledge can also be incorporated into this process. There are two main strategies for learning a network structure from data: *constraint-based approaches* and *score-based approaches*. Some hybrid algorithms that combine both of the approaches also exist.

The constraint-based approaches are based on conditional independence tests between the variables, and they aim to find a network structure that best explains the found dependencies [100, 108]. From the results of these independence tests, it is possible to, for example, decide the existence of an arc. Several different algorithms are based on this principle but use various methods, such as different kinds of tests and ways to interpret the

results. Examples of these algorithms are IC [108] and PC [100]. Even though constraint-based algorithms are generally fast and can be used even with larger networks [60], score-based approaches tend to produce better results with small data sets or when the independences are weak [122].

In score-based approaches, a set of potential DAGs is defined, and each of them is assigned a score that describes how well the graph represents the dataset. The best DAG is the one with the highest score, so, in the end, the goal is to solve an optimization problem. Most of the scoring criteria are based on either the *maximum log-likelihood* of the structure or on the posterior probability of the network. Bayesian Information Criterion (BIC) [95] and Akaike Information Criterion (AIC) [3] are both based on the maximum log-likelihood of the structure, to which they add a different penalty term to penalize models that are too complex to avoid over-fitting. Score functions that compute the posterior probability of the network are often referred to as *Bayesian scores*, and they are defined using the Bayes rule. Examples of Bayesian scores are K2 [29] and BD [47].

There are a couple of important properties that a score should fulfill. Most efficient learning algorithms advantage of *decomposable* scores [47]. The score is said to be decomposable if it can be calculated as a sum of scores for individual variables, and the score for each variable is only dependent on the variable itself and its parents. Also, the score should be the same for all *equivalent* networks. Two Bayesian networks are said to be equivalent if they define the same probability distribution.

However, the amount of different possible networks grows quickly when the number of variables grows, and finding an optimal network is an NP-hard problem [24, 26]. For moderate-sized data, it is still possible to use exact algorithms. Exact algorithms, unlike heuristic algorithms, always find the optimal solution. As using exact algorithms removes certain uncertainty from the results, there has been interest in developing exact algorithms, even though they run in exponential time. Many of these algorithms are based on dynamic programming [56, 76, 104], but others also exist [20, 38, 95].

In practice, heuristic algorithms are often used for structure learning. Examples of such algorithms are greedy search based algorithms, best-first search [59] and Monte-Carlo methods [42, 68]. Many heuristic algorithms

use decomposable scores to reduce the number of needed computations. As most of the algorithms add, remove or reverse existing arcs in the network, having a decomposable score means that it is possible to just calculate the change in the score. For example, greedy search uses this principle, and in each iteration, it calculates the change in the score for each possible change in the arcs, and chooses the change that maximizes the score [25]. However, like other local search algorithms, it can get stuck on a local maximum, but for example simulated annealing [27] have been used in attempts to solve this problem.

Most practical data contain missing values, and many structure learning algorithms described above cannot handle these situations as described. One approach to addressing the problem is to preprocess the data by either removing all entries with missing data or by filling in the missing values. Removing all observations with missing data can be used if only a relatively small portion of the observations include missing values. However, this is not always the case, and then it is better to impute the missing data. For filling in, some good guesses can be used, and the simplest solution is to us the mean or median of the observations. Better results can often be obtained by estimating a predictive model based on the other variables. For categorical variables, it is possible just to handle the missing values as legitimate by adding an additional category for them.

Some structure learning algorithms also work with missing values. One of the best-known ones is the *Structural Expectation Maximization* (SEM) algorithm [40]. It is based on the EM principle [34], and many different algorithms based on this principle have since been developed [16,66]. However, in many cases, these algorithms can get stuck on a local maximum. One solution to this is to use random restarts. Another popular group of methods that can handle missing values is based on the Monte Carlo techniques [31, 72, 77].

There are also other challenges to consider. The data might contain additional factors that are just not recorded in the data. This can be seen as a problem of missing data, where all the observations for one variable are missing. Thus, most approaches that are listed above to handle missing values can also be applied in this case. In addition, when the number of

variables grows, the number of possible data vectors increases exponentially with it. This means that we have usually only observed a small fraction of all the data vectors that could be possible to be observed. Also, it might be that the data cannot be naturally described with just one Bayesian network.

3.3 Inference in Bayesian networks

Once the Bayesian network model has been constructed, there is usually interest in extracting information from that model. This is called probabilistic inference or reasoning in the network. Typically, we are given the values for some subset of the variables, and we then want to determine the probability of some other variables being in a certain state. More formally, the aim is to determine a probability $P(S_1|S_2, G, \theta)$, where S_1 is the subset of the variables for which we want to determine the probability distribution when we are given the values of the variables in subset S_2 as well as the network G and its' parameters θ. For example, we could be interested in finding the probability that a patient has a certain disease given his symptoms.

As Bayesian networks define a joint probability distribution over all the variables, any inference problem can be solved with *marginalization*. Marginalization refers to summing out over all the irrelevant variables. However, this takes exponential time with respect to the number of nodes in the graph. Like structure learning, probabilistic inference in multi-connected networks has been shown to be NP-hard [28], and the inference algorithms can be divided to exact and approximate algorithms. Sometimes, especially in cases with discrete variables, using exact algorithms is possible, but often computing an exact solution takes too long and different heuristic approaches need to be used.

Exact inference means analytically calculating the conditional probability distribution over the variable or variables of interest. A common method for this is the variable elimination algorithm [33, 123], which is based on doing the marginalization more efficiently by utilizing a factored representation of the joint probability distribution. The idea behind it is to exploit the chain-rule decomposition of the joint distribution and to avoid repeating calculations by storing the already calculated results. However, the query

variables need to be specified in advance, and thus the algorithms needs to be rerun for every new query. The junction tree algorithms [6, 78] avoid this problem by generalizing the variable elimination algorithm. The network is transformed into a tree structure called a junction tree, where a message passing algorithm [54, 64] is then applied. The message passing algorithms are based on nodes passing messages to their neighbors and updating their conditional probability distributions based on the message.

Using exact algorithms is not always possible, and thus several approximate algorithms for inference in Bayesian networks have also been developed. There are many algorithms based on sampling or Monte Carlo methods, and for example Gibbs sampling and the Metropolis-Hasting algorithm can be utilized for approximate inferences [31, 72, 77]. Another group of algorithms is based on variational inference, for example, the mean field methods [74] and loopy belief propagation [121].

3.4 Bayesian networks in modeling student performance

Using Bayesian networks in educational data mining and especially in predicting students' performance has been studied in different contexts. Bayesian networks have been used for predicting performance also in the context of programming, but mainly with variables based on programming behavior and students' background [2, 91, 109]. Mostly these studies have experimented with several different classifiers, and Bayesian networks have just been one of the classifiers. The results below are reported as in the studies and can not be directly compared. The amount of classes in the tasks varies, and in imbalanced datasets, accuracy can be misleading.

Ahadi et al. used several different classifiers and variables based on students' background as well as key log data to predict whether the student would be in the high or low performing half of the class [2]. They achieved classification accuracies of 72%–86% depending on the performance measure when using Bayesian networks, but for example, random forests performed better in general. Similarly, Vihavainen used Bayesian networks and key log data to classify students into three groups: fail, pass and excellent [109]. After two weeks of collecting data, he was able to classify 64% of the students

correctly, and at the end of the course 78% of the students.

Romero at al. [91] also used Bayesian networks to predict performance on an introductory computer science course. A Moodle discussion forum was used on the course, and the dataset was constructed from the messages posted on this forum. They compared several different classification and clustering methods, including Bayesian networks, in predicting whether the student will pass or fail. They were able to achieve accuracies as high as 90%.

More examples of using Bayesian networks to predict success in educational contexts can be found in other fields. Bekele and Menzel applied Bayesian networks in predicting students performance on a mathematics course in a senior high school in Ethiopia [8]. Their study and the purpose of the study is very similar to the one presented in this thesis. However, the context and used variables are different. The model included eight variables including math performance, gender, group work attitude, interest for math, achievement motivation, self-confidence, shyness and English performance. They were able to predict correctly in about 64% of the cases whether the student would have below satisfactory, satisfactory or above satisfactory performance [8]. They had 514 samples after removing samples that had positive answers for included lie detector questions, and the most relevant attributes were previous English and math performance.

Also Sharabiani et al. [99] used Bayesian networks when aiming to predict the grade (A, B, C or D/F) of students' on three core course of an engineering program. They used different demographic and academic factors including gender, age, race, citizenship status and grades achieved on some prerequisite courses. In addition, their proposed model included variables that described the student's capacity each semester and the level of difficulty of each course for each student. They had included 300 students in the analysis, and their model achieved an accuracy of only 36% on the CS course but 70% and 73% on the physics and mathematics course [99]. They also compared the proposed model with other traditional machine learning methods, and in general produced the best results. Only in the CS course random forests method achieved a better accuracy with 46% [99].

There are also studies that try to predict students' performance at the

university in general, not just on one course. Hien and Haddaway [48] aimed to predict students' graduating cumulative Grade Point Average based on applicant background at the time of admission using a Naïve Bayes model, as that yielded best results out of all tested network structures. The motivation behind the study was to find a better way to recognize good applicants. The data set consisted of 1386 master's students and 212 doctoral students. The prediction accuracy of the master student model was 60% and the doctoral student model 88% when having five categories for the grade. The most important predictors were previous institution and previous cumulative grade point average. The biggest problem they faced with the study was that the model tended to overestimate the performance of students with lower grades due to the imbalance of data: most students accepted to the university have good grades, and thus they have very little data on students with low grade point averages (GPA) [48].

Thai-Nghe et al. [105] performed a similar study where they compared the performance of decision trees and Bayesian networks when trying to predict academic performance. The study was conducted at two different institutions in Asia and included 20492 and 936 students. The attributes in both studies were slightly different but included factors related to previous academic performance and demographics. Attributes such as previous GPA or entry GPA, English performance, institution rank and home country were the most informative ones. They achieved a success rate of 61% when trying to classify the students in one of four classes and a success rate of 66% and 79% when trying to classify the students in one of two classes when using Bayesian Networks [105]. Decision trees performed slightly better.

Misiunas et al. had collected records from 1024 students including variables related to demographics, high school performance, college performance, financial situation and working status. They used pre-college information to predict college performance and achieved an accuracy of 47% with GPA and 55% with degree completion. Using ongoing college attributes yielded in a 75% accuracy in predicting degree completion and 76% accuracy in predicting high school GPA.

Nokelainen et al. [73] studied the relationship of self-attribution theory and mathematical giftedness. The data included 86 students participating

in Mathematics Olympians, 52 students participating in secondary school national mathematics competition and 74 students taking an advanced mathematics course in a polytechnic that all completed a given Self-Confidence Attitude Attribute Scales (SaaS) questionnaire. Using the 18 variables they were able to reach a classification accuracy of 65% when classifying the students into Olympians, national competition participants, and polytechnics [73].

Bayesian networks have been used in student modeling in general. In addition to predicting success, they have been applied for modeling item-item relationships [35], students' behavior within a tutoring system [75] and future group performance in face-to-face collaborative learning [101].

4 Research design and methodology

The aim of this study is to explore variables that affect performance on introductory programming courses. The motivation behind it is to understand reasons or risk factors behind failing a programming course, and at the same time, to understand reasons why some students perform better than others. On the other hand, one goal is to be able to recognize students who are at risk to fail in order to provide assistance and/or individually designed study material for them. The research question can be divided into two parts:

1. Do factors relating to students background and psychological and cognitive characteristics affect how well students perform in an introductory programming course, and how do these features depend on each other?

2. Is it possible to predict students' performance on an introductory programming course using these factors?

The first question aims to study different factors that affect how students learn to program, and how these factors depend on each other. Most studies to date that have studied traditional, questionnaire-based factors have focused on only one or few of them at a time. Here, we are interested in finding a bigger picture of the domain and not to just study individual predictors, but to also see how the predictors relate to each other. Different psychological and cognitive variables can be expected to be dependent on each other,

and understanding these relationships can help to understand how or why certain variables seem to be related to performance. Modeling the domain as a whole allows us to answer flexibly different kinds of questions, such as predicting how different interventions would affect the performance or how a student would perform on the course. As correlations do not imply causation, studying simple correlations does not yet allow us to make conclusions about for example how improving or changing the learning strategies would change the performance.

The data-driven variables described in Sec. 2.3 measure more the current knowledge and skills of the students rather than characteristics that lead to or relate to these skills, and these variables have been excluded. Data-driven variables have already been studied and found to be good predictors in many contexts [2, 22, 80, 113], though there is still a lot to be improved even with these variables. The effect of psychological and cognitive variables as well as students' background on programming performance have also been studied, but the results have been inconsistent, and only very few studies have included several variables. Thus, there is still perhaps more to achieve in understanding how these factors relate to the learning process and the performance or whether they relate to it at all. Therefore, the focus of this thesis is on psychological and cognitive factors as well as student's background information. These factors can also be used to predict students' success before the course has even started, and the information can be used to for example plan admission criteria, or to plan the course structure according to the participants' needs.

The features included in this study are presented in Sec. 4.2. As most previous studies that have aimed to find factors that affect success on introductory programming courses have calculated correlations between these variables and a selected performance measure, We will also present the correlations measured in this context. The goal, however, is to use Bayesian networks to visualize how these variables depend on each other and on the performance on an introductory course. The methods used for learning the network are explained in Sec. 4.4.

With the second question, we intend to study if it is possible to predict whether a student will fail or pass an introductory programming course.

This is done by using the models constructed in answering the first research question, as well as by experimenting with simpler classifiers. Classification is a common way to evaluate the performance of a model. The two simple classifiers have been added here to see whether the more complicated model can perform any better than these two. The classification methods used in this thesis are described in Sec. 4.5.

4.1 Context

The data for the study comes from three different cohorts of an introductory programming course organized in spring and fall semester in 2015 and in spring semester 2016 at University of Helsinki. The course lasts for seven weeks, and it covers the basics of object oriented programming. The programming language on the course is Java. All course material is available online, and the material for each week includes a comprehensive introduction to the topic or topics of the week, as well as the exercises that are incorporated in the material.

The course focuses more on actual programming than traditional lectures, and thus includes a large amount of practical exercises. A new exercise set is introduced every week, and it typically consists of many small problems that combine into larger programs. The students can work on the exercises either in the computer labs, where they may ask for help from teaching assistants, or they can work on the exercises independently at home. Teaching assistants are available in compute classrooms several hours a week, and they are mostly students that have recently completed the course themselves. A more detailed description of the course organization is found for example in [61, 111].

The course grade consists of three parts: programming exercises (70%), computer exam (15%) and a traditional pen and paper exam (15%). Thus, the course includes a lot of exercises, and each week a new exercise set, that is worth ten points, is published. Unlike the normal paper and pen exam, the computer exam can be done on students' own time wherever they choose at the end of the course. The students can use the internet and all material during this exam, but asking for help is not allowed. The grade is given on a scale pass/fail, where to pass the course the students needed to collect 70%

(spring 2015) or 75% (fall 2015, spring 2016) of the total course points and to achieve at least half of the points in both of the exams. Spring 2016 also introduced the possibility to achieve a grade 5, which is the best possible grade. To reach this grade instead of just passing, you had to obtain at least 90% of the exercise points and 90% of the exam points.

During the course, there are one or two short questionnaires each week that measure for example students' learning strategies, motivation, and self-esteem. Answering these questionnaires is voluntary, but students can earn a small amount of extra course points if they decided to fill the forms. The questionnaires are part of the online material, and are always included in the beginning of the topic of the week. This data has not been collected specifically for this study, and the included questionnaires have not been planned specifically for this purpose.

The participants include both computer science majors and other majors. Especially the courses organized during spring semester have a lot of minor students, but the information of students' majors was not available at this time. There are generally more male students than female students, and the students come from very varying age groups. Table 1 introduces more statistics about the participants and their backgrounds.

4.2 Factors included in the study

The aim of this study is to model factors that affect success in introductory programming courses. The focus is specifically on students attitudes, motivations, learning strategies and background rather than their current skills. Thus, the variables included here are mostly either based on students' background like presented in Sec. 2.1 or psychological and cognitive variables like presented in Sec. 2.2. All background variables are listed in Table 1 and other variables including the abbreviations used in this thesis as well as some descriptive statistics are listed in Table 2. The variables were chosen from existing data collected previously. More detailed descriptions of these factors are given below.

The most important decision to make is the performance measure. The most common measure used in previous studies is the midterm or final grade

Variable and abbreviation	Possible values	# of students
Grade (grade)	pass	213
	fail	87
Gender (gen)	female	94
	male	155
	unknown	51
Age	22 or younger	132
	23 or older	123
	unknown	45
Programming experience (PE)	no experience	195
	experience	105
Working status (WS)	not working	184
	working	69
	unknown	47

Table 1: Course participants' background information for discretized variables

that is as a continuous variable representing a number between 0 and 100 (e.g. [11,113,117,119]). A similar measure in this context is the course points, and that is used as the performance measure when computing correlations. This is a value between 0 and 110, where 100 points are considered to be full points, but as some extra points can be collected by answering the questionnaire or completing additional exercises, some students have collected more than 100 points. As discrete variables are needed for learning a structure for Bayesian network, the grade is used as the performance measure there. However, the course points, and thus also the grade, is heavily affected by the practical exercises, whereas in many other contexts the grade is mostly affected by the exam or exams [11,113]. Therefore, we have chosen to use the points achieved in the pen and paper exam as another performance measure. This was chosen instead of the computer exam to have more comparable results with previous studies. Also, most students who took the computer exam got full points the mean being 14.04/15 and median 15. Most likely due to the large amount of exercises, the students who make it to the final

	Variable	Abb.	Values	Mean	s. d.	# of answers
Performance	Exercise Points	Exer	0–80	54.87	15.11	300
	Exam Points	Exam	0–15	12.47	2.58	260
	Computer Exam Points	CoExam	0–15	14.04	1.82	261
	Course Points	Point	0–110	77.90	23.38	300
MSLQ Learning Strategies	Critical Thinking	CT	[0,1]	0.64	0.09	232
	Effort Regulation	ER	[0,1]	0.56	0.14	232
	Elaboration	Ela	[0,1]	0.65	0.11	232
	Help Seeking	HS	[0,1]	0.61	0.14	232
	Self Regulation	SR	[0,1]	0.53	0.19	232
	Organization	Org	[0,1]	0.62	0.10	232
	Peer Learning	PL	[0,1]	0.51	0.18	232
	Rehearsal	Reh	[0,1]	0.44	0.18	232
	Time and Environment	TE	[0,1]	0.48	0.16	232
MSLQ Motivation	Extrinsic Goal Orientation	EGO	[0,1]	0.51	0.16	217
	Intrinsic Goal Orientation	IGO	[0,1]	0.74	0.14	217
	Self-Efficacy	SE	[0,1]	0.74	0.16	217
	Task Value	TV	[0,1]	0.84	0.13	217
	Test Anxiety	TA	[0,1]	0.46	0.17	217
	Learning Beliefs	LB	[0,1]	0.83	0.13	217
Big Five	Extroversion	Ext	[0,1]	0.60	0.16	227
	Agreeableness	Agr	[0,1]	0.75	0.11	227
	Conscientiousness	Con	[0,1]	0.64	0.13	227
	Emotional Stability	ES	[0,1]	0.71	0.12	227
	Openness	Ope	[0,1]	0.70	0.09	227
Self Beliefs	Debugging Self Efficacy	DSE	[0,1]	0.74	0.15	97
	Prog. Anxiety	PA	[0,1]	0.51	0.20	97
	Prog. Aptitude Mindset	PAM	[0,1]	0.29	0.13	97
	Prog, Interest	PI	[0,1]	0.78	0.14	97
	Prog. Self Concept	SC	[0,1]	0.78	0.14	97
	General Self Efficacy	GSE	[0,1]	0.72	0.14	223
	Self Esteem	RSE	[0,1]	0.50	0.17	217

Table 2: Psychological and cognitive variables and the performance measures included in this study. The table also lists the abbreviations used throughout the figures in this thesis as well as the range of possible values, means, standard deviations and the number of answers for each variable.

weeks also do well on the computer exam. As all students did not participate in the exam, the amount of students included in the analysis in this case is smaller.

As described in Sec. 2, there are several different variables that have been associated with students' performance in introductory programming courses. However, the results with all of these variables have been inconsistent. This is understandable, as with student modeling and educational questionnaires there is always a lot of uncertainty present. The courses in different universities vary, including the context, teaching, exercises, programming languages, etc. Also students' attitudes towards studying and the course can vary depending on for example the culture or the university. Therefore, it is not easy to pick only some variables that clearly relate to performance on programming courses, but the aim is to examine a wide range of different kind of features. The variables were chosen based on previous studies in predicting success on introductory programming courses and the data available in our context. Most variables chosen have been shown to correlate with programming performance by at least one study.

During the first week of the course, the student's are asked to answer a questionnaire about their background. The variables extracted from that questionnaire are presented in Table 1. Especially previous programming experience has been extensively studied as a predictor, and many studies have found correlations between it and success on introductory programming courses [45, 49, 113, 118, 119]. Gender, on the other hand, has not been shown to correlate with programming performance [18, 81, 107, 113, 118], but it can have an effect on other variables used here. Age has been studied less in this context, and where it has been included, the population did not include many older students [11]. In this case, as shown in Table 1, we have students from different age groups. Working status of the students can reflect for example on the time and effort they are able to put on the course.

To asses the learning strategies and motivational orientations of the students, we used the Motivated Strategies for Learning Questionnaire (MSLQ) [83]. The instrument was developed to assess these two aspects of college students and is always related to a certain situation. Thus, the results can vary depending on the class or subject. The questionnaire is divided

into two parts. The learning strategies questionnaire is incorporated in the material for week 1, and the section measures nine different variables [83]:

- *Critical Thinking*: Describes the degree in which the students apply previous knowledge in new situations.
- *Metacognitive Self Regulation*: Describes the use of these strategies that involve techniques like planning, monitoring and regulating.
- *Organization*: Describes the use of organization strategies. They involve techniques like outlining and finding main ideas, and they help in constructing connections and in selecting appropriate information.
- *Elaboration*: Describes the use of elaboration strategies that involve techniques like summarizing, paraphrasing and creating analogies, and thus these strategies help with storing information into long-term memory.
- *Rehearsal*: Describes the students use of rehearsal strategies. The strategies do not generally help in acquisition of new information, and involve techniques like reciting items that need to be learned from a list.
- *Time and Study Environment*: Describes students' ability to manage their time and study environments.
- *Effort Regulation*: Describes how well the students can control their effort and attention with uninteresting tasks or with distractions.
- *Peer learning*: Describes the level of collaboration with peers.
- *Help Seeking*: Describes the ability of the students' to recognize when they need assistance, and to identify where they can get the needed help.

The motivation section questionnaire is incorporated in the material for Week 2, and it measures six different variables [83]:

- *Control Of Learning Beliefs*: Describes whether the student believes that his efforts on the task will lead to a positive result.

- *Extrinsic Goal Orientation*: Describes the student's perceptions of the reasons for performing the task. Students with high extrinsic goal orientation engage in the task for reasons like grades, rewards, performance, and competition.
- *Intrinsic Goal Orientation*: Describes the student's perceptions of the reasons for performing the task. Students with high intrinsic goal orientation engage in the task for reasons like challenge, curiosity, and learning.
- *Self-Efficacy For Learning and Performance*: Describes person's judgement of their ability to perform a task and reach goals. Self-efficacy is always related to a certain domain, and this aims to measure student's self-efficacy for learning and performance.
- *Task Value*: Describes how important, interesting or useful the student sees the task.
- *Test Anxiety*: Describes the level of worry and anxiety test cause for a student.

Though self-efficacy for learning and performance is already included in MSLQ, we have also included the *General Self-Efficacy scale* (GSE) [96] as a variable in this study. GSE measures students beliefs whether they are able to complete tasks and whether their own actions are responsible for the outcome in general, not only related to learning as in MSLQ. The questionnaire was introduced in the material for week 4.

Self-esteem has been linked to success at school by several studies [50], and even though the link with success in programming is less clear, we have decided to also include it in this study. Rosenberg's self-esteem scale (RSE) is a widely used measure for self-esteem, and it has also been studied in the context of programming. The results have been inconsistent with some studies finding moderate correlations [11], but other studies failing to find any significant correlation [113]. The questionnaire was introduced on Week 4 of the course.

To examine the effect of personality on success in introductory programming courses we have included the so called *Big Five* personality dimensions.

The five dimensions are: *extroversion, agreeableness, conscientiousness, emotional stability* and *openness to experience*. It is a widely used and studied theory, and several questionnaires that measure these traits exist. Here, the dimensions were measured during Week 4 using the Mini-Markers questionnaire [94]. Though the Big Five theory has not been directly used in performance prediction for programming courses, their impact on for example pair programming performance has been examined. However, their predictive value was found to be very modest [46].

Scott and Ghinea have also developed an instrument that measures student's self-beliefs specifically in introductory programming courses [97]. Very few instruments to study specifically computing education exists, and the other instruments also used in this study are designed for education research or personality research in general. Scott's and Ghinea's work aims to fill this gap, though the work is relatively recent and has not yet been validated in different contexts. The instrument contains five variables: *debugging self-efficacy, programming self-concept, programming interest, programming anxiety* and *programming aptitude mindset* [97].

4.3 Data preprocessing

Before performing the analyses, the data needs to be preprocessed. From the original data, we need to collect students who have actually participated in the course and also agreed to provide some information about them. Once we have the processed data set, the continuous variables are still discretized for structure learning and classification. This chapter describes how the preprocessing was done.

First, we have removed the observations, that have collected less than ten course points, that means the students who have signed up for the course but completed less than one week's exercises. Many students sign up for the course but for some reason do not show up in the beginning. Also the students who have answered less than two of the questionnaires were excluded from the study. After this, 300 students were left in the data.

Continuous variables were discretized for practical reasons. The chosen methods for learning a Bayesian network here were only feasible with discrete

variables with networks this big. Discretizing variables and treating them as categorical removes the need to make assumptions about the distributions, and can help in finding more complex, non-linear relationships. Discretization has been shown to increase the performance of at least naive Bayes classifiers [37]. However, with discretization, some information is always lost, and especially linear relationships can be harder to find afterward. Thus, discretization method should be considered carefully.

The students on the course are graded on a scale pass / fail, and the achieved grade is directly used as the performance measure. The exam points were categorized into two equal sized bins. Programming experience was categorized into two classes: no experience or experience, which covers all experience from taking a course or studying on your own to several years of work experience. Similarly, workload outside of school was categorized into two classes: no work and a part-time job or full-time employment. The students were also divided into two approximately equal-sized categories based on their age at the time of the course.

The rest, that is MSLQ variables, Rosenberg's self-esteem, GSE, the Big Five variables and programming self-beliefs were discretized by using equal frequency interval binning (EF). That is, the data was divided into four bins of equal frequency. Supervised discretization methods tend to lead in slightly better results [37], but the discretization is then optimized for a certain class label. Here, the main goal is not classification but to find a network that describes the relationships between the variables and could be used to also predict other features than just performance. Thus, we have chosen an unsupervised discretization method, and EF tends to lead to consistent results.

4.4 Learning the network structure and parameters

Several different algorithms for learning a structure of a Bayesian network exist, and the algorithms are based on different principles and metrics. The data at hand here contains missing values, as many students have not answered all questionnaires. Removing rows with missing information would eliminate two-thirds of the data. Thus, ignoring rows that contain missing

values is not an option. We have chosen to compare two approaches: using an algorithm that incorporates the learning of the structure and learning the missing values and to impute the missing data and use a traditional structure learning method. We have chosen to use score-based methods as they tend to produce better results with small data sets or when the independences are weak [122]. In both cases the learning algorithm is similar and based on greedy search where a locally optimal decision is made in every iteration. Thus, the algorithms can get stuck on a local maximum.

Score based algorithms need a scoring function to determine the best network. The algorithms here use the expected Bayesian Information Criterion (BIC) scoring function [41, 95] that can be written as:

$$BIC = log(P(X|\hat{\theta}, G)) - \frac{d(M)}{2} logN \tag{2}$$

Here $(p(X|\hat{\theta}, G))$ is the likelihood of the data given the estimated parameters and the structure of the network, N is the amount of observations in the data X and $d(M)$ is the amount of parameters in the model. As the scoring functions in general, BIC aims to find a balance between maximizing the likelihood but avoiding over-fitting, so the term $-\frac{d(M)}{2} logN$ is added to penalize complex models with many edges.

Imputation means replacing the missing values in the data with some good guess of the real value. Here we have chosen to use the median for that particular feature. After imputation, the network structure is learned using a *K2* algorithm [29]. The algorithm does not require an initial network, but it requires an ordering of the nodes. It reduces the search space of different DAGs by considering only networks where a node can be considered to be the parent of lower-ordered nodes. The algorithm starts with an empty network, and in each iteration it tests parent insertions and chooses the one that leads to the best total score of the network. The algorithms stops when adding single parent can no longer increase the score. The parents can be added independently as the order guarantees that we can not have any cycles.

The EM-algorithm is a general technique for finding the maximum-likelihood estimate for the parameters from data in the presence of missing values. Hence, it can also be used to learn Bayesian networks, and it was first adapted to learn the parameters of a Bayesian network with a known

structure [65]. It is an iterative algorithm that alternates between two steps, the E-step and the M-step, until the parameter values converge. First, the parameters are initialized by ignoring the missing values. Then, in the E-step, the expected values of the unobserved variables are calculated by using the current parameter values. In M-step, the parameter values are recalculated to maximize the probability of the data. These two steps are then repeated until the algorithm converges. Like this, the EM can be used to learn the parameters when we already know the structure.

Structure learning with missing data is a harder problem than just parameter learning. Friedman's Structural EM (SEM) algorithm incorporates the structure search step inside the EM algorithm [40]. In practice, the structure learning part can be performed using different procedures. The algorithm chosen here, the SEM algorithm included in the structure learning package (SLP) package [67], combines the EM principle with a greedy search (GS) like algorithm.

Similarly to K2, the algorithm here is greedy, and in each iteration, it finds the neighboring network that maximizes the score. It starts with an initial network structure and estimates the probability distribution of the variables. Then, it defines all the neighboring structures, i.e the structures that differ from the current structure by one insertion, deletion or reversion of an arc and calculates the expected score for each DAG. The DAG that maximizes the expectation of the score is chosen for the next iteration. This is repeated until the score cannot be significantly improved anymore. The algorithm is also known as *hill climbing*.

With structure learning algorithms, another important thing to consider is the initialization. Structural EM algorithm requires an initial network and parameters, the prior, that it starts editing to find an optimal network. The algorithm can be started with an empty network that does not contain any edges. Alternatively, it can be initialized with a network designed by an expert, or with a network that is constructed randomly or by a different, simpler algorithm. It is difficult to determine one best strategy for initialization [15]. A popular strategy is to initialize the algorithm is with a random structure, but it might not lead to an optimal solution. SEM can get stuck on a local maximum, and the final network can highly depend

on the initial structure. Thus, we used 100 short runs of the algorithm starting with a random structure and finding the initial structure that leads to highest log-likelihood on two iterations. That structure was then used to initialize the algorithm that was allowed to run for 100 iterations. This strategy was similar to the one that was presented by Biernackia et al. [15]. The parameters were created randomly using a Dirichlet prior.

Similarly, K2 requires an ordering of the nodes as an input. Again, this can be determined for example based on previous knowledge, or it can be chosen randomly. As the ordering affect the final results, we again used 100 random starts and chose the network with the highest score.

For the implementations, we used Matlab and the Bayes Net Toolbox (BNT) [71] as well as the Structure Learning Package (SLP) [67] developed on top of it. SLP includes an algorithm `learn_struct_EM`, that takes an initial Bayesian network and the data as input, and calculates an output network. Similarly, `learn_struct_k2` learns a locally optimal structure from the data given the initial ordering.

4.5 Inference algorithm and classification

Classification performance is one of the most popular evaluation methods for Bayesian networks. Thus, the learned structures are also evaluated based on how well they can classify students as passing or failing or high or low performing half of the class. To answer the second research question, that is whether it is possible to predict the performance of students' with the variables included in this study, we have also included two simpler classifiers: Naive Bayes (NB) classifier and tree augmented Naive Bayes (TAN) classifier.

Naïve Bayes is simple but well known and widely used classifier. The structure of the network contains an arc from the class node to each of the other variable nodes. This simplifies the joint distribution to:

$$P(C, X_1, ..., X_n) = P(C)P(X_1|C)...P(X_n|C) \tag{3}$$

where C is the class variable (the variable to be predicted) and $X_1, ...X_n$ are the predictor variables. Thus, NB assumes conditional independence between all variables given the class node. Even though this independence assumption

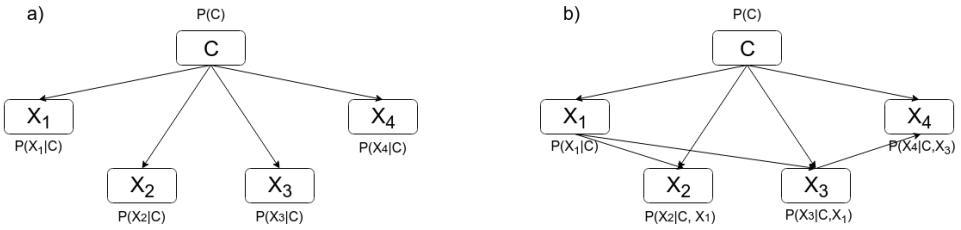

Figure 2: Example of a) Naïve Bayes (NB) classifier and b) Tree Augmented Naïve Bayes (TAN) classifier.

is often unrealistic and actual probability estimations are mostly not accurate, NB classifiers can still usually achieve good classification results [36]. An example of a Naïve Bayes classifier can be seen in Figure 2a.

TAN relaxes the independence condition of NB by allowing arcs between the attributes so that each predictor can have at most one other predictor as a parent. An example of a TAN classifier can also be seen in Figure 2b.

The classification is also performed using the BNT package [71] in Matlab, and the inference is performed using the *jtree* engine, i.e. the junction tree algorithm. Junction tree algorithm refers to an exact inference algorithm that is based on transforming the network into a tree structured network, a junction tree, where a message-passing procedure is carried out. Probabilistic reasoning in multi-connected networks is NP-hard [28], but it is possible to transform a multi-connected network to a singly connected network, where reasoning can be performed in linear time. However, finding an optimal junction tree is also an NP-hard problem [120], and the newly formed network can have much more parameters than the original one. Thus, even though efficient linear time algorithms can be used, in the worst case the amount of parameters can be exponentially higher than in the original network. Still, in practice, junction tree algorithms often perform well and they are widely used.

Junction tree algorithm transforms the network with three steps:

1. *Moralization:* Nodes with a common child are connected, and all edges are made undirected.

2. *Triangulation:* Undirected edges are added until every induced cycle in the graph have at most three vertices.

3. Constructing a *junction tree*: The triangulated graph can be transformed into a junction tree by connecting nodes in the maximal cliques as one variable.

The junction tree is equivalent to the original network. This means that the tree describes the same probability distribution as the original multi-connected network. A message passing algorithm [54, 64] can then be applied in the transformed network. A detailed description of junction tree algorithms can be found for example in [53, 78].

The classification was performed by dividing the data set into two parts: 70% for training and 30% for testing. The network structure and parameters were learned using the training data, and the performance was evaluated by computing the classification accuracy on the test set. All values for all performance measures were removed from the test set in all cases. The simpler classifiers, Naïve Bayes and Tree augmented Naïve Bayes were also evaluated in the same way to be able to compare the results. EM principle was used to learn the parameters.

In addition, NB was used to study whether these factors can be used to predict performance within one course. This was estimated using 10-fold cross-validation. This means, that the data was randomly divided into ten equal sized sets. Then, the network parameters were learned using nine of the sets while one set was used for testing. This was repeated so that each of the ten sets was used for testing once.

The performance of these classifiers is evaluated with two different measures. The classified test cases can be divided into one of the four categories: true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). The measures below can be calculated using these values. Classification accuracy is a widely used measure, and it has been reported in most of the previous studies presented in this thesis. However, the value depends on the prevalence of the classes and can be misleading with imbalanced data sets. It is calculated as follows:

$$ACC = \frac{TP + TN}{P + N} \tag{4}$$

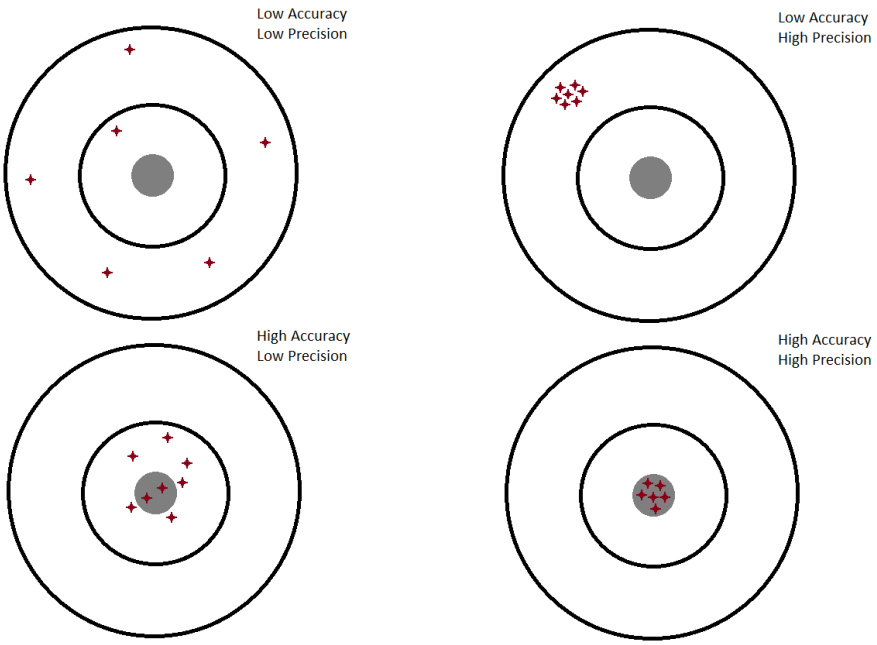

Figure 3: An illustration of accuracy and precision.

As accuracy itself is not enough to describe how well the classifier performs, F_1 score is also calculated. It is the harmonic mean of *precision* and *recall*. Figure 3 describes the difference between accuracy and precision.

$$precision = \frac{TP}{TP + FP} \tag{5}$$

$$recall = \frac{TP}{TP + FN} \tag{6}$$

$$F_1 = \frac{2TP}{2TP + FP + FN} = 2 \times \frac{precision \times recall}{precision + recall} \tag{7}$$

5 Experiments and results

In this section, we present the results obtained for the two research questions defined in Sec. 4. First, Sec. 5.1 describes the replication of some previous studies in our context. For this, we have calculated Pearson correlation coefficients between factors included in this study and two different performance

measures. We also compare the results with previous work. Section 5.2 presents the network structures found for these factors. Section 5.3 presents the classification results achieved with the constructed networks and two simple Bayesian classifiers.

5.1 Replication of previous studies

Many previous studies exploring factors that affect performance on introductory programming courses have focused on studying the effects of individual factors on performance separately. The results have been inconsistent and varied depending on the context. Moreover, only a small number of studies have been performed, and further verification of the results is needed. Thus, we have also studied the factors individually. The performance measures used in these tests are the course points and the exam points.

In addition to studying the whole, combined dataset, we have considered all the three course instances separately. Though the materials and exercises on all of these courses were mostly the same, there are always small differences. For example, the teaching assistants are mostly different each year, and the participants vary. The courses during fall periods are usually bigger. For example, here there were 151 students in the fall 2015 course and only 78 and 76 students int the spring courses. Moreover, the courses organized in fall include a lot of major students, whereas the spring courses are mostly for minor students.

T-tests and Wilcoxon rank sum tests for independent samples based on the background variables were performed to study the differences in the performance of each group. Performance, in this case, was measured only with combined course points. T-tests have been the most popular test in previous studies, and thus we chose also to use it. However, the assumption of normality was violated here, and even though the sample size is quite big, we also performed a Wilcoxon rank sum test. The descriptive statistics of performance for the combined data set for each group are presented in Table 3. The tests found no significant differences in performance between any of the groups at significance level 0.01, but Wilcoxon rank sum test found a significant difference for the samples grouped by programming experience

		n	Mean	S.D	Median
Gender	female	94	75.3	26.3	84.8
	male	155	79.6	22.1	86.8
Age	22 or younger	132	79.7	22.6	87.4
	23 or older	123	76.7	24.7	84.8
Programming experience	no experience	195	77.5	21.9	83.3
	experience	105	78.7	25.9	88.5
Working	not working	184	78.0	24.0	86.4
	working	69	78.5	23.0	85.7

Table 3: Comparison of the mean, standard deviation and median for course points grouped by different variables.

at significance level 0.05. No significant differences at level 0.01 were found on the individual courses either, but in fall 2016 both t-test and Wilcoxon test found a significant difference in performance at level 0.05 when the students were grouped by gender or by programming experience.

The results are mostly in line with previous studies. Most previous studies have found no statistically significant differences between samples grouped by gender [11, 18, 81, 107, 118], programming experience [11, 107], age [11] or working status [11]. However, also some other studies have found significant differences between the samples that were grouped by previous programming experience [18, 45, 113, 116].

We have also calculated the Pearson correlations between the continuous variables chosen in this study and two selected performance measures: course points and exam points. The results are shown for the combined data set as well as separately for each course instance. As the correlations with both the course points and exam points were similar, we only present the correlations with course points for individual courses to make the presentation clearer. These results can be seen in Table 4. In addition, we have collected results achieved in some previous studies, and these are presented in the same table.

We can see that the correlations in our context are smaller than in many of the previous studies. Self-efficacy for learning and performance has been

		Spring 15 r/course	Fall 15 r/course	Spring 16 r/course	All students r/exam	r/course	Previous research
MSLQ LS	CT	0.26*	0.23***	0.17	0.15**	0.23***	0.28* [113], 0.58*** [13]
	ER	0.02	0.14	0.18	0.11*	0.13**	0.28* [113], 0.62*** [13]
	Ela	0.05	0.23***	0.16	0.09	0.17***	no [13]
	HS	0.01	0.05	0.15	0.01	0.07	no [13]
	SR	0.23*	0.08	-0.05	-0.05	0.05	0.46*** [13]
	Org	0.19	0.12	0.08	0.16**	0.15**	no [13]
	PL	0.10	-0.07	-0.03	-0.04	0.00	-0.06 [113], 0.37** [13]
	Reh	0.22	0.08	-0.07	-0.09	0.05	no [13]
	T&E	0.06	0.05	-0.05	-0.03	0.05	0.37** [13]
MSLQ Motivation	EGO	-0.14	0.06	0.09	0.05	0.00	no [12]
	IGO	0.02	0.19**	0.07	0.25***	0.12*	0.33* [113], 0.51*** [12]
	SE	-0.04	0.34***	0.20	0.29	0.20***	0.54*** [113], 0.57*** [12]
	TV	0.00	0.00	0.22	0.04	0.08	0.06 [113], 0.54 [13], 0.44*** [12]
	TA	0.08	-0.23**	0.07	0.02	-0.06	no [12]
	LB	-0.08	0.16*	0.13	0.12*	0.08	0.30** [12]
Big Five	Ext	-0.03	-0.21*	-0.05	-0.19***	-0.12*	
	Agr	-0.21*	-0.13	-0.01	-0.16**	-0.11*	
	Con	0.29**	0.11	0.14	0.03	0.19***	
	ES	-0.19	-0.07	-0.05	-0.07	-0.10	
	Ope	0.01	-0.04	-0.02	0.03	-0.02	
Self Beliefs	DSE	0.21	0.36*	0.19	0.38***	0.31***	
	PA	0.06	-0.27	-0.15	-0.12	-0.04	
	PAM	0.06	-0.21	0.22	-0.19*	0.02	
	PI	0.29*	0.21	0.18	0.27***	0.30***	
	PSC	0.28*	0.42**	0.09	0.38***	0.31**	
	GSE	0.17	0.00	-0.11	0.11	0.06	
	RSE	-0.05	0.13	0.10	0.06	0.04	0.13 [113],0.36*** [11]

Table 4: Pearson Correlation coefficients (r) measured for two different performance measures: course points and exam points (* $p < .10$, ** $p < .05$, *** $p < .01$). The term "no" means that no statistically significant correlation was found, but the exact numbers were not reported.

shown to correlate strongly with the performance by previous studies [12,113], but in this context, the correlation was significant only with course points but not with exam points, and smaller. No correlations were found for the courses organized in spring, but a moderate, significant correlation was found for the course organized in fall.

Similarly, intrinsic goal orientation has been found to correlate with performance [12,113], and again we found significant but smaller correlations, and only in the combined and Fall 2015 data sets. Also, the results with critical thinking [13,113] and intrinsic goal orientation [12,113] were consistent with previous studies; again the correlations were smaller and not present or as significant in the spring datasets [13, 113].

In general, in the data sets collected on the courses organized in spring the factors do not seem to have significant correlations with performance. More statistically significant correlations can be found for the course organized in fall. This can be due to the larger dataset, or the fact that computer science major students mostly take the course in autumn whereas the spring courses include a lot of minor students.

Table 4 also includes variables that have not been studied in this context. These are the General Self-efficacy scale, Big Five personality features and programming self-beliefs. As self-efficacy is related to specific tasks, it is perhaps not surprising that we did not find significant correlations between GSE and the performance measures. Two of the Big Five measures, extroversion and agreeableness had significant, small, negative correlations with the exam score, and conscientiousness had a significant correlation with only the course points.

Many of the programming self-beliefs variables had moderate, significant correlations with both performance measures. Especially debugging self-efficacy, programming interest and programming self concept correlated with performance. The results for individual courses were mostly not significant, but this can be due to small amount of answers for this questionnaire. The questionnaire had fewer answers than any of the other questionnaires, and both t-test and Wilcoxon rank sum test actually revealed a significant difference in the performance of the ones that answered the questionnaire (mean 72.8, median 81.5) and the ones that did not (mean 88.6, median

92.3). Thus, a data set with more answers would be needed for more reliable results.

5.2 Modeling relationships between factors that affect performance

One of the goals of this thesis was to model the relationships between different background factors, psychological and cognitive factors and programming performance. This was done by learning a Bayesian network from the data. Two different algorithms were used for this purpose to compare the achieved results. As there is a lot of missing values in the data, both algorithms needed to be able to handle the problem. One of the algorithms is based on the EM principle and hill climbing (referred to as SEM). In the other case, we used imputation and replaced all missing values with the median value of that feature, and the network was learned using a K2 algorithm. Descriptions of these algorithms can be found in Sec. 4.4.

Both of the algorithms returned one structure with the highest score. These structures are presented in Figures 4 and 5. The colors in the pictures refer to the group that particular factor belongs to, and the explanations for the abbreviations in the nodes can be found in Table 2. Also, the BIC scores of the networks can be seen in both of the figures.

These two networks are fairly different, and thus interpreting much about the actual relationships in the domain is hard. Differences can be due to the different way of handling missing values. It also seems that in both cases there are several networks with similar scores but different structures. Thus, choosing only one network does not perhaps make sense in this case. Also other models with high probability should be considered in reasoning in this case, but this was not possible when using these algorithms and the BNT package.

Neither of the networks is fully connected. Especially in Figure 5 we can see that the nodes are mostly connected based on the group of the factor. That is, for example, all MSLQ Learning Strategies factors and programming self-beliefs factors are connected, but separate from all other factors. Only MSLQ Motivation and background factors are connected with

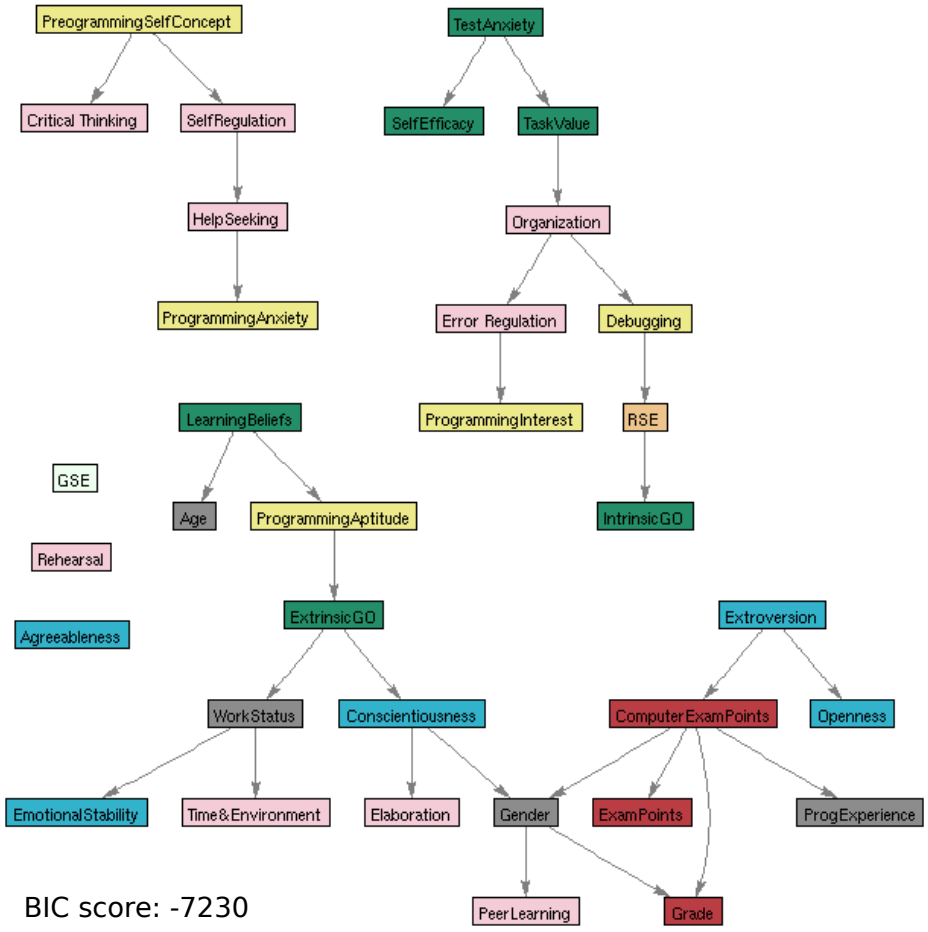

Figure 4: Network structure learned from data using Structural EM algorithm. The colors of the nodes express the group of the variable. Explanations for groups and abbreviations can be found in Table 2

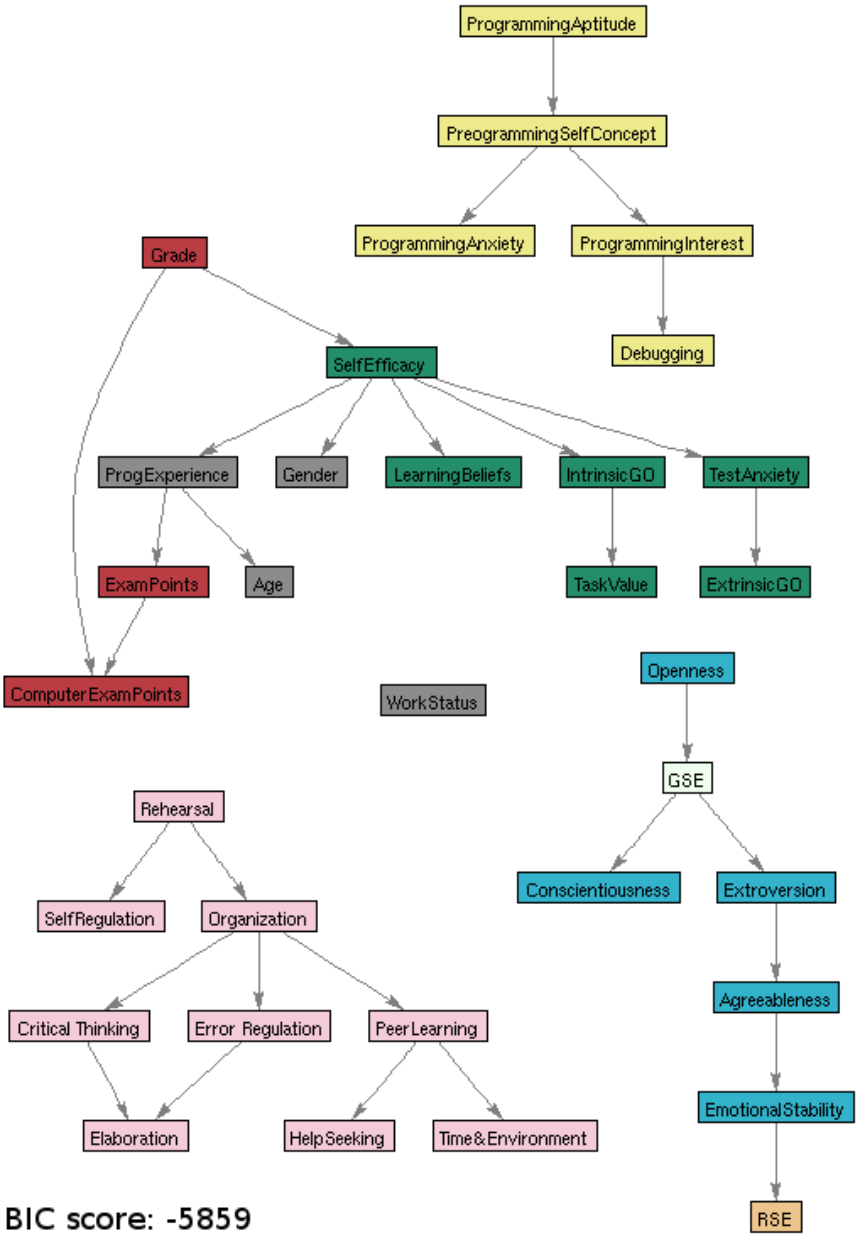

Figure 5: Network structure learned from data using imputation and K2 algorithm. The colors of the nodes express the group of the variable. Explanations for groups and abbreviations can be found in Table 2.

the performance measures. In Figure 4 the division is not that clear, but more factors from different groups are connected with the performance measures. The two networks presented here do not have any common edges.

5.3 Predicting performance on the course

Classification performance is one of the most commonly used criteria to assess the quality of a network. The structures presented in Figure 4 were also evaluated based on whether they can be used to predict if the students will pass or fail the course or whether the student will be in the low or high performing part of the class when it comes to exam points. The data was divided into two parts: 70% for learning the structure and parameters of the model and 30% for testing.

The results are presented in the confusion matrices in Figure 6. As we can see, the classifiers tended to over-estimate the amount of students who passed the course. This is a common problem with imbalanced data sets. There are more students in the data that passed than students who failed. For example, Hien and Haddaway [48] had a similar problem when trying to predict the cumulative grade point average of international applicants. As most students accepted to the school performed well, they had very little data on low-performing students and the classifier overestimated the performance of students with lower grades. In this case, almost all students are classified as passing, which is the majority class. Thus, neither of these classifiers is good at predicting the course outcome of the students.

The classifier here also over-estimated the amount of students who would be in the better performing half according to exam points, even though in this case the dataset is balanced. The performance of these classifiers was not good in this case either, but the performance was near random.

In addition, two simpler classifiers, Naïve Bayes (NB) and Tree Augmented Naïve Bayes (TAN), were used to compare whether the learned structures perform better than a simple structure. These classifiers are much faster than learning a more complicated structure and it's parameters, but generally they still produce good results [36]. The same dataset and division as for evaluating the networks presented in Figures 4 and 5 was used to test

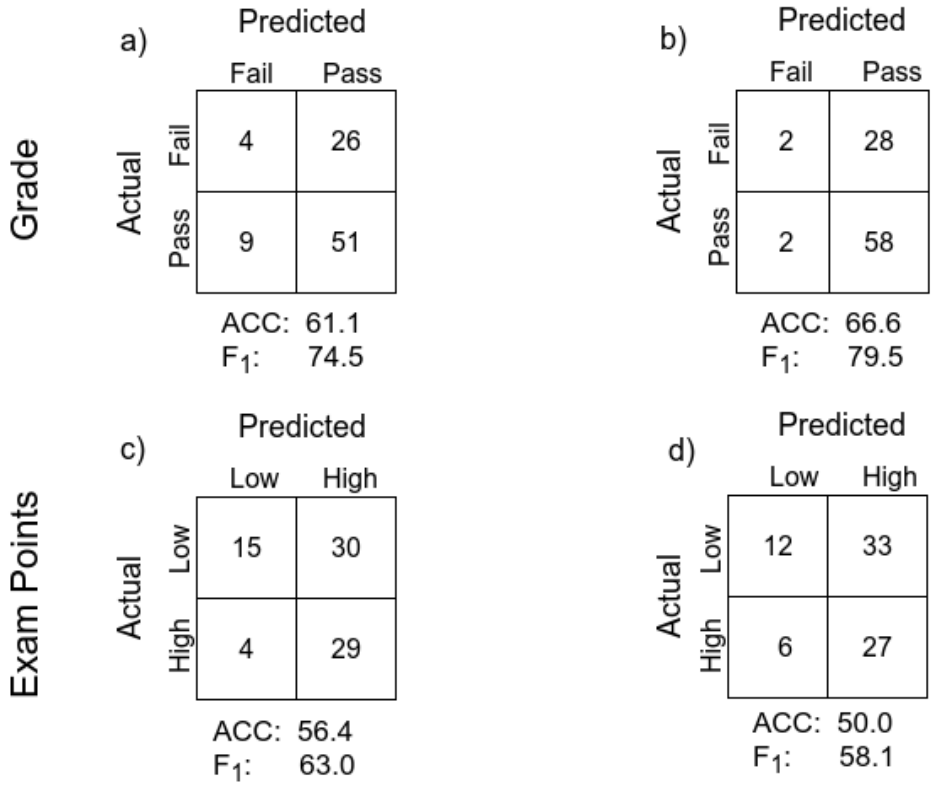

Figure 6: Confusion matrices visualizing classification performance of the learned structures. a) Predicting whether a student will pass or fail using EM. b) Predicting whether a student will pass or fail by replacing missing values with the median. c) Predicting whether a student will be in the high or low performing half in exam points using EM. b) Predicting whether students will be in the high or low performing half in exam points by replacing missing values with the median.

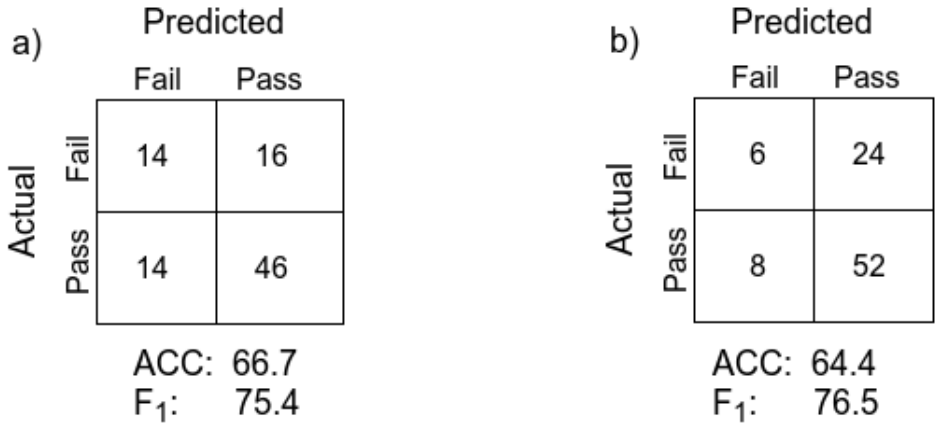

Figure 7: Confusion matrices visualizing classification performance when predicting whether student will pass or fail using a) Naïve Bayes classifier b) Tree augmented Naïve Bayes classifier.

the classification accuracy of NB and TAN classifiers. As the results with predicting grade as well as performance on the exam were very similar, for simplicity we are only presenting the results in predicting the grade. These results are presented in Figure 7. The performance of these classifiers was similar to the learned structures.

In addition, Naïve Bayes classifier was used to see whether the classification accuracy is better within one course than on the whole dataset. Though materials on these courses were mostly the same, there usually still is many differences in the practicalities as well as in the participants of different course instances. With data-driven variables, the classification performance within one course seems to be better than the classification rates over different courses [2]. This study was performed using 10-fold cross-validation, and the results are presented in table 5. The classification rate on the whole dataset and within one course does not seem to vary much, and we were not able to achieve better prediction accuracies within one course.

	Spring 15	Fall 15	Spring 16	Combined
ACC	65.8	62.3	64.9	64.0
F_1	76.8	72.4	77.7	75.2

Table 5: Classification accuracies and F_1 measures of NB classifier when predicting whether the student will pass or fail the course. Data sets are from three different course instances separately, and one data set combines all three instances.

6 Discussion

None of the factors included in this study seem to be good predictors of programming performance in our context. The correlations were in general smaller than in many of the previous studies, and the prediction rates were poor. One reason for this can be the way the course is organized. The course is very practical having a lot of exercises and the grade is also heavily dependent on the number of completed exercises. The exercises get harder gradually, and they are designed to build students' confidence slowly. Also, help is likely more easily available than in many traditional courses, as the instructors are in the labs to provide guidance several hours a week. A Large amount of exercises can also lead to good results, and most students who pass get full or close to full points in the exams. In addition, the grading system on a scale of pass/fail is different from most other contexts.

The most surprising result was perhaps the results concerning self-efficacy for learning and performance. The results with most other factors have been inconsistent depending on the context, but self-efficacy has been shown to correlate with performance in many studies. However, in this context the correlation with grade was small, and correlation with exam performance was not significant. Previous research has shown that self-efficacy can change during the course and give different predictions depending on at which point it is measured, post self-efficacy being a better predictor of performance than pre self-efficacy [117]. In this case, the MSLQ motivation questionnaire was introduced during week 2, but the students can answer the questionnaires at any time during the course. Self-efficacy is also always related to a specific task, and thus it is perhaps not surprising that general self-efficacy does not

seem to be related to programming performance.

Programming self-beliefs instrument was only recently introduced by Scott and Ghinea in 2014. Thus, it has only been tested in their context before. Out of the instruments used in this study, the programming self-beliefs had the strongest correlations with performance. However, this questionnaire had fewer answers than any of the other ones, and the large amount of missing answers can affect the results. In fact, there was a statistically significant difference in results between the students that answered the questionnaire and the ones that did not. One reason for so many missing values can be that the questionnaire was the last questionnaire introduced on the course. Programming self-beliefs can potentially be good predictors of performance, but more answers for the questionnaire are required to determine this.

One limitation related to computing multiple correlation coefficients is the multiple comparisons problem. When performing several statistical tests, in some fraction of these tests the null hypothesis might be rejected just by chance. It is possible that some of the found correlations here are also false positives. The problem could be tackled with for example using the Bonferroni correction, but as none of the previous studies had taken this into account, we decided to use the same significance levels as those studies did.

The network structures constructed with the algorithms were different, and the classification rates were poor. There can be several reasons for this. One clear problem is the missing at random assumption made for the data. As noted earlier, the data in this case is most likely not missing at random. However, the EM algorithm requires the assumption, and thus a bias is introduced in the results. For example programming self-beliefs questionnaire had fewer answers than the other ones, but the reasons for this are not clear.

Having missing data creates additional challenges also. In these cases it is often possible to find only a local maximum. Thus, the learning algorithms used here can also get stuck on a local maximum, and the results are highly dependent on initialization. We tried to solve the problem by using multiple random initializations, but because of increased running times, the number of different initializations was not high. In addition, the structural EM implementation has been previously tested with some known networks [67].

The algorithm was not able to find the structure that is generally considered as correct even with larger datasets, and with a dataset of size 500, only couple of dependencies were found. For the data sets of similar sizes as the one here, the classification rates were 52–68% [67], but the results got better when the size of the dataset was increased. If the dependencies in the data are weak, 300 students might not be enough to find them, especially when the data contains as many missing values as it does now.

The data was also imbalanced so that more students passed (213) than failed (87) the course, and when using the learned structures for classification most students were classified as passing students. Lack of data from students who perform poorly has caused problems in classifying also in other cases [48, 69]. In their study with high-school performance, Marquez-Vera et al. were able to get better true negative rates with some algorithms when using cost-sensitive classification [69], though in most cases there were no clear improvements.

Other studies have been successful in predicting student performance using Bayesian networks [2, 8], but these studies have always included either data-driven variables or variables based on previous academic performance. Other differences can also be found.

Ahadi et al. achieved better classification accuracies when predicting programming performance also with Bayesian networks in their study [1]. However, the factors used in that study were mostly different including data-driven variables and some variables based on previous academic performance. Thus, Bayesian networks can be used for predicting success on a programming course with different factors than the ones included here.

Bekele and Menzel [8] were able to classify students performance as below satisfactory, satisfactory or above satisfactory correctly in 64% of the cases on a mathematics course also using Bayesian networks. These results are better than the ones achieved here as the students were classified into one of three categories. However, there are several differences. They had a bigger sample size with no missing values. The questionnaire included lie detector questions that permitted excluding students who did not answer seriously to the questions. In addition, the analysis included variables on previous English and math performance, which were found to be the most relevant

predictors [8]. It is probably no surprise that previous math performance is a good predictor of performance on a mathematics course, and these previous academic performance measures have been shown to be good predictors also in programming [2, 113, 118].

Lack of variables related to academic performance can thus be one reason for poor classification results. compared to other studies We also have no way to verify whether the students have answered the questionnaires seriously. It is possible that some students have for example just randomly responded something to collect extra points quickly. Thus, a verification method like Bekele and Menzel [8] used could improve the results. They discarded about 10% of the answers because of answers to these lie detector questions.

On the other hand, Sharabiani et al. [99] achieved good classification accuracies on physics and mathematics courses, but not on a computer science course. This could imply that the same methods and predictors that work on other subjects do not apply on CS courses and that performance prediction on CS courses is perhaps a more demanding problem.

However, even though there are several limitations in this study, it is possible that these factors just are not related to programming performance at least in this context. The small or nonexistent correlations support this, as do the poor classification rates. Similar methods have successfully been used to predict student performance using different factors. Inconsistent results in previous literature can also be a result of missing strong relationships. This result is not a total surprise, and actually, it can be seen as a positive result. Perhaps not being able to predict performance based on questionnaires and background information is a good thing, and means that at least in well-organized courses everybody has an equal chance to succeed.

7 Conclusions and suggestions for future work

In this study we have examined factors that can affect performance in introductory programming courses. The focus has been on factors related to students' background and psychological and cognitive characteristics.

We have performed a replication of many previous studies that have examined the effect of individual variables on performance. This was done

by performing t-tests and computing Pearson correlation coefficients. We have also used two different methods to learn a Bayesian network from data collected from three instances of an introductory programming course. The constructed networks were evaluated by measuring the classification rate on performance. Moreover, the predictive power of these variables was examined with two simple classifiers: Naïve Bayes and Tree Augmented Naïve Bayes.

The variables chosen for this study do not seem to be good predictors of performance in our context. Some small or moderate correlations were found, but the models were able to achieve classification rates were poor or near random.

If the goal is just to predict performance, we recommend using data-driven variables. It has already been demonstrated that these factors can be quite good predictors [2, 113], and as the research on these is relatively new, the results will most likely still get better. In addition, using data-driven variables as predictors is also more convenient as they do not require any extra effort from the students, but the data is naturally collected while they solve exercises. Data-driven variables can also reflect learning and the changes in students' skills. If data-driven variables cannot be used, factors based on previous academic performance could then be the best choice.

However, with planning the teaching or choosing what materials to show to each student, information about personality traits and learning strategies could be helpful. Even though it seems that these traits do not strongly correlate with success or are not good predictors of performance, understanding how these affect the learning process can still bring valuable information. Further studies using different methods or perhaps larger data sets are still needed to verify whether the relationships are just weak or nonexistent, or whether the poor results here were due to limitations of the methodology. Also, studies in different contexts are needed before any conclusions can be generalized.

There are several improvements and different approaches that could be used. The data set used could be bigger, and some lie detector questions could be incorporated in the questionnaires. A different set of cognitive and psychological variables could also be used. Promising results have been achieved with achievement goals [124], attributions of success [107, 113, 119]

and comfort level [118]. More sophisticated imputation method could also be used, and experimenting with different classification algorithms can be beneficial. For example, decision trees or random forests have been found to perform better than Bayesian networks when predicting performance [2, 105].

In another context, some factors that characterize students who are at risk of failing were identified using decision trees, but they did not recommend the method for prediction [93]. Thus, perhaps the variables used here are not suitable for prediction, but can rather be used to recognize some "danger zones" [93]. One way to approach the problem could be combining these factors with the programming data to see if certain programming strategies relate to certain psychological or cognitive factors. Even though these strategies or factors might not directly relate to performance, finding such connections can bring valuable information about the learning processes and for example help to provide individually targeted material.

References

[1] Ahadi, Alireza and Lister, Raymond: *Geek genes, prior knowledge, stumbling points and learning edge momentum.* In *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, page 123, New York, New York, USA, 2013. ACM Press.

[2] Ahadi, Alireza, Raymond, Lister, Haapala, Heikki, and Vihavainen, Arto: *Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance.* In *ICER '15 Proceedings of the eleventh annual International Conference on International*, pages 121–130, 2015.

[3] Akaike, Hirotogu: *Information theory and an extension of the maximum likelihood principle.* In *Breakthroughs in statistics*, pages 610–624. Springer, 1992.

[4] Andersen, Stig K, Olesen, Kristian G, Jensen, Finn Verner, and Jensen, Frank: *Hugin-a shell for building bayesian belief universes for expert systems.* pages 1080–1085, 1989.

[5] Bandura, Albert: *Social foundations of thought and action: A social cognitive theory.* Prentice-Hall, Inc, 1986.

[6] Barber, David: *Probabilistic modelling and reasoning: The junction tree algorithm.* Course notes, 2004, 2003.

[7] Beaubouef, Theresa and Mason, John: *Why the high attrition rate for computer science students.* ACM SIGCSE Bulletin, 37(2):103, jun 2005.

[8] Bekele, Rahel and Menzel, Wolfgang: *A Bayesian approach to predict performance of a student (BAPPS): A Case with Ethiopian Students.* algorithms, 22(23):24, 2005.

[9] Bell, Tim: *Establishing a nationwide CS curriculum in New Zealand high schools.* Communications of the ACM, 57(2):28–30, feb 2014.

[10] Bennedsen, Jens and Caspersen, Michael E: *Failure rates in introductory programming.* SIGCSE Bull., 39(2):32–36, 2007.

[11] Bergin, Susan and Reilly, Ronan: *Programming: factors that influence success.* SIGCSE Bull., 37(1):411–415, 2005.

[12] Bergin, Susan and Reilly, Ronan: *The influence of motivation and comfort-level on learning to program.* In *Proceedings of the PPIG*, volume 17, pages 293–304, 2005.

[13] Bergin, Susan, Reilly, Ronan, and Traynor, Desmond: *Examining the role of self-regulated learning on introductory programming performance.* First International Workshop on Computing Education Research, pages 81–86, 2005.

[14] Biamonte, A. J.: *Predicting success in programmer training.* In *Proceedings of the second SIGCPR conference on Computer personnel research - SIGCPR '64*, pages 9–12, New York, New York, USA, jul 1964. ACM Press.

[15] Biernackia, Christophe, Celeuxb, Gilles, and Govaertc, Gérard: *Choosing starting values for the EM algorithm for getting the highest likelihood*

in multivariate Gaussian mixture models. Computational Statistics & Data Analysis, 41(3-4):56–575, 2003.

[16] Borchani, Hanen, Amor, Nahla Ben, and Mellouli, Khaled: *Learning Bayesian Network Equivalence Classes from Incomplete Data.* In Todorovski, Ljupčo, Lavrač, Nada, and Jantke, Klaus P. (editors): *Discovery Science*, volume 4265 of *Lecture Notes in Computer Science*, pages 291–295. Springer, Berlin, Heidelberg, 2006.

[17] Brown, Neil Christopher Charles, Kölling, Michael, McCall, Davin, and Utting, Ian: *Blackbox: A Large Scale Repository of Novice Programmers' Activity.* In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14*, pages 223–228, New York, New York, USA, mar 2014. ACM Press.

[18] Byrne, Pat and Lyons, Gerry: *The effect of student attributes on success in programming.* ACM SIGCSE Bulletin, 33(3):49–52, sep 2001.

[19] Campbell, Vivian and Johnstone, Michael: *The Significance of Learning Style with Respect to Achievement in First Year Programming Students.* In *2010 21st Australian Software Engineering Conference*, pages 165–170. IEEE, 2010.

[20] Campos, Cassio P. de and Ji, Qiang: *Efficient Structure Learning of Bayesian Networks using Constraints.* The Journal of Machine Learning Research, 12:663–689, feb 2011.

[21] Capstick, C. K., Gordon, J. D., and Salvadori, A.: *Predicting performance by university students in introductory computing courses.* ACM SIGCSE Bulletin, 7(3):21–29, sep 1975.

[22] Carter, Adam S., Hundhausen, Christopher D., and Adesope, Olusola: *The Normalized Programming State Model.* In *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15*, pages 141–150, New York, New York, USA, jul 2015. ACM Press.

[23] Chamillard, A T and Karolick, Dolores: *Using learning style data in an introductory computer science course.* In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 291–295, New Orleans, Louisiana, USA, 1999. ACM.

[24] Chickering, David Maxwell: *Learning Bayesian networks is NP-Complete.* In Fisher, D. and Lenz, H. (editors): *Learning from Data: Artificial In- telligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

[25] Chickering, David Maxwell: *Optimal structure identification with greedy search.* Journal of machine learning research, 3(Nov):507–554, 2002.

[26] Chickering, David Maxwell, Heckerman, David, and Meek, Christopher: *Large-Sample Learning of Bayesian Networks is NP-Hard.* The Journal of Machine Learning Research, 5:1287–1330, dec 2004.

[27] Chickering, Do, Geiger, Dan, and Heckerman, David: *Learning bayesian networks: Search methods and experimental results.* In *proceedings of fifth conference on artificial intelligence and statistics*, pages 112–128, 1995.

[28] Cooper, Gregory F.: *The computational complexity of probabilistic inference using bayesian belief networks.* Artificial Intelligence, 42(2-3):393–405, mar 1990.

[29] Cooper, Gregory F and Herskovits, Edward: *A bayesian method for the induction of probabilistic networks from data.* Machine learning, 9(4):309–347, 1992.

[30] Corman, Larry S: *Cognitive style, personality type, and learning ability as factors in predicting the success of the beginning programming student.* SIGCSE Bull., 18(4):80–89, 1986.

[31] Cousins, S B, Chen, W, and Frisse, M E: *A tutorial introduction to stochastic simulation algorithms for belief networks.* Artificial intelligence in medicine, 5(4):315–40, aug 1993.

[32] Cuny, Janice and Aspray, William: *Recruitment and retention of women graduate students in computer science and engineering.* ACM SIGCSE Bulletin, 34(2):168, jun 2002.

[33] Dechter, R.: *Bucket Elimination: A Unifying Framework for Probabilistic Inference.* In *Learning in Graphical Models*, pages 75–104. Springer Netherlands, Dordrecht, 1996.

[34] Dempster, A. P., Laird, N. M., and Rubin, D. B.: *Maximum Likelihood from Incomplete Data via the EM Algorithm on JSTOR.* Journal of the Royal Statistical Society, 39(1):1–38, 1977.

[35] Desmarais, Michel C. and Gagnon, Michel: *Innovative Approaches for Learning and Knowledge Sharing*, volume 4227 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, Berlin, Heidelberg, oct 2006.

[36] Domingos, Pedro and Pazzani, Michael: *On the Optimality of the Simple Bayesian Classifier under Zero-One Loss.* Machine Learning, 29(2/3):103–130, 1997.

[37] Dougherty, James, Kohavi, Ron, and Sahami, Mehran: *Supervised and unsupervised discretization of continuous features.* In *Machine learning: proceedings of the twelfth international conference*, pages 194–202, 1995.

[38] Etminani, Kobra, Naghibzadeh, Mahmoud, and Razavi, Amir Reza: *Globally Optimal Structure Learning of Bayesian Networks from Data.* In Diamantaras, Konstantinos, Duch, Wlodek, and Iliadis, Lazaros S. (editors): *Artificial Neural Networks - ICANN*, volume 6352 of *Lecture Notes in Computer Science*, pages 101–106, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[39] European Schoolnet: *Computing our future – Priorities, school curricula and initiatives across Europe Publisher DTP and design.* Technical report, 2014.

[40] Friedman, Nir: *The Bayesian structural EM algorithm.* In *UAI'98 Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 129–138, 1998.

[41] Friedman, Nir *et al.*: *Learning belief networks in the presence of missing values and hidden variables.* In *ICML*, volume 97, pages 125–133, 1997.

[42] Friedman, Nir and Koller, Daphne: *Being Bayesian About Network Structure. A Bayesian Approach to Structure Discovery in Bayesian Networks.* Machine Learning, 50(1/2):95–125, 2003.

[43] Friedman, Nir, Linial, Michal, Nachman, Iftach, and Pe'er, Dana: *Using bayesian networks to analyze expression data.* Journal of computational biology, 7(3-4):601–620, 2000.

[44] Gregorc, A.: *Gregorc style delineator: Development, technical and administration manual*, 1982.

[45] Hagan, Dianne and Markham, Selby: *Does it help to have some programming experience before beginning a computing degree program?* SIGCSE Bull., 32(3):25–28, 2000.

[46] Hannay, Jo E., Arisholm, Erik, Engvik, Harald, and Sjøberg, Dag I.K.: *Effects of Personality on Pair Programming.* IEEE Transaction on software engineering, 36(1), 2010.

[47] Heckerman, David, Geiger, Dan, and Chickering, David M: *Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.* Machine Learning, 20(3):197–243, 1995.

[48] Hien, Nguyen Thi Ngoc and Haddawy, Peter: *A decision support system for evaluating international student applications.* In *2007 37th annual frontiers in education conference - global engineering: knowledge without borders, opportunities without passports*, pages F2A–1–F2A–6. IEEE, oct 2007.

[49] Holden, Edward and Weeden, Elissa: *The impact of prior experience in an information technology programming course sequence.* In *Proceedings*

of the 4th conference on Information technology curriculum, pages 41–46, Lafayette, Indiana, USA, 2003. ACM.

[50] Holly, W.: *Self-esteem: Does it contribute to student's academic success.* Oregon. School of Study Council: University of Oregon, Eugene, OR, 1987.

[51] Ihantola, Petri, Vihavainen, Arto, Ahadi, Alireza, Butler, Matthew, Börstler, Jürgen, Edwards, Stephen H., Isohanni, Essi, Korhonen, Ari, Petersen, Andrew, Rivers, Kelly, Rubio, Miguel Ángel, Sheard, Judy, Skupas, Bronius, Spacco, Jaime, Szabo, Claudia, and Toll, Daniel: *Educational data mining and learning analytics in programming: Literature review and case studies.* In *Proceedings of the 2015 ITiCSE on Working Group Reports*, ITICSE-WGR '15, pages 41–63, New York, NY, USA, 2015. ACM.

[52] Jadud, Matthew C: *Methods and tools for exploring novice compilation behaviour.* In *Proceedings of the second international workshop on Computing education research*, pages 73–84, Canterbury, United Kingdom, 2006. ACM.

[53] Jensen, Finn V and Jensen, Frank: *Optimal junction trees.* In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 360–366. Morgan Kaufmann Publishers Inc., 1994.

[54] Jensen, Finn Verner, Olesen, Kristian G., and Andersen, Stig Kjaer: *An algebra of bayesian belief universes for knowledge-based systems.* Networks, 20(5):637–659, aug 1990.

[55] Kalyuga, Slava, Ayres, Paul, Chandler, Paul, and Sweller, John: *The expertise reversal effect.* Educational psychologist, 38(1):23–31, 2003.

[56] Koivisto, Mikko and Sood, Kismat: *Exact Bayesian Structure Discovery in Bayesian Networks.* The Journal of Machine Learning Research, 5:549–573, dec 2004.

[57] Kolb, David A: *Management and the learning process.* California Management Review, 18(3):21–31, 1976.

[58] Kontkanen, Petri, Myllymäki, Petri, Silander, Tomi, Tirri, Henry, and Grunwald, Peter: *Comparing predictive inference methods for discrete domains.* In *In Proceedings of the sixth international workshop on artificial intelligence and statistics.* Citeseer, 1997.

[59] Korf, Richard E: *Linear-space best-first search.* Artificial Intelligence, 62(1):41–78, 1993.

[60] Koski, T. J. T. and Noble, J. M.: *A review of Bayesian networks and structure learning.* Mathematica Applicanda, (Vol. 40, No. 1):53–103, 2012.

[61] Kurhila, Jaakko and Vihavainen, Arto: *Management, structures and tools to scale up personal advising in large programming courses.* In *Proceedings of the 2011 conference on Information technology education*, pages 3–8. ACM, 2011.

[62] Lau, Wilfred W. F. and Yuen, Allan H. K.: *Exploring the effects of gender and learning styles on computer programming performance: implications for programming pedagogy.* British Journal of Educational Technology, 40(4):696–712, jul 2009.

[63] Lau, Wilfred W.F. and Yuen, Allan H.K.: *Modelling programming performance: Beyond the influence of learner characteristics.* Computers & Education, 57(1):1202–1213, aug 2011.

[64] Lauritzen, S L and Spiegelhalter, D J: *Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems.* Source Journal of the Royal Statistical Society. Series B (Methodological), 50(2):157–224, 1988.

[65] Lauritzen, Steffen L.: *The EM algorithm for graphical association models with missing data.* Computational Statistics & Data Analysis, 19(2):191–201, feb 1995.

[66] Leray, Philippe and François, Olivier: *Bayesian network structural learning and incomplete data.* In *Proceedings of the International and*

Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR) 2005, pages 33—-40, 2005.

[67] Leray, Philippe and Francois, Olivier: *BNT Structure Learning Package: Documentation and Experiments.* Laboratoire PSI, Universitè et INSA de Rouen, Tech. Rep, 2006.

[68] Madigan, David, York, Jeremy, and Allard, Denis: *Bayesian Graphical Models for Discrete Data Bayesian Graphical Models f or Discrete Data 2Battelle Pacific Northwest Laboratories.* Source: International Statistical Review / Revue Internationale de Statistique International Statistical Institute (ISI) International Statistical Review, 63(2):215–232, 1995.

[69] Márquez-Vera, Carlos, Cano, Alberto, Romero, Cristóbal, and Ventura, Sebastián: *Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data.* Applied intelligence, 38(3):315–330, 2013.

[70] Mayer, David B and Stalnaker, Ashford W: *Selection and evaluation of computer personnel-the research history of sig/cpr.* In *Proceedings of the 1968 23rd ACM national conference*, pages 657–670. ACM, 1968.

[71] Murphy, Kevin P.: *The Bayes Net Toolbox for MATLAB.* 2001.

[72] Neal, Radford M: *Probabilistic Inference Using Markov Chain Monte Carlo Methods.* 1993.

[73] Nokelainen, P., Tirri, K., and Merenti-Valimaki, H. L.: *Investigating the Influence of Attribution Styles on the Development of Mathematical Talent.* Gifted Child Quarterly, 51(1):64–81, jan 2007.

[74] Opper, Manfred. and Saad, David.: *Advanced mean field methods: theory and practice.* MIT Press, 2001.

[75] Pardos, Zachary A., Heffernan, Neil T., Anderson, Brigham, and Heffernan, Cristina L.: *The Effect of Model Granularity on Student Performance Prediction Using Bayesian Networks.* In Conati, Cristina,

McCoy, Kathleen, and Paliouras, Georgios (editors): *Proceedings of the 11th International Conference on User Modeling*, volume 4511 of *Lecture Notes in Computer Science*, pages 435–439, Berlin, Heidelberg, jul 2007. Springer Berlin Heidelberg.

[76] Parviainen, Pekka and Koivisto, Mikko: *Exact structure discovery in Bayesian networks with less space.* pages 436–443. AUAI Press, jun 2009.

[77] Paul Dagum, Eric Horvitz: *A Bayesian Analysis of Simulation Algorithms for Inference in Belief Networks,.* Networks, 23:499—-516, 1993.

[78] Pearl, J: *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[79] Petersen, Andrew, Spacco, Jaime, and Vihavainen, Arto: *An exploration of error quotient in multiple contexts.* In *Proceedings of the 15th Koli Calling Conference on Computing Education Research - Koli Calling '15*, pages 77–86, New York, New York, USA, 2015. ACM Press.

[80] Piech, Chris, Sahami, Mehran, Koller, Daphne, Cooper, Steve, and Blikstein, Paulo: *Modeling how students learn to program.* In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*, page 153, New York, New York, USA, feb 2012. ACM Press.

[81] Pillay, Nelishia and Jugoo, Vikash R.: *An investigation into student characteristics affecting novice programming performance.* ACM SIGCSE Bulletin, 37(4):107, dec 2005.

[82] Pintrich, P. R., Smith, D. A. F., Garcia, T., and Mckeachie, W. J.: *Reliability and Predictive Validity of the Motivated Strategies for Learning Questionnaire (Mslq).* Educational and Psychological Measurement, 53(3):801–813, sep 1993.

[83] Pintrich, Paul R. and Others, And: *A Manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ).* nov 1990.

[84] Pourret, Olivier, Naïm, Patrick, and Marcot, Bruce: *Bayesian Networks: A Practical Guide to Applications.* John Wiley & Sons, 2008.

[85] Ramalingam, Vennila, LaBelle, Deborah, and Wiedenbeck, Susan: *Self-efficacy and mental models in learning to program.* ACM SIGCSE Bulletin, 36(3):171, sep 2004.

[86] Ramalingam, Vennila and Wiedenbeck, Susan: *Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy.* Journal of Educational Computing Research, 19(4):367–81, 1998.

[87] Roberts, Eric S., Kassianidou, Marina, and Irani, Lilly: *Encouraging women in computer science.* ACM SIGCSE Bulletin, 34(2):84, jun 2002.

[88] Rodrigo, Ma. Mercedes T., Tabanao, Emily S., Baker, Ryan S., Jadud, Matthew C., Amarra, Anna Christine M., Dy, Thomas, Espejo-Lahoz, Maria Beatriz V., Lim, Sheryl Ann L., Pascua, Sheila A.M.S., and Sugay, Jessica O.: *Affective and behavioral predictors of novice programmer achievement.* ACM SIGCSE Bulletin, 41(3):156, aug 2009.

[89] Romero, Christobal and Ventura, Sebastian: *Educational data mining: a review of the state of the art.* Trans. Sys. Man Cyber Part C, 40(6):601–618, 2010.

[90] Romero, Cristobal, Espejo, Pedro G, Zafra, Amelia, Romero, Jose Raul, and Ventura, Sebastian: *Web usage mining for predicting final marks of students that use moodle courses.* Computer Applications in Engineering Education, 21(1):135–146, 2013.

[91] Romero, Cristóbal, López, Manuel Ignacio, Luna, Jose María, and Ventura, Sebastián: *Predicting students' final performance from participation in on-line discussion forums.* Computers & Education, 68:458–472, 2013.

[92] Rosenberg, M.: *Society and the adolescent self-image.* Princeton University Press, 1965.

[93] Rountree, Nathan, Rountree, Janet, Robins, Anthony, and Hannah, Robert: *Interacting factors that predict success and failure in a CS1 course.* In *Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 101–104, Leeds, United Kingdom, 2004. ACM.

[94] Saucier, Gerard: *Mini-Markers: A Brief Version of Goldberg's Unipolar Big-Five Markers.* Journal of Personality Assessment, 63(3):506–516, 1994.

[95] Schwarz, Gideon: *Estimating the Dimension of a Model.* The Annals of Statistics, 6(2):461–464, mar 1978.

[96] Schwarzer, Ralf and Jerusalem, Matthias: *Generalized Self-Efficacy scale.* Measures in health psychology: A user's portfolio. Causal and control beliefs, 1995.

[97] Scott, Michael James and Ghinea, Gheorghita: *Measuring Enrichment: The Assembly and Validation of an Instrument to Assess Student Self-Beliefs in CS1.* In *Proceedings of the 10th ACM International Workshop on Computing Education Research*, 2014.

[98] Scutari, Marco: *Learning Bayesian Networks with the bnlearn R Package.* Journal of Statistical Software, 35(3):1–22, 2010.

[99] Sharabiani, Ashkan, Karim, Fazle, Sharabiani, Anooshiravan, Atanasov, Mariya, and Darabi, Houshang: *An enhanced bayesian network model for prediction of students' academic performance in engineering programs.* In *2014 IEEE Global Engineering Education Conference (EDUCON)*, pages 832–837. IEEE, apr 2014.

[100] Spirtes, Peter, Glymour, Clark, and Scheines, Richard: *Causation, Prediction, and Search*, volume 81 of *Lecture Notes in Statistics.* Springer New York, New York, NY, 1993.

[101] Stevens, R., Soller, A., Giordani, A., Gerosa, L., Cooper, M., and Cox, C.: *Developing a Framework for Integrating Prior Problem Solving and Knowledge Sharing Histories of a Group to Predict Future Group*

Performance. In *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–9. IEEE, 2005.

[102] Su, Chengwei, Andrew, Angeline, Karagas, Margaret R, and Borsuk, Mark E: *Using Bayesian networks to discover relations between genes, environment, and disease.* BioData mining, 6(1):6, jan 2013.

[103] Tabanao, Emily S., Rodrigo, Ma. Mercedes T., and Jadud, Matthew C.: *Predicting at-risk novice Java programmers through the analysis of online protocols.* In *Proceedings of the seventh international workshop on Computing education research - ICER '11*, page 85, New York, New York, USA, aug 2011. ACM Press.

[104] Tamada, Yoshinori, Imoto, Seiya, and Miyano, Satoru: *Parallel Algorithm for Learning Optimal Bayesian Network Structure.* The Journal of Machine Learning Research, 12:2437–2459, feb 2011.

[105] Thai-Nghe, Nguyen, Janecek, Paul, and Haddawy, Peter: *A comparative analysis of techniques for predicting academic performance.* In *2007 37th annual frontiers in education conference - global engineering: knowledge without borders, opportunities without passports*, pages T2G–7–T2G–12. IEEE, oct 2007.

[106] Uusitalo, Laura: *Advantages and challenges of Bayesian networks in environmental modelling.* Ecological Modelling, 203(3):312–318, 2007.

[107] Ventura, Philip R.: *Identifying predictors of success for an objects-first CS1.* Computer Science Education, 15(3):223–243, sep 2005.

[108] Verma, Thomas and Pearl, Judea: *Equivalence and synthesis of causal models.* pages 255–270, 1990.

[109] Vihavainen, Arto: *Predicting students' performance in an introductory programming course using data from students' own programming process.* In *2013 IEEE 13th International Conference on Advanced Learning Technologies*, pages 498–499. IEEE, 2013.

[110] Vihavainen, Arto, Luukkainen, Matti, and Ihantola, Petri: *Analysis of source code snapshot granularity levels.* In *Proceedings of the 15th Annual Conference on Information technology education*, pages 21–26, Atlanta, Georgia, USA, 2014. ACM.

[111] Vihavainen, Arto, Paksula, Matti, and Luukkainen, Matti: *Extreme apprenticeship method in teaching programming for beginners.* In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM, 2011.

[112] Watson, C, Li, F W B, and Godwin, J L: *Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior.* In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 319–323, 2013.

[113] Watson, Christopher, Li, Frederick W B, and Godwin, Jamie L: *No tests required: comparing traditional and dynamic predictors of programming success.* In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 469–474, Atlanta, Georgia, USA, 2014. ACM.

[114] Watson, Christopher and Li, Frederick W.B.: *Failure rates in introductory programming revisited.* In *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14*, pages 39–44, New York, New York, USA, 2014. ACM Press.

[115] Werth, Laurie Honour: *Predicting student performance in a beginning computer science class.* In *SIGCSE '86 Proceedings of the seventeenth SIGCSE technical symposium on Computer science education*, pages 138–143, 1986.

[116] Wiedenbeck, Susan: *Factors affecting the success of non-majors in learning to program.* In *Proceedings of the 2005 international workshop on Computing education research - ICER '05*, pages 13–24, New York, New York, USA, oct 2005. ACM Press.

[117] Wiedenbeck, Susan, Labelle, Deborah, and Kain, Vennila N R: *Factors affecting course outcomes in introductory programming.* In *16th Annual*

Workshop of the Psychology of Programming Interest Group, pages 97–109, 2004.

[118] Wilson, Brenda Cantwell: *A Study of Factors Promoting Success in Computer Science Including Gender Differences.* Computer Science Education, 12(1-2):141–164, aug 2010.

[119] Wilson, Brenda Cantwell and Shrock, Sharon: *Contributing to success in an introductory computer science course: A Study of Twelve Factors.* ACM SIGCSE Bulletin, 33(1):184–188, mar 2001.

[120] Yannakakis, Mihalis: *Computing the Minimum Fill-In is NP-Complete.* SIAM Journal on Algebraic Discrete Methods, 2(1):77–79, mar 1981.

[121] Yedidia, Jonathan S, Freeman, William T, and Weiss, Yair: *Generalized Belief Propagation.* Advances in Neural Information Processing Systems 13, 2001.

[122] Yuan, Changhe and Malone, Brandon: *Learning Optimal Bayesian Networks: A Shortest Path Perspective.* Journal of Artificial Intelligence Research, 48:23–65, 2013.

[123] Zhang, Nevin Lianwen and Poole, David: *Exploiting causal independence in Bayesian network inference.* Journal of Artificial Intelligence Research, 5(1):301–328, 1996.

[124] Zingaro, Daniel and Porter, Leo: *Impact of Student Achievement Goals on CS1 Outcomes.* SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pages 279–296, 2016.