KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

On Constraints, Optimisation, Probability and Data Mining

Behrouz Babaki

Supervisors:
Prof. dr. Luc De Raedt
Prof. dr. Tias Guns

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

September 2017

On Constraints, Optimisation, Probability and Data Mining

Behrouz BABAKI

Examination committee:
Prof. dr. ir. Paul Van Houtte, chair
Prof. dr. Luc De Raedt, supervisor
Prof. dr. Tias Guns, supervisor
Prof. dr. Patrick De Causmaecker
Dr. ir. Anton Dries
Prof. dr. Siegfried Nijssen
(Université catholique de Louvain)
Prof. dr. Christel Vrain
(University of Orléans)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

September 2017

Abstract

Constraint satisfaction and optimization (CSP(O)), probabilistic inference, and data mining are three important subdomains of artificial intelligence. CSP(O) investigates methods for efficiently solving combinatorial problems, probabilistic inference deals with answering queries about uncertain knowledge bases, while data mining aims at finding and modeling regularities in the data. Even though these domains have been developed independently, there are strong connections and interactions between them. Studying and extending their interactions is the main theme of this thesis.

This thesis has five contributions. First, we build on existing methods in probabilistic inference and constraint programming and propose a method for adding probabilities to the CSP(O) models. This allows us to constrain or optimize over probability values. Second, we develop a novel algorithm for optimizing the expected utility in stochastic constraint programs. Unlike earlier works that assume independence of random variables, we assume a joint distribution represented by a Bayesian network. Third, we develop an algorithm for obtaining the exact solution of the constrained clustering problem with the maximum sum of squares objective. By using the column-generation framework, we decompose the problem into two components: one that generates candidate clusters that adhere to user constraints, and one that tries to find the optimal solution among combinations of the generated candidates. Fourth, we develop an exact algorithm to solve a special class of graph clustering problems. The formulation of this problem involves an exponential number of constraints. We propose a mechanism to incrementally include only a subset of these constraints in the problem. Fifth, we develop a mechanism for learning the distribution of taxi requests from large datasets of taxi trip records.

We show that combining techniques from multiple domains can yield improvements in addressing a number of existing and new problems. We conclude by a review of directions for future work.

Beknopte samenvatting

Constraint satisfaction en optimisatie (CSP(O)), probabilistische inferentie en data mining zijn drie belangrijke subdomeinen van artificiële intelligentie. CSP(O) onderzoekt methoden om efficiënt combinatoriële problemen op te lossen, probabilistische inferentie behandelt het beantwoorden van queries betreffende onzekere knowledge bases en data mining tenslotte heeft als doel het vinden en modelleren van regelmatigheden in data. Hoewel deze domeinen onafhankelijk ontwikkeld zijn, zijn er desalniettemin sterke verbanden en interacties aanwezig. Het bestuderen en uitbreiden van deze interacties is het hoofdonderwerp van deze thesis.

Deze thesis bestaat uit vijf contributies. Ten eerste bouwen we verder op bestaande methoden in probabilistische inferentie en constraint programming en stellen we een methode voor om probabiliteiten toe te voegen aan CSP(O) modellen. Dit laat ons enerzijds toe om om te gaan met constraints op probabiliteitswaarden, anderzijds om te optimaliseren over probabiliteitswaarden. Ten tweede ontwikkelen we een nieuw algoritme om het verwachte nut in stochastische constraint programma's te optimaliseren. In tegenstelling tot eerdere studies die onafhankelijkheid van willekeurig variabelen veronderstellen, nemen wij een gezamenlijke verdeling aan, voorgesteld door een Bayesiaans netwerk. Ten derde ontwikkelen we een algoritme om tot de exacte oplossing komen van het constrained clustering probleem met maximum sum of squares objective. Door het gebruik van het kolom-generatie framework, bereiken we een decompositie van het probleem in twee componenten: één dat kandidaat clusters, die voldoen aan constraints van de gebruiker, genereert, en één dat probeert de optimale oplossing te vinden tussen combinaties van de gegenereerde kandidaten. Ten vierde ontwikkelen we een exact algoritme om een speciale klasse van problemen op te lossen in het domein van grafenclustering. Het formuleren van dit probleem vereist een exponentieel aantal constraints. Wij stellen een mechanisme voor om op incrementele wijze slechts een subset van deze constraints te moeten beschouwen in het probleem. Ten vijfde ontwikkelen

we een mechanisme om de distributie van taxiverzoeken te leren uit grote datasets van taxirit opnamen.

We tonen aan dat het combineren van technieken uit verschillende domeinen voordelen biedt bij een aantal bestaande en nieuwe problemen. We besluiten met een bespreking van mogelijke onderzoeksrichtingen voor toekomstig werk.

Acknowledgements

First, I have to thank my promoters Luc De Raedt and Tias Guns for their supervision during the course of my PhD. After five years of research, I find myself heavily influenced by their perspective. Having an established notable researcher and a rising star as promoters is a rare opportunity that I was lucky to enjoy. I am thankful to Paul Van Houtte for chairing the private and public defences. I also thank the jury members Siegfried Nijssen, Patrick De Causmaecker, Anton Dries, and Christel Vrain, for the time that they devoted to reading this thesis and providing feedback.

Even though Siegfried is not officially listed as a promoter, he has had a significant role in supervising my research, in particular with respect to the work presented in chapters 3 and 5 of this thesis. Talking to him has always been an enlightening and motivating experience, and some of the most pleasant times of my PhD were the times during which we collaborated closely.

A number of other people helped me in my research, too. The research presented in chapter 7 was conducted in collaboration with Anton. He has also answered my numerous questions on programming tools and techniques. Ever since Anton shares an office with Wannes Meert, I have divided my questions between the two of them to reduce the load. I am thankful to Guy Van den Broeck for discussions that helped me better define some of my research questions. Angelika Kimmig and Jonas Vlasselaer patiently answered my questions about probabilistic inference and probabilistic logic programming. I am also thankful to Dries Van Daele and Anna Latour for the pleasant collaboration experience that I had with them.

I started my PhD sharing an office with Anton Dries and Benjamin Negrevergne. Those who know them will confirm that how delightful it is to have the simultaneous company of them. They continued to provide a much-needed atmosphere of joy and humor for years. Sharing an office with three postdocs gave me the privilege of having access to expert advice at the first steps of my

research career. In those early years, on most Fridays we headed to a meeting room to discuss matters related to the ICON project. Besides Luc and Siegfried, we were accompanied by other members of the ICON team, namely Vladimir Dzyuba, Thanh Le Van, and Sergey Paramonov. I fondly remember the coffee breaks that we had right after these meetings.

Vladimir and I started our PhD almost at the same time and were in the same class of MAI program right before. During our PhD, we participated in the ICON challenge, taught exercise sessions of the data mining course, and together with Jan Van Haaren delivered a project on technology and knowledge transfer. I also enjoyed the long discussions on technical and general topics that we had on several occasions. Behind his dark humor, I found Vladimir a kind and compassionate person.

In my first year, even a short talk with Irma Ravkic, Martin Znidarsic, or Mathias Verbeke was enough to change my mood. What they shared was their big smiles and kind hearts. I owe much more than that to Irma. Being a year senior to me, she was always there to guide me through the administrative steps of my PhD.

In the final years of my PhD, I shared an office with Davide Nitti and Francesco Orsini. Sharing an office with two nice and passionate Italian colleagues is already a pleasure. In addition to that, I learned a lot from them about subsymbolic machine learning. We never ran out of topics for technical discussions. These discussions became even more interesting when another Italian fellow, Stefano Teso, joined DTAI. Francesco and I supervised a master thesis, taught the UAI exercise sessions, and together with Dimitar Shterionov and Daniele Alfarone, organized the Fluffy workshop. Thanks for all the good memories!

Speaking of subsymbolic machine learning, I have to mention the *coursera lunch* sessions that we had going for a while. Together with Golnoosh Farnadi, Geert Heyman, Tuur Leeuwenberg, Aparna Nurani Venkitasubramania and Niraj Shrestha, we learned a lot about the trendy topic of deep neural networks.

Very Recently, I have been sharing office with two researchers from the latest generation of DTAI researchers: Gust Verbruggen and Arcchit Jain. I am amazed and delighted by their passion and energy. I hope that they do a better job in taking care of the office plants than what I did when Benjamin put me in charge upon his departure.

Another highlight of my last years was having lunch at the cafeteria with Jessa Bekker, Sebastijan Dumancic, Toon Van Craenendonck, Antoine Adam, Evgeniya Korneva, Vincent Vercruyssen, Pedro Miguel Zuidberg Dos Martires, Gust Verbruggen and Elia Van Wolputte. Gust and Elia also helped me by

translating the abstract of my thesis into Dutch.

In my difficult times over the past five years, I depended on friends that were there to hear my story and give me hope to overcome the obstacles that appeared in my way. Nasim Zand always knew how to cheer me up. Shiva Tehrani supported me during a large part of my PhD. In particular at times close to a submission deadline, I depended on her help in all matters of life. Ashkan Tabibian has done me more favors than I can ever return.

The workload of PhD research left me with little time to develop social relationships outside the department. In this situation, my close connection to two colleagues was of special value. In my first years, it was Benjamin with whom I shared my passions, frustrations, confusions and questions about research. His understanding and sympathy were always comforting. I also enjoyed his deliberately unorthodox views which were an invitation to reconsider the widely accepted traditions. I am also thankful for his so-called *pearls of wisdom* which were observations about the research activity and scientific community formulated as short and amusing phrases. I found these observations quite accurate. In the later years, Jessa was the friend that I knew I can always rely on. I'm thankful to her in particular for encouraging me to take part in more social activities. A notable example was the *international kitchen* experience with some DTAI members and Nikolina Sostaric which I truly enjoyed.

Above all, I am thankful to Golnoosh. From the early days of my PhD all the way to its end, she was there to hear me, sympathize with me, and encourage me. Our tea breaks and occasional lunches were the highlights of my days. I consider myself lucky to have her by my side throughout this journey.

Finally, I am thankful to my parents for constantly supporting me, especially at times that they did so despite not fully agreeing with my choices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Constraint satisfaction and optimization, probabilistic inference, and data mining are important subdomains of artificial intelligence, each with a long and rich history and numerous applications. Constraint satisfaction and optimization investigates methods for efficiently solving combinatorial problems, probabilistic inference deals with answering queries about uncertain knowledge bases, while data mining aims at finding and modeling regularities in the data. Despite the differences in their methods and applications, there are strong connections and interactions between these three domains. The theme of this thesis is investigating and extending such interactions. We therefore start with a brief introduction of these domains. Then we will review some of the existing work on cross-domain connections, and give an overview of the new connections that we have established in this thesis.

1.1 Constraint Satisfaction and Optimization

A *constraint satisfaction problem* (CSP) specifies a set of constraints on a set of variables. As an example, consider three variables X_1, X_2, X_3 which can take only values 0 or 1. The constraint $X_1 + X_2 + X_3 \leq 2$ restricts the values of these variables such that their sum does not exceed 2. This problem specification in terms of variables and constraints is called a *model*. To solve this CSP, one must find values for the variables such that the constraint is not violated. The values $X_1 = 0, X_2 = 1, X_3 = 0$ make up a solution.

A key idea in constraint satisfaction is to separate the problem specification

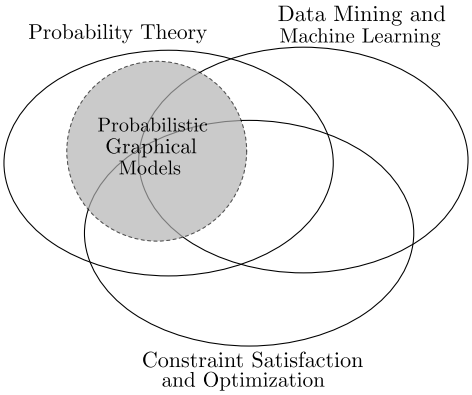


Figure 1.1: The connections between subdomains of artificial intelligence.

from the mechanism that finds the solution. An implementation of such a solving mechanism is called a *solver*. The advantage of this separation is that it offers a declarative approach for solving a problem: the user only needs to specify the problem as a model, and the solver will take care of finding the solution.

The central principles employed in constraint programming solvers are *search* and *propagation*. Search is an enumerative procedure that assigns values to a variable from its domain. Propagation is a means for reducing the number of values that are examined during search (i.e. the search space). It involves removing values from the domain of a variable by reasoning over a constraint that includes that variables and domains of other variables in that constraint.

In a CSP, any assignment that respects the constraints is a valid solution. If some solutions are preferred to others, we can define a score for solutions and ask for the solutions with the highest score. This gives a *constraint optimization problem* (COP), which is a CSP together with an *objective function* that maps each solution to a real number.

Standard constraint programming solvers can solve COPs using the *branch and bound* method. If the constraints and the objective function are linear, and the variables are integer or continuous (e.g. the knapsack problem), then we have a *mixed integer linear programming* (MILP) problem.There are solvers dedicated to finding the optimal solution for MILP problems. The key principle in these solvers is to use a relaxed version of the problem to obtain bounds during search. This relaxed problem which is obtained by dropping the integrality condition is a linear program and can be solved efficiently.

Knapsack Problem

Given a set of items and their weights and values, we want to collect a subset of them such that the total weight of collected items does not exceed the capacity of our knapsack, and their total value is maximized. We can formulate this problem as a COP:

$$\max . \sum_i v_i X_i$$

s.t.

$$\sum_i w_i X_i \leq C$$

$$X_i \in \{0, 1\} \quad \forall i$$

The binary variable X_i encodes our decision about taking or leaving item i. The constants v_i and w_i represent the value and weight of item i. The objective function $\sum_i v_i X_i$ is equal to the total value of collected items, and the constraint $\sum_i w_i X_i \leq C$ ensures that the total weight of collected items does not exceed the capacity C.

Finding an exact solution for an optimization problem can be difficult. An alternative to exact search is to use *approximation algorithms* or *heuristics* that produce near-optimal solutions at a lower computational cost. An approximation algorithm is accompanied by a bound on the ration between the near-optimal and optimal solutions. There is no such theoretical guarantee for heuristic search methods, and their effectiveness is evaluated empirically. In this thesis we follow the convention of AI community in using the term *approximate* for any method that does not produce exact solutions.

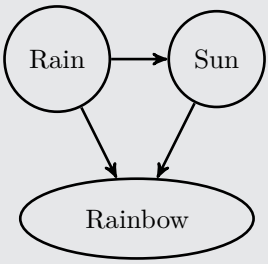
1.2 Probabilistic Reasoning

It is well known that reasoning only based on deterministic facts and rules is not enough to address the challenges of the real world, largely due to their inherently uncertain nature. A response to this problem is to use probability theory as a basis for reasoning under uncertainty. In general, this is a difficult task as it can involve visiting an exponential number of possibilities. However, there

Bayesian Network

A Bayesian network is a directed acyclic graph that represents a probability distribution. To each node in the network corresponds a conditional probability of that node given its parent nodes. The Bayesian network models the joint distribution over all nodes as the product of these conditional probabilities. The figure below shows a small Bayesian network.

$$P(Rain, Sun, Rainbow) = P(Rain) \times P(Sun|Rain) \times P(Rainbow|Sun, Rain)$$



There are inference algorithms for answering probability queries in Bayesian networks, such as $P(Rainbow = True)$ and $P(Rain = False, Sun = False)$. These algorithms use independence relationships in the Bayesian network to improve the efficiency in calculation of probability values.

are representations that allow for efficient probabilistic reasoning. Probabilistic graphical models are a class of such representations that can encode conditional independence relations between groups of random variables. The reasoning algorithms use these independencies to decompose the problem into subproblems that can be solved independently. Depending on the structure of the distribution, this can lead to significant computational savings. One of the most-studied types of probabilistic graphical models are *Bayesian networks*. In this thesis, we mostly focus on these models.

1.3 Data Mining

Data mining (DM) is concerned with discovering knowledge from data. In different data mining tasks, the discovered knowledge can have different forms. The two data mining tasks that we deal with in this thesis are pattern mining and clustering. In pattern mining, the user is interested in finding substructures that appear regularly in the data. In clustering, the discovered knowledge is presented as a grouping of data instances into a number of clusters.

Frequent Itemset Mining

Consider a database of transactions where each transaction consists of a set of items. In the toy database on the right, the first to third transactions are $\{B\}$, $\{E\}$, and $\{A, C\}$. The problem of frequent itemset mining is to find subsets of items (called itemsets) that are included in at least as many transactions as a given threshold.

A	B	C	D	E
0	1	0	0	0
0	0	0	0	1
1	0	1	0	0
1	0	0	0	1
0	1	1	0	0
0	0	0	1	1
0	0	1	1	1
1	1	1	0	0
1	1	0	0	1
1	1	1	0	1

Some of the frequent itemsets for threshold 2 in this toy database are $\emptyset$, $\{E\}$, $\{B, C\}$, and $\{A, B, E\}$. If we add the constraint that the itemsets must contain at least two items, the itemsets $\emptyset$ and $\{E\}$ will be excluded.

The goal in the task of *frequent pattern mining* is to find substructures that appear more than a certain number of times in the data. Extensive research has been conducted on efficient algorithms for finding frequent patterns of different types.

Usually the large number of discovered frequent patterns makes it difficult for the user to analyze them. This has motivated the research on methods for reducing these results to a smaller set of patterns. *Constraint-based pattern mining* reduces the number of output patterns by requiring the patterns to satisfy extra constraints other than frequency. Another approach is to use *condensed representations* which means to eliminate the redundancies in the discovered patterns.

Clustering is a descriptive data mining task. The goal in clustering is to create a model of the data in terms of groups that constitute it. The clustering algorithms aim to group the data into clusters that have certain properties. One such property that is common to most clustering algorithms is that the members of a cluster are similar to each other and different from the members of other clusters.

The desired properties of clusters can be represented in terms of constraints. A *must-link* constraint between two data instances requires that these instances belong to the same cluster. A *can-not-link* constraint forbids such a co-membership. Another example is the *size* constraint which restricts the number

of members of a cluster.

1.4 Connections between Subdomains of Artificial Intelligence

The connections between constraint satisfaction and optimization (CSP(O)), probabilistic inference, and DM/ML have been studied before. Figure 1.2 positions our work in relation to the domains of AI and these cross-domain studies. In this thesis, we propose alternative frameworks for adding probabilistic models to CSP(O) formulations. Our aim is to exploit both probabilistic and deterministic structures in solving these problems. Our work on pattern mining in Bayesian networks (CP4BN in figure 1.2) links three topics of DM/ML, CSP(O), and PGM. In our work on factored stochastic constraint programming (FSCP in figure 1.2), we build on an existing work on stochastic constraint programming (SCP in figure 1.2) which combined CSP(O) with probabilities(Walsh 2002).

We contribute to topics at the intersection of CSP(O) and DM by formulating and solving mining and learning tasks using integer linear programming and constraint programming. Our two works on clustering (CCCG and BNCC in figure 1.2) are a continuation of a trend on combining DM/ML and CSP(O). This trend was initiated by a framework (CP4IM in figure 1.2) for solving

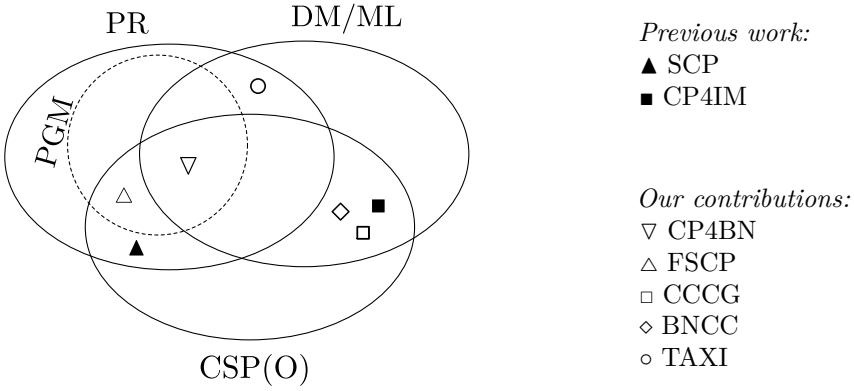


Figure 1.2: Existing and new research on connections between multiple domains: Probability Theory (PR), Probabilistic Graphical Models (PGM), Data Mining/Machine Learning (DM/ML), and Constraint Satisfaction/Optimization (CSP(O)).

itemset mining problems by constraint programming (Guns et al. 2011b). We also extend this framework to pattern mining in Bayesian networks. Finally, in our work on learning taxi passenger demand (TAXI in figure 1.2), we learn probabilistic models from the data using standard methods in statistical machine learning.

The matter of combining optimization and probabilistic inference has been studied in the uncertainty reasoning community, for example under the topic of influence diagrams. However, constraint processing has not received much attention in these studies. Such a combination has been also studied in the constraint programming community. But it is common practice in these studies to either assume that the random variables are independent (Walsh 2002) or to sample scenarios from the probability distribution and combine them into a single deterministic constraint program (Manandhar et al. 2003). The former method assumes strict independence relationships and the latter approach (called *scenario-based stochastic constraint programming*) ignores the structure of the probability distribution. Sampling scenarios has been also a successful approach for solving stochastic vehicle routing problems (Bent and Hentenryck 2004).

Research at the intersection of DM and CSP(O) has a natural motivation. At the core of data mining tasks usually lies a search in a hypothesis space for finding all/best hypotheses according to some criteria. The standard practice for finding these hypotheses is to develop search algorithms that are targeted at that specific learning or mining problem. The search space can be discrete or continuous, and the algorithms can be approximate or exact.

An alternative approach for solving these problems is to formulate them as CSP(O) models. This approach has multiple advantages: 1) Moving away from developing algorithms to formulating the problem in a certain language, makes it easier to rapid-prototype new ideas. 2) Once a good formulation for a task is developed, variants of it can be obtained by modifying the constraints and/or the objective function. 3) Advances in constraint solving technologies translate into improvements in mining and learning tasks.

There are successful examples of using this principle in continuous domain, such as using mathematical programming solvers for solving the underlying optimization problem in support vector machines (Schölkopf et al. 2000). Recently, there has been an interest in using the same approach for problems with a combinatorial search space. Several pattern mining tasks have been formulated as constraint satisfaction problems and solved using standard constraint programming solvers (Guns et al. 2011b; Guns et al. 2011a; Négrevergne and Guns 2015). A similar approach (although using other constraint solving frameworks) has been applied to tasks such as structured-output prediction (Teso

et al. 2017) and inference in statistical relational learning (Riedel 2008).

An area in DM where using CSP(O) has been popular is constraint-based learning and mining. In constraint-based learning, the space of valid hypotheses is expressed in terms of constraints. An example of using CS for constraint-based supervised learning is structure learning of Bayesian networks, which has been formulated using integer linear programming (Bartlett and Cussens 2017). In constrained clustering, which is an example of unsupervised constraint-based learning, the desired properties of clusters are expressed through constraints. By formulating the clustering problem as a constraint programming model, a wide range of such properties can be enforced by adding extra constraints to the base model (Dao et al. 2017).

Similarly, in constraint-based pattern mining, the desired properties of patterns are described by constraints such as *frequency* and *closedness*. When a pattern mining task is modeled as a constraint solving problem, such constraints can be enforced by adding extra constraints, or modifying the existing ones (Guns et al. 2011b).

1.5 Contributions

This thesis has three parts. In each part we study the connections between a number of domains in artificial intelligence. In the first part we present mechanisms for integrating constraint programming and probabilistic inference. In the second part, we evaluate the potential of integer linear programming for the task of constrained clustering. In the third part we investigate a learning problem which is part of a stochastic optimization pipeline. The research questions that are answered in each of the three parts are:

Q1 *How can we use probabilistic graphical models within CSP(O) formulations?*

Q2 *What is the potential of formulating constrained clustering as integer linear programming problems?*

Q3 *How can we use data mining techniques to learn the distribution of passenger requests from records of taxi trips?*

The contributions of this thesis with respect to question **Q1** are as follows:

- We developed a way to represent the results of probability queries as variables in a CSP model. This method uses existing technologies in probabilistic inference and constraint programming. This method allows solving CSP(O) problems that have constraints or objective functions that

are defined in terms of the result of a probability query. We illustrate this technique by applying it to a novel data mining task, namely constraint-based pattern mining in Bayesian networks.

- We developed a novel algorithm for optimizing the expected utility in constraint programs which have probabilistic parameters. These parameters can have a joint distribution represented by a Bayesian network. We developed a novel bounding mechanism that takes advantage of the probabilistic structure. Our method outperforms the existing algorithms for solving such problems.

The contributions obtained in this thesis with respect to question **Q2** are the following:

- We developed an algorithm for obtaining the exact solution for the constrained clustering problem with the maximum sum of squares (MSS) objective. This algorithm is based on formulating the clustering problem as an integer linear program and solving it by using the column generation method. Our algorithm supports a set of constraints that are common in constrained MSS clustering.
- Motivated by a data mining application, we developed an exact algorithm to solve a special class of graph clustering problems. We developed two integer linear programming formulations of this problem. Our solution methods are efficient enough for finding the optimal solution of real-world instances that motivated this problem.

Finally, we have the following contribution with respect to question **Q3**:

- We developed a mechanism for learning the distribution of taxi requests from large datasets of taxi trip records. In our experiments, the model obtained through this mechanism outperformed the existing approaches.

1.6 Structure of the thesis

We first present the background material in **Chapter 2**. This chapter provides the background on a number of topics that are used in the subsequent chapters, namely constraint programming, integer linear programming, and inference in graphical models.

The structure of rest of the thesis follows the three research questions formulated above. The three parts of this thesis are as follows:

1.6.1 Part I: Probabilistic Models in Constraint Satisfaction and Optimization

Chapter 3 introduces the new problem of pattern mining in Bayesian networks. Similar to (Guns et al. 2011b), we use constraint programming to formulate the problem. One difference between pattern mining in Bayesian networks and in databases is that in the former the input is represented intensionally in a compact form. Instead of unfolding this compact form into its extensional equivalent, we use knowledge compilation to compile this distribution into an intermediate structure which is then embedded in a constraint program as a set of constraints. This chapter has been previously published in the following paper:

- Behrouz Babaki, Tias Guns, Siegfried Nijssen, and Luc De Raedt. "Constraint-based querying for bayesian network exploration". In Élisa Fromont, Tijl De Bie, and Matthijs van Leeuwen, editors, *Advances in Intelligent Data Analysis XIV - 14th International Symposium, IDA 2015, Saint Etienne, France, October 22-24, 2015, Proceedings,* volume 9385 of *Lecture Notes in Computer Science,* pages 13–24. Springer, 2015.

Chapter 4 presents a new flavor of stochastic constraint programming. In this setting, the joint distribution of random variables is represented by a Bayesian network. Hence we deal with a deterministic and a probabilistic structure. Existing methods can only exploit one of these two structures. We exploit both structures by combining two computational tasks of probabilistic inference and constraint satisfaction in an And-Or search tree. This chapter is based on the following paper:

- Behrouz Babaki, Tias Guns, and Luc De Raedt. "Stochastic constraint programming with and-or branch-and-bound". In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017,* pages 539–545. ijcai.org, 2017.

1.6.2 Part II: Constrained Clustering using Integer Linear Programming

The two chapters in this part present applications of CSP(O) in DM.

Chapter 5 The goal in this chapter is to use integer linear programming for solving a clustering problem. The number of variables in our formulation is

exponential in the number of data instances. We show that this problem can be solved by adding the variables to the model in a lazy fashion. This results in a hybrid scheme that allows us to deal with a subset of constraints in a subproblem outside the integer programming workflow. This chapter is previously published in this paper:

- Behrouz Babaki, Tias Guns, and Siegfried Nijssen. "Constrained clustering using column generation". In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings,* volume 8451 of *Lecture Notes in Computer Science,* pages 438–454. Springer, 2014.

Chapter 6 deals with a clustering problem where the relations among data instances are represented as a graph. A key requirement in this problem is that the subgraphs induced by the clusters must be connected. We formulate this problem as an integer linear program, and express the connectedness property as a set of constraints. Since the number of these constraints is exponential in the size of input graph, we solve the problem by including only a sufficient subset of these constraints. This chapter is published in this paper:

- Behrouz Babaki, Dries Van Daele, Bram Weytjens, and Tias Guns. "A branch-and-cut algorithm for constrained graph clustering". *Data Science meets Optimization workshop (colocated with CPAIOR), Padova, Italy, 2017.*

1.6.3 Part III: Learning Taxi Passenger Demand

Chapter 7 deals with the problem of learning the passenger request for taxi trips. The motivation for learning this model is to produce samples to be used by a scenario-based algorithm for the stochastic routing problem. In this chapter we study the problem of learning such distributions from large amounts of data. This chapter is based on the following work:

- Behrouz Babaki, and Anton Dries. *Feature-based Taxi Request Prediction.* (manuscript in preparation).

Finally, **Chapter 8** concludes this thesis by presenting a summary, conclusions and directions for future research.

Chapter 2

Background

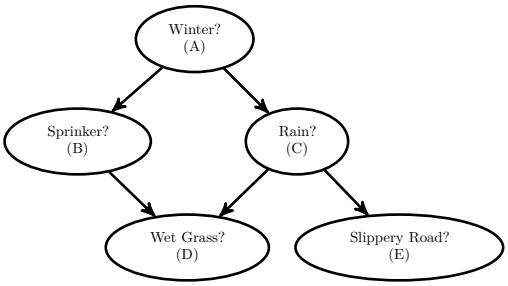
This chapter provides the background on probabilistic graphical models, constraint programming, integer programming and constraint-based pattern mining.

2.1 Bayesian Networks

The idea in probabilistic reasoning is to use probability theory to reason over uncertain beliefs. This requires the belief to be represented as a probability distribution. However, the number of possible worlds grows exponentially in the number of entities that one holds a belief about. This immediately poses a challenge both in terms of representation and reasoning. In the general case, one will need to visit all these possibilities and assign a probability to each one. Similarly, at inference time all these possibilities need to be enumerated.

Probabilistic graphical models address this problem using a basic insight. Usually, the *conditional independencies* provide significant information about beliefs. The language of graphs can easily represent these independencies. Using these models, it is sufficient to specify these independences as a graph, and only specify the probability distribution for each factor. *Bayesian networks* are one of the most popular types of probabilistic graphical models.

Definition 1. *A* Bayesian network $\mathcal{G}$ *is a directed acyclic graph where each node represents a random variable* X_i *in* $\mathcal{X} = \{X_1, \ldots, X_n\}$*. Let* $Pa^{\mathcal{G}}_{X_i}$ *denote the parents of* X_i *in* $\mathcal{G}$*. A joint distribution* P *over the set of variables* $\mathcal{X}$ *is said to factorize according to* $\mathcal{G}$ *if* $P(X_1, \ldots, X_n)$ *can be expressed as the product*



A	$P_1(A)$
0	0.4
1	0.6

A	B	$P_2(B\|A)$
0	0	0.25
0	1	0.75
1	0	0.8
1	1	0.2

A	C	$P_3(C\|A)$
0	0	0.9
0	1	0.1
1	0	0.2
1	1	0.8

A	C	D	$P_4(D\|B,C)$
0	0	0	1
0	1	0	0
0	0	1	0.2
0	1	0	0.8
1	0	0	0.1
1	1	0	0.9
1	0	1	0.05
1	1	0	0.95

C	E	$P_5(E\|C)$
0	0	1
0	1	0
1	0	0.3
1	1	0.7

Figure 2.1: A Bayesian network over five variables

$\prod_{i=1}^{n} P(X_i|Pa_{X_i}^{\mathcal{G}})$. *We denote the distribution factored according to* $\mathcal{G}$ *by* $P_{\mathcal{G}}$. *We denote by* $D(X_i)$ *the domain of variable* X_i, *that is, the possible values the variable can take. An assignment of value* x_i *to variable* X_i *is denoted by* $(X_i = x_i)$.

The joint probability of variables in a Bayesian network is fully specified by its graph and the distributions $P(X_i|\mathrm{Pa}_{X_i}^{\mathcal{G}})$. When the variables are discrete, each of these distributions can be represented extensionally as a conditional probability table (CPT) (Pearl 1989).

A *probabilistic query* is a question about probability of events according to a distribution. These queries are answered by *probabilistic inference* algorithms. There are several types of such algorithm (for a detailed discussion of these algorithms, see (Darwiche 2009)). The common principle among them is to take advantage of the conditional independencies to decompose the inference problem.

Example 1. *Consider the graph in figure 2.1. It shows that the joint probability of random variables* A, B, C, D, E *factorizes as follows:*

$$P(A, B, C, D, E) = P_1(A) \cdot P_2(B|A) \cdot P_3(C|A) \cdot P_4(D|B, C) \cdot P_5(E|C)$$

The joint probability distribution is fully described by this graph and the conditional probability tables for $P_1, \ldots, P_5$. *The query* $P(A = 1)$ *in example 1*

can be decomposed and solved as follows:

$$
\begin{aligned}
P(A = \textit{true}) &= \sum_{B,C,D,E} P(A = 1, B, C, D, E) \\
&= \sum_{B,C,D,E} P_1(A = 1) \cdot P_2(B|A = 1) \cdot P_3(C|A = 1) \cdot P_4(D|B, C) \cdot P_5(E|C) \\
&= \sum_{B,C,D} P_1(A = 1) \cdot P_2(B|A = 1) \cdot P_3(C|A = 1) \cdot P_4(D|B, C) \cdot \underbrace{\sum_{E} P_5(E|C)}_{=1} \\
&= \sum_{B,C} P_1(A = 1) \cdot P_2(B|A = 1) \cdot P_3(C|A = 1) \cdot \underbrace{\sum_{D} P_4(D|B, C)}_{=1} \\
&= \sum_{B} P_1(A = 1) \cdot P_2(B|A = 1) \cdot \underbrace{\sum_{C} P_3(C|A = 1)}_{=1} \\
&= P_1(A = 1) \cdot \underbrace{\sum_{B} P_2(B|A = 1)}_{=1} \\
&= P_1(A = 1) = 0.6
\end{aligned}
$$

2.2 Probabilistic Inference by Knowledge Compilation

One of the methods for inference in Bayesian networks is to first compile it to an *arithmetic circuit* and then evaluate the query on the circuit. Compilation is an offline step and has to take place only once for each network. The resulting circuit can be used for answering multiple queries (Darwiche 2003).

Consider the Bayesian network in left side of figure 2.2. Compiling this network results in the circuit depicted in the right side of the figure. We will now see how this circuit can be used to answer probability queries. The circuit has two types of leaves: *parameters* which are set according to the probability values in the network CPTs, and the λ variables, which are called *indicators*. The indicator

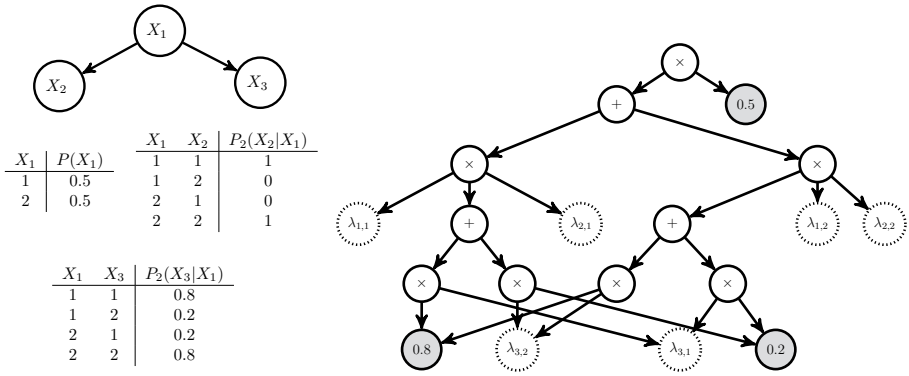


Figure 2.2: Left: a Bayesian network with three variables. Right: The arithmetic circuit obtained by compiling the Bayesian network on the left (figure from (Darwiche 2009))

variables are set according to the query that the user wants to evaluate. Once these variables are set, the circuit can be evaluated using a bottom-up pass.

For each variable X_i and value $j \in D(X_i)$ there exists an indicator variable denoted by λ_{ij}. To evaluate the query $P(X_{i_1} = j_1, \ldots, X_{i_k} = j_k)$, the indicator variables should be set as follows:

- If a variable is assigned to a value in the query, set the indicator variable corresponding to that assignment to 1, and those corresponding assignment to other values of that variable to 0. For example in the query above, the indicator variables λ_{i_1,j_1} should be set to 1 and all other $\lambda_{i_1,j}$ for $j \neq j_1$ should be set to 0.
- If a variable is not mentioned in the query (i.e. it is marginalized over), the indicator variables corresponding to all values of this variable should be set to 1.

Figure 2.3 shows how the query $P(A = 1, C = 0)$ is evaluated.

An arithmetic circuit can be seen as a function f over indicator variables. Consider a partial assignment A, Bayesian network variable X_i and the value $j \in D(X_i)$. It has been shown that the following equality holds:

$$\frac{\partial f}{\partial \lambda_{i,j}}(A) = \begin{cases} P(A \cup \{(X_i = j)\}) & \text{if } X_i \text{ is not assigned in } A \\ P(A \backslash \{(X_i = k)\}) \cup \{(X_i = j)\}) & \text{if } (X_i = j) \in A \end{cases}$$

It has also been shown that the derivatives of f with respect to all λ_{ij} variables can be computed in a single top-down pass. Algorithm 1 shows how the value of f and its derivatives are computed in two passes through the circuit. It takes the arithmetic circuit $\mathcal{AC}$ and two arrays vr and dr for storing the value and derivative in each node. The value and derivative corresponding to node v in the circuit are represented by $\mathsf{vr}(v)$ and $\mathsf{dr}(v)$. It is assumed that the values of leaf nodes are initialized in $\mathsf{vr}(v)$. Let us denote the root node of the circuit by r. The function TwoPass computes the value of circuit output in the $\mathsf{vr}(r)$. It also computes the derivatives of leaf nodes v in $\mathsf{dr}(v)$. The computed derivatives for the circuit of figure 2.2 are shown in figure 2.3.

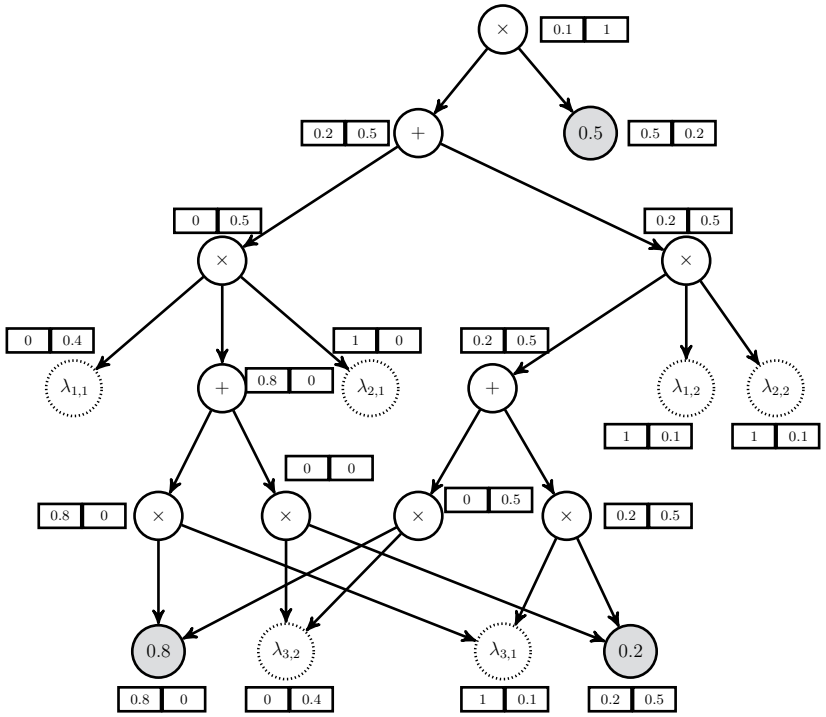


Figure 2.3: The result of circuit evaluation and differentiation for the query $P(X_1 = 2, X_3 = 1)$ by algorithm 1. The vr values are shown in the left boxes and the dr values are shown in the right boxes.

Algorithm 1 Evaluating and differentiating an arithmetic circuit (from Darwiche 2009)

function TwoPass($\mathcal{AC}, vr, dr$)
  for each circuit node v (visiting children before parents) **do**
    compute the value of v and store it in $\mathsf{vr}(v)$
  end for
  $\mathsf{dr}(v) \leftarrow 0$ for all non-root nodes v
  $\mathsf{dr}(r) \leftarrow 1$ ▷ r is the root node
  for each circuit node v (visiting parents before children) **do**
    for each parent p of node v **do**
      if p is an addition node **then**
        $\mathsf{dr}(v) \leftarrow \mathsf{dr}(v) + \mathsf{dr}(p)$
      else
        $\mathsf{dr}(v) \leftarrow \mathsf{dr}(v) + \mathsf{dr}(p) \prod_{v' \neq v} \mathsf{vr}(v')$, where v' is a child of parent p
      end if
    end for
  end for
end function

2.3 Constraint Programming

Constraint programming is an instance of the *declarative programming* paradigm. Like other declarative methods, in constraint programming the user only specifies the problem and not the solution method. Constraint programming systems are equipped with solvers that decide how to search for solutions to the given problem. A constraint satisfaction problem is a triple $(\mathcal{V}, D, \mathcal{C})$ in which $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a set of domains which map every variable $v \in \mathcal{V}$ to a set of values $D(v)$, and $\mathcal{C}$ is a set of constraints which further limit values from D that can be assigned to $\mathcal{V}$ (F. Rossi et al. 2006).

Example 2. *We have three items with weights 10, 15, and 20. Each item has a value. The values of these three items are 5, 1, and 10. We want to pick two of these items while respecting the following conditions: 1) The sum of weights of selected items should not exceed 35, and 2) If we pick the first item, we can not select the third one.*

To formulate this problem in constraint programming, we introduce discrete variables x_1, x_2, and x_3 as indicators for selection of each of three items. We also introduce the real variable s which represents the total weight of selected items. So $\mathcal{V} = \{x_1, x_2, x_3, s\}$. The domain of discrete variables is the finite set $\{0, 1\}$, and the domain of continuous variable is the interval $[0, 45]$. In other words, $D(x_1) = D(x_2) = D(x_3) = \{0, 1\}$ and $D(s) = [0, 45]$. Finally, set $\mathcal{C}$

consists of following constraints:

$$x_1 + x_2 + x_3 = 2 \tag{2.1}$$

$$s = 10x_1 + 15x_2 + 20x_3 \tag{2.2}$$

$$s \leq 35 \tag{2.3}$$

$$(x_1 = 1) \rightarrow (x_3 = 0) \tag{2.4}$$

Constraint 2.1 reflects that we want to pick exactly two items. Constraint 2.2 specifies that s is sum of selected items. Constraints 2.3 and 2.4 reflect the two conditions specified in the problem description[1]*.*

The two main operations of a constraint solving system are **search** and **propagation**. Search is the act of trying different values (or ranges of values) for variables. For discrete variables, this is done by assigning values from the domain of that variable. For continuous variables, search is done by partitioning the domain into disjoint intervals. The intervals that have a size smaller than a predefined precision will not be partitioned anymore.

Propagation is the act of removing values (or ranges) from the domain of variables that given the domains of other variables and the model constraint, cannot be part of a solution. Given a constraint c defined over variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$, we define the set of admissible values for variable x_1 as:

$$\{a_1 \in D_1 : \exists a_2 \in D_2, \ldots, \exists a_n \in D_n \text{ such that } C(a_1, \ldots, a_n) \text{ holds}\}$$

For continuous variables, the propagation mechanism computes a superset, namely the smallest interval enclosing this set.

Example 3. *Consider a simple model consisting of three continuous variables x_1, x_2, and y with domains $[a, b]$, $[c, d]$, and $[-\infty, \infty]$, respectively. Propagation according to the constraint $y = x_1 + x_2$ will reduce the domain of variable y to $[a + c, b + d]$. If we replace this constraint by $y = x_1 * x_2$, the domain of y will be reduced to $[min(ac, ad, bc, bd), max(ac, ad, bc, bd)]$.*

If we reduce the domains of variables in a CSP in such a way that all constraints hold for each value in the domain of each variable, we have found a solution for that CSP.

[1] This problem can be modeled without using continuous variables. However, we have included variable s for the sake of demonstration.

Example 4 (Example 2 continued)**.** *For solving the problem formulated in example 2, we start by assigning value 1 for* x_1*. This assignment, together with constraint 2.2 triggers a propagation step which reduces the domain of* s *to* $[10, 45]$*. A similar propagation step based on constraint 2.4 removes value 1 from the domain of* x_3*. This in turn changes the domain of* s *to* $[10, 25]$*. We continue by assigning value 0 to* x_2*. As a result, the domain of* s *reduces to* $[10, 10]$*. At this point, all variables are assigned a value from their domain, and all constraints hold for these values. This means that we have found a solution.*

Note that in the previous example, we did not have to search for the value of s, as its value was determined by propagation. In other words, our previous assignments to other variables, dictated that s should be equal to 10.

Global constraints A global constraint is a constraint over a non-fixed number of variables. Global constraint are often semantically redundant; meaning that they can be represented by a set of simpler constraints. However, a global constraint provides better access to the structure of the problem. A famous example is the $\texttt{alldifferent}(x_1, \ldots, x_n)$ constraint which enforces that each pair of variables $x_1, \ldots, x_n$ should take different values. Consider $\texttt{alldifferent}(x_1, x_2, x_3)$. This constrain can be replaced by three inequality constraints $x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3$. Now consider a situation where the domain of all three variables is $\{1, 2\}$. The `alldifferent` constraint can reason over these domains and issue a failure. However, the inequality constraints do not have access to this global view and can not detect the failure.

Constraint optimization A *constraint optimization* problem is a CSP together with a function f defined over $\mathcal{V}$. The goal is to find a solution S^* such that $f(S^*) \leq f(S)$ for all solutions S to the problem. This problem can be solved using a CSP solver using a simple procedure. Initially a solution S is obtained by backtracking search. Then the constraint $f(\mathcal{V}) < S$ is added to the constraint set $\mathcal{C}$. Solving this problem will give a solution with a smaller objective function. These steps are repeated until no solution is found. The solution obtained last is optimal.

2.4 Mixed Integer Linear Programming

A *mixed integer linear programming* (MILP) problem is a constraint optimization problem with linear constraints and objective function. This class of optimization problems has been extensively studied and sophisticated solvers have been

developed for solving this type of problems (Achterberg 2007). These problems can contain both integer and continuous variables. A MILP problem can be specified by the tuple (A, b, c, I) where $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix, $b \in \mathbb{R}^m$ is the right-hand-side vector, $c \in \mathbb{R}^n$ is the cost vector, and $I \subseteq \{1, \ldots, n\}$ is the set of indices of integer variables. The vector of decision variables is denoted by x. By definition, $x_j \in \mathbb{Z}, \forall j \in I$ and $x_j \in \mathbb{R}, \forall j \notin I$. Further restrictions on domains of variables can be specified as bound constraints $l_j \leq x_j \leq u_j$ which are a special case of linear constraints. The problem is to find values for x_j variables from their domains such that the objective function $c^T x$ is minimized and the constraints $Ax \leq b$ hold.

Given a constraint optimization problem P, a relaxation P_R is an optimization problem such that each solution of P is a valid solution for P_R. A *linear programming (LP) relaxation* of a MILP problem is obtained by turning the integer variables into continuous variables. This gives a linear programming problem that can be solved efficiently. MILP solvers heavily rely on solving these LP relaxations. These solvers employ a range of sophisticated mechanisms. We will review the two most important mechanisms, namely branch-and-bound search and the cutting planes method.

2.4.1 Branch-and-bound search

As indicated by its name, *branch-and-bound search* relies on two principles. *Branching* is the practice of dividing a problem into smaller instances. In MILP solvers, these subproblems are generated by adding constraints to the base problem. Note that the disjunction of these constraints should not remove any solutions from the base problem. This process resembles the search in CSP solvers. The *bounding* mechanism is employed in order to avoid the enumeration of all solutions of a problem. During the search, the smallest objective value among all discovered solutions up to that moment produces an upper bound. If the lower bound for a subproblem exceeds this upper bound, that subproblem can be safely ignored. The lower bound is obtained by solving the LP relaxation of the subproblem. An important factor that determines the strength of this bound is how close the constraints of the relaxed problem are to the original constraints. The *strength* of an LP relaxation is the degree to which this holds. Besides providing a bound, the LP relaxation can also guide the branching decisions. The most-commonly used branching strategy in MILP solvers is to pick an integer variable x_j which has a fractional value $\hat{x}_j$ in the solution of the LP relaxation. The two branches are created by adding the constraints $x_j \leq \lfloor \hat{x}_j \rfloor$ and $x_j \geq \lfloor \hat{x}_j \rfloor$. This branching mechanism is depicted in figure 2.4.

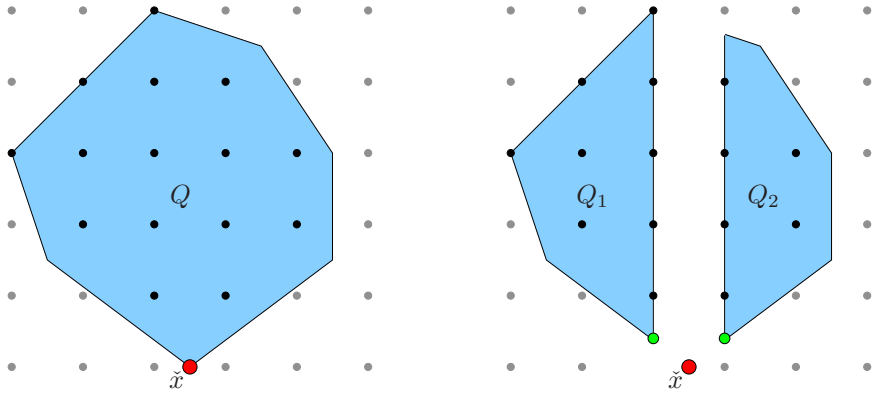


Figure 2.4: Branching decomposes problem Q into two subproblems Q_1 and Q_2. These subproblems are constructed by adding constraints $x_j \leq \lfloor \hat{x}_j \rfloor$ and $x_j \geq \lfloor \hat{x}_j \rfloor$ to Q. Figure from (Achterberg 2007).

2.4.2 Cutting planes

The performance of the branch and bound scheme depends on the strength of the LP relaxations that are solved during the search. These LP relaxations can be strengthened using the *cutting plane* method (Wolsey 1998). Given solution $\hat{x}$ for an LP relaxation of a MILP problem, a cutting plane is an additional linear constraint that is violated by $\hat{x}$ but does not cut off any solution from the original MILP problem. This constraint separates $\hat{x}$ from the convex hull of integer solutions and thereby strengthens the LP relaxation. This concept is depicted in figure 2.5. There are general methods for generating cutting planes for integer programs.

In addition to the cuts automatically generated by the solver, the user can also provide cuts throughout the branch and bound tree. These cuts can be added each time that the linear relaxation is solved, or when a new feasible solution for the MILP problem is obtained. This approach is typically used when the number of constraints in problem formulation is huge. In this method two steps are iteratively repeated: 1) A model that includes only a subset of the constraints is solved. 2) A constraint that is violated by the current solution, a cut, is added to the model. These steps are repeated until no constraint is violated. To use this method, we need an oracle that given an assignment x can check if x satisfies all constraints and if not, finds a constraint that is violated by x. Since in the latter case the added constraint separates x from the feasible region, the problem solved by the oracle is called the *separation* problem.

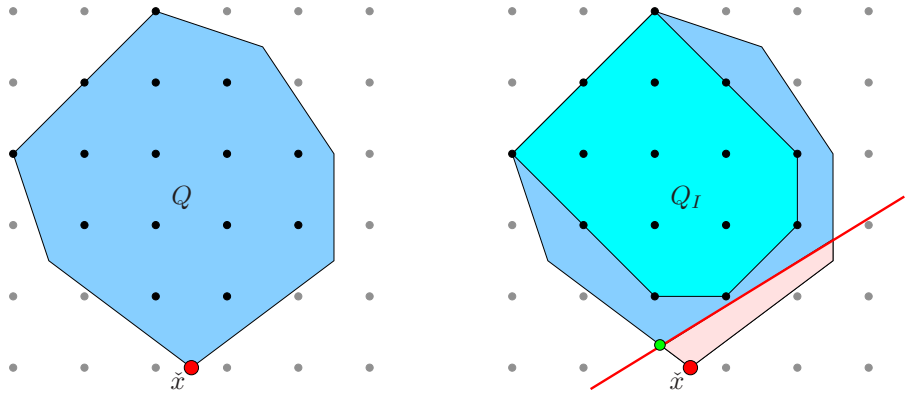


Figure 2.5: To strenghten the LP, a cut separates the solution of LP relaxation $\hat{x}$ from the convex hull of the integer program. Figure from (Achterberg 2007)

2.4.3 Column Generation

Column generation is a method for solving linear programs that have a huge number of variables. It has a principle similar to the cutting plane method. In cutting planes method we incrementally include a subset of constraints in the model. Similarly, in column generation we incrementally add variables to the model until a proof of optimality is obtained. This method makes it possible to obtain the optimal solution by only including a manageable subset of the variables in the model (Feillet 2010).

We will first introduce some notations. Consider the linear programming problem with objective function $\min cx$ subject to m linear constraints represented by inequalities $Ax \leq b$. For a variable x_i, we denote the corresponding column in A by a_i. Let I denote the set of indices of variables that are already included in the LP model. Let x_I and c_I denote the vector of included variables and their cost in the objective function. Let A_I be the matrix composed of columns in A that correspond to x_I. For variables that have not been added to the model, we similarly define the index set E, vectors x_E and c_E and the matrix A_E.

The first step in the column generation algorithm is to solve the restricted problem $\min c_I x_I$ subject to constraints $Ax_I \leq b$. After solving this problem to optimality, besides the solution to the restricted problem, we obtain the vector of values $\lambda \in \mathbb{R}^m$. In the standard simplex method, these values are used for checking the optimality or proceeding with the next iteration of the algorithm. Following the same principle, in column generation we use these values for verifying optimality or adding a new variable.

To verify optimality, we first try to find a variable with a negative *reduced cost*, i.e. a variable x_j for which $c_j - \lambda a_j < 0$. If such a variable exists, then it is moved from x_E to x_I and this procedure is repeated. Otherwise, the obtained solution is optimal and the algorithm terminates.

Since there are a large number of variables in x_E, the task of finding the variable with the smallest reduced cost gives rise to another search problem, called the *pricing problem*. The efficiency of a column generation algorithm is heavily affected by the efficiency of this pricing subproblem.

Branch and price The column generation algorithm solves linear programming problems. To solve a MILP problem with a large number of variables, the column generation method can be used within a standard branch and bound algorithm for solving the LP relaxations. This combination is called the *branch and price* method.

2.5 Pattern Mining

The goal of pattern mining is to find all patterns π in a dataset $\mathcal{D}$ that have certain properties. These properties are specified as a set of constraints p. The set of all patterns is described by a language $\mathcal{L}$. The pattern mining problem is concerned with finding all patterns in language $\mathcal{L}$ that satisfy p, that is the set $\{\pi \in \mathcal{L} | p(\pi, \mathcal{D}) \text{ holds}\}$ (Mannila and Toivonen 1997). A well-known example is the problem of finding frequent itemsets which we now define formally.

Definition 2 (Frequent itemset mining)**.** *Assume a set of entities* $\mathcal{I} = \{i_1, \ldots, i_n\}$*, called* items*. Let* $\mathcal{D}$ *be a set of* transactions*, such that each* $T \in \mathcal{D}$ *is a set of items (i.e.* $T \subseteq \mathcal{I}$*). An* itemset *is a set of items. Assuming that we are given a threshold* θ*, a* frequent itemset *is an itemset which is a subset of at least* θ *transactions.* Frequent itemset mining *is the problem of finding such itemsets.*

Frequent itemsets are a building block for some data mining tasks which have a focus on finding patterns in databases. The most prominent application of frequent itemsets is in finding association rules, which express how frequently two (sets of) items appear together in the data (Agrawal and Srikant 1994). For example, in a database of customer purchases, an association rule ($\{\text{wine,cheese}\} \rightarrow \{\text{grapes}\}\,(70\%)$) states that 70 percent of customers who have bought wine and cheese, have also bought grapes.

$\mathcal{D}$	Itemsets
T_1	$\{A, B\}$
T_2	$\{C\}$
T_3	$\{A, D\}$
T_4	$\{A, B, C\}$

Table 2.1: An itemset database

Example 5. *Table 2.1 shows a database $\mathcal{D}$ over items $\{A, B, C, D\}$. For $\theta = 2$, itemsets $\{A\}$, $\{B\}$, $\{C\}$, and $\{A, B\}$ are frequent. A rule that can be extracted using these itemsets is $\{\{A\} \rightarrow \{A, B\} : (66\%)\}$.*

The size of the search space of all itemsets is $2^{|\mathcal{I}|}$. This means that even for moderate number of items, simple generate-and-test methods are not sufficient. Over the past two decades, considerable effort has been invested in developing efficient methods for enumerating the solutions. The proposed methods vary along aspects such as search space traversal (depth-first search and breath-first search) and database representation (row-based, column-based, and tree-based) (Aggarwal et al. 2014).

2.5.1 Constraint-based pattern mining

A problem of frequent itemset mining is that usually the number of discovered patterns is too large. This number can be reduced if we set the frequency threshold too high; but then only the well-known patterns will be discovered. This indicates that the frequency constraint is not sufficient for discovering the interesting patterns. The general idea in *constraint-based pattern mining* is to resolve this problem by adding other types of constraints (Nijssen and Zimmermann 2014).

An important class of constraints concern the ones that restrict the set of discovered patterns to *condensed representations*. They are meant to reduce the redundancy in the discovered patterns. A pattern is redundant if other discovered patterns imply that this pattern conforms to the constraints. An example condensed representation in frequent itemset mining is *maximal itemset*. A frequent itemset is maximal if all its supersets are infrequent. Maximal itemsets form a border between the frequent and infrequent itemsets. The problem of mining maximal can be formally defined as follows.

Definition 3 (Maximal Frequent Itemset Mining)**.** *Given an itemset database $\mathcal{D}$ and a threshold θ. Let $frequency_{\mathcal{D}}(I)$ denote the frequency of itemset I in this database. The maximal frequent itemset mining problem consists of*

computing the set

$$\{I | I \subseteq \mathcal{I}, frequency_D(I) \geq \theta, \forall I' \supset I : frequency_D(I') < \theta\}$$

The naive way of enforcing these constraints is to first discover the frequent itemsets and then filter the discovered patterns according to the rest of constraints. However, if the frequency is low, a large number of patterns will be first generated and later disposed. The alternative approach is to enforce the constraints during search. This has motivated various algorithms for solving pattern mining problems subject to different classes and combinations of constraints.

2.5.2 Frequent pattern mining using constraint programming

A recent trend for solving frequent itemset mining problems is to formulate them as constraint programming problems (Guns et al. 2011b). This approach follows the declarative programming paradigm, as the itemset mining problem is expressed by specifying a set of variables, their domains, and a number of constraints over them. It is then up to the constraint solving system to find solutions for the problem. In contrast to typical itemset mining methods that only address a certain variant of itemset mining problems (or have to be substantially modified to extend to another variant), adapting a constraint programming formulation of this problem to a new setting is often possible by adding or modifying a set of constraints. It has also been demonstrated that computational efficiency of this solving technique can be improved by introducing new means of data representation and novel propagation schemes.

Let $\mathcal{S}$ be the set of transactions in the databse $\mathcal{D}$ and $\mathcal{I}$ be the set of items. To model the problem of frequent itemset mining in constraint programming, we use a boolean variable I_i for each item and a boolean variable T_t for each transaction. A solution will correspond to a frequent itemset which is represented by I_i variables that are set to one. A variable T_t will be equal to one if the corresponding transaction contains this itemset. The following constraints establish the relationship between the variables I_i and T_t:

$$T_t = 1 \leftrightarrow \bigwedge_{i \in \mathcal{I}} (\mathcal{D}_{ti} = 1 \vee I_i = 0) \qquad \forall t \in \mathcal{S}$$

where the parameter $\mathcal{D}_{ti}$ is equal to one if transaction t contains item i and is equal to zero otherwise. The frequency constraint can be expressed as the following CP constraint:

$$\sum_{t \in \mathcal{S}} T_t \geq \theta$$

This formulation can be easily extended to model the problem of maximal frequent itemset mining. By adding the following constraints to the model, the set of solutions will only include the maximal itemsets:

$$I_i = 0 \rightarrow \sum_{t \in \mathcal{S}} T_t \mathcal{D}_{ti} < \theta \qquad \forall i \in \mathcal{I}$$

This concludes our review of probabilistic graphical models, constraint programming, integer programming and constraint-based pattern mining. In this review, we treated these domains independently, as they have been quite independent of each other in their development. In the consequent chapters we will see some of the connections between these domains and the ways in which they interact.

Part I

Probabilistic Models in Constraint Satisfaction and Optimization

Chapter 3

Constraint-Based Querying for Bayesian Network Exploration

In this chapter, we introduce a novel general framework and tool for answering exploratory queries over Bayesian networks. The framework is inspired by queries from the constraint-based mining literature designed for the exploratory analysis of data. This chapter is based on the following paper:

- Behrouz Babaki, Tias Guns, Siegfried Nijssen, and Luc De Raedt. "Constraint-based querying for bayesian network exploration". In Élisa Fromont, Tijl De Bie, and Matthijs van Leeuwen, editors, *Advances in Intel- ligent Data Analysis XIV - 14th International Symposium, IDA 2015, Saint Etienne, France, October 22-24, 2015, Proceedings,* volume 9385 of *Lecture Notes in Computer Science,* pages 13–24. Springer, 2015.

3.1 Introduction

Understanding a Bayesian network is not always easy. In particular users who are faced with a large network for the first time, or with networks that are dynamically updated when new data arrives, may not understand the knowledge encoded in such a network. It has been argued that BN's (especially those used

for diagnosis) should be extensively evaluated before being used in practice (Przytula et al. 2003).

While the Bayesian network literature already provides a set of queries and corresponding inference techniques that are helpful in gaining a better understanding of a network, most of the standard queries specify (and fix) the variables of interest, and then either ask for a most likely assignment to the variables or the computation of a particular probability.

This contrasts with common practice in the field of exploratory data mining, where one aims at understanding data by discovering and analyzing patterns. Since the seminal work on frequent itemset mining by Agrawal et al. (Agrawal, Imielinski, et al. 1993), numerous techniques for exploratory mining of patterns under constraints have been developed (Nijssen and Zimmermann 2014). The notions of frequency and pattern in constraint-based pattern mining actually correspond to the notions of probability and explanation in a Bayesian network. In pattern mining, one typically searches over a space of possible patterns. In Bayesian networks, this corresponds to searching over subsets of variables and their values. In this paper, we exploit the similarities between these two fields and introduce constraint-based queries for Bayesian networks.

The contribution of this chapter is three-fold. First, inspired by constraint-based mining, we introduce an expressive set of exploratory queries for Bayesian networks. Secondly, we identify how these queries can be expressed as constraints over the variables and joint distribution of the Bayesian network. Finally, we show how these constraints can be expressed as a generic constraint program, combining ideas from constraint programming, itemset mining and knowledge compilation, in particular CP4IM (Guns et al. 2011b) and arithmetic circuits (AC) (Darwiche 2003). Our method operates on the arithmetic circuit directly and can hence be applied to any graphical model that can be compiled into an AC. By doing so, we bridge the gap between constraint-based pattern mining and graphical models and contribute towards more intelligent analysis of Bayesian networks.

3.2 Examples of Bayesian Network Exploration

After introducing a BN pattern, we show examples of exploratory queries over a Bayesian network in an illustrative scenario.

Definition 4 (BN pattern)**.** *Consider a joint probability $P_{\mathcal{G}}$ represented by a Bayesian network $\mathcal{G}$. A pattern A over $P_{\mathcal{G}}$ is a partial assignment, that is, an assignment to a subset of the variables $\mathcal{X}$ in $\mathcal{G}$: $A = \{(X_1 = x_1), \ldots, (X_m = x_m)\}$, where the X_i are different variables and x_i is a possible value in $D(X_i)$.*

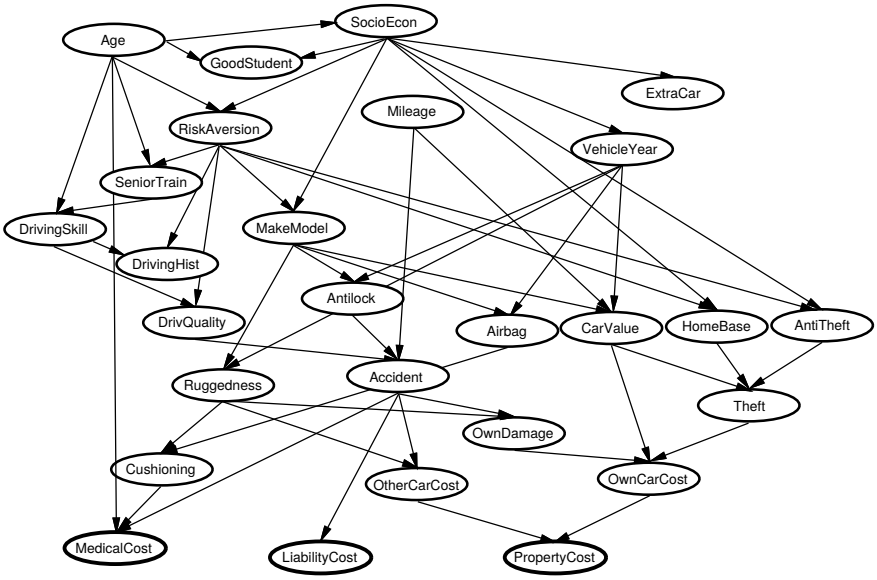


Figure 3.1: The car insurance network (Binder et al. 1997).

The probability of a pattern A, denoted by $P_{\mathcal{G}}(A)$, is $P((X_1 = x_1), \ldots, (X_m = x_m))$, that is, the marginal probability of the assignment. Our queries below will enumerate all satisfying BN patterns.

Example constraint-based queries

Assume the manager of a New York car insurance company has just obtained a Bayesian network that describes the factors influencing cost claims of customers, cf. the network in Figure 3.1. She wants to analyze the network to be able to assess costs, get more insight and provide recommendations to her personnel. In order to do so, she is interested in exploring patterns of interest in the network and poses a number of queries.

Q1. What are likely patterns given the evidence PropertyCost = Million? These claims impose high costs on the company. Using a minimum probability of 0.015, she obtains 12 patterns, most of which contain either **SeniorTrain = False** or **Theft = False**. She is not interested in these and excludes them while lowering the threshold in the next query.

Q2. What are likely patterns that do not contain SeniorTrain=False and Theft=False given the evidence PropertyCost = Million (with threshold

$\theta = 0.0105$)**?** She now gets 76 patterns, among which the pattern **A = {PropertyCost = Million, DrivingSkill = Substandard, DrivQuality = Poor, LiabilityCost = Thousand}**, which she finds interesting as it indicates a connection between high property cost, the driving capabilities of the customer, and the liability cost incurred. However, she wonders whether the pattern cannot be simplified.

Q3. Is there a simplification of pattern A with the same probability? She finds a variant of pattern **A** in which **DrivingSkill=Substandard** is removed, indicating that this assignment was implied and that there is some determinism in the network.

Now, turning her attention to the variable "Age", our manager wonders:

Q4. Are there any patterns that would allow to distinguish the age groups Adolescent and Senior? She queries for patterns that have widely varying conditional probabilities when conditioned on each of these. After excluding the variables **SeniorTrain** and **GoodStudent**, which she already knows about, one of the top patterns is **{RiskAversion = Cautious, OtherCarCost = Thousand}**. Indeed the probability of having a cautious personality and incurring low third-party costs is nearly 6 times higher in senior customers.

Finally, a machine learning expert suggests to use a network trained on the company data instead (a simple naïve Bayes model). She wonders:

Q5. What are the patterns that have different probabilities according to the original and learned network? It turns out that the pattern **{Airbag=False, AntiLock=False, VehicleYear=Older}** has the largest difference of probabilities, hence the naïve Bayes model ignores the well-known relation between these three variables (namely older cars are rarely equipped with these safety components).

3.3 BN query framework

We now formalize the queries above using constraints over patterns. Many other queries can be formulated this way, leading to a general querying framework.

Definition 5 (BN Pattern Query)**.** *Consider a joint probability distribution $P_{\mathcal{G}}$ represented by a Bayesian network $\mathcal{G}$. We denote the set of all patterns of $\mathcal{P}_{\mathcal{G}}$ by $\mathcal{I}$. A BN pattern query $\mathcal{Q}$ is a tuple $(\mathcal{P}_{\mathcal{G}}, \mathcal{C})$ where $\mathcal{C} : \mathcal{I} \to \{0, 1\}$ is a conjunction of constraints over a pattern. Pattern A is a solution for*

Q1: $probability(A, \mathcal{G}, \theta)$, $superset(A, \{\text{PropertyCost=Million}\})$
Q2: $probability(A, \mathcal{G}, \theta)$, $superset(A, \{\text{PropertyCost=Million}\})$, $exclude(A, \{\text{SeniorTrain, Theft}\})$
Q3: $maxprobability(A, \mathcal{G}, \theta')$, $free(A, \mathcal{G})$, $subset(A, \{\text{PropertyCost = Million, DrivingSkill = Substandard, DrivQuality = Poor, LiabilityCost =Thousand}\})$
Q4: $exclude(A, \{\text{SeniorTrain, GoodStudent}\})$, $ev\text{-}difference(A, \mathcal{G}, \{\text{Age=Adolescent}\}, \{\text{Age=Senior}\}, \beta)$
Q5: $difference(A, \mathcal{G}^1, \mathcal{G}^2, \beta)$

Table 3.1: The example queries expressed using constraints over pattern A.

$\mathcal{Q}$ *if* $\mathcal{C}(A) = 1$. *The result of a query consists of all patterns that satisfy the constraints.*

The queries used in the examples in Section 3.2 are given in Table 3.1. Most constraints have close counterparts in the constraint-based pattern mining literature. The main difference is that the notion of *(relative) frequency* of a pattern in a database is replaced by the *probability* of the pattern in the BN. The constraints and their definitions are listed in Table 3.2 and explained below.

Probability constraint. Query Q1 requires that the probability of a pattern A according to $P_{\mathcal{G}}$ should be larger than a threshold θ. We call this constraint *probability*$(A, \mathcal{G}, \theta)$ and a pattern that respects it θ*-probable*. This definition is similar to the definition of a *frequency* constraint in frequent pattern mining.

Sub/superset and exclusion constraints Query Q1 also requires that patterns include given assignments. We enforce this with a superset constraint. Similarly, we can use *exclude*(A, V) to exclude variables from the pattern as in query Q2. The definition is given in Table 3.2, where vars(A) are the variables occurring in A.

Note that a superset constraint is conceptually similar to adding *evidence* in Bayesian networks, only that in our setting the computed probabilities will need to be normalized by the probability of the evidence to obtain the conditional probability.

Freeness, maximality and closedness constraint Query Q3 requires that a pattern does not contain redundant variable assignments. This is similar to the well-studied problem of simplifying explanations by excluding irrelevant variables (Shimony 1993), e.g. because of deterministic relations between assignments (Druzdzel and Suermondt 1994). For pattern $A = B \cup C$ (where $B \cap C = \emptyset$),

code	mathematical notation	CP formulation
PRB	$P_{\mathcal{G}}(A) \geq \theta$	$\mathsf{F}_1 \geq \theta$
MXP	$P_{\mathcal{G}}(A) \leq \theta$	$\mathsf{F}_1 \leq \theta$
SBS	$A \subseteq B$	$\forall i : \mathsf{Q}_i \neq 0 \implies (X_i = \mathsf{Q}_i) \in B$
SPS	$B \subseteq A$	$\forall (X_i = x_i) \in B : \mathsf{Q}_i = x_i$
EXC	$B \cap \text{vars}(A) = \emptyset$	$\forall X_i \in \text{vars}(B) : \mathsf{Q}_i = 0$
FRE	$\forall (X = x) \in A : P_{\mathcal{G}}(A \backslash (X = x)) > P_{\mathcal{G}}(A)$	$\forall i : \mathsf{Q}_i \neq 0 \rightarrow \left(\sum_j \mathsf{D}_{i,j}\right) > \mathsf{F}_1$
MAX	$\forall X \notin \text{vars}(A), \forall x \in D(X) : P_{\mathcal{G}}(A \cup \{(X = x)\}) < \theta$	$\forall i : \mathsf{Q}_i = 0 \rightarrow \bigwedge_j (\mathsf{D}_{i,j} < \theta)$
CLS	$\forall X \notin \text{vars}(A), \forall x \in D(X) : P_{\mathcal{G}}(A \cup \{(X = x)\}) < P_{\mathcal{G}}(A)$	$\forall i : \mathsf{Q}_i = 0 \rightarrow \bigwedge_j (\mathsf{D}_{i,j} < \mathsf{F}_1)$
DIF	$\lvert P_{\mathcal{G}^a}(A) - P_{\mathcal{G}^b}(A) \rvert \geq \beta$	$\lvert {\mathsf{F}_1}^a - {\mathsf{F}_1}^b \rvert \geq \beta$
DDF	$\lvert P_{\mathcal{G}}(A) - r_{\mathcal{D}}(A) \rvert \geq \beta$	$\lvert \mathsf{F}_1 - \mathsf{R} \rvert \geq \beta$
VDF	$\lvert P_{\mathcal{G}}(A \cup B)/P_{\mathcal{G}}(B) - P_{\mathcal{G}}(A \cup C)/P_{\mathcal{G}}(C) \rvert \geq \beta$	$\lvert {\mathsf{F}_1}^a/c_a - {\mathsf{F}_1}^b/c_b \rvert \geq \beta$

Table 3.2: Constraints for BN pattern queries over patterns A, B, and C and network $\mathcal{G}$. Constraints are represented by three-letter codes PRB: *probability*$(A, \mathcal{G}, \theta)$; MXP: *maxprobability*$(A, \mathcal{G}, \theta)$; SBS: *subset*(A, B); SPS: *superset*(A, B); EXC: *exclude*(A, B); FRE: *free*$(A, \mathcal{G})$; MAX: *maximal*$(A, \mathcal{G}, \theta)$; CLS: *closed*$(A, \mathcal{G})$; DIF: *difference*$(A, \mathcal{G}^a, \mathcal{G}^b, \beta)$; DDF: *DB-difference*$(A, \mathcal{G}, \mathcal{D}, \beta)$; and VDF: *ev-difference*$(A, \mathcal{G}, B, C, \beta)$.

if variable assignments in B determine those in C, i.e., $P_{\mathcal{G}}(A) = P_{\mathcal{G}}(B)$, we consider those in the set C irrelevant. We call a pattern *free* if none of its assignments is irrelevant. This definition is similar to the definition of free patterns in data mining (Boulicaut and Jeudy 2001). In the presence of a *superset* constraint, the *free* constraint should only consider variables that are not required by the *superset* constraint.

Inspired by the related notions of maximality and closedness in frequent itemset mining, we introduce these for BN patterns too. They enforce that a pattern A does not have any superset that is θ-probable (i.e. *maximal*$(A, \mathcal{G}, \theta)$) or has the same probability as A (i.e. *closed*$(A, \mathcal{G})$).

Difference constraints Queries Q4 and Q5 both ask for patterns that demonstrate a difference between two probabilistic models. Let $P_{\mathcal{G}_1}(A)$ and $P_{\mathcal{G}_2}(A)$ be the probability of pattern A according to networks $\mathcal{G}_1$ and $\mathcal{G}_2$. The constraint *difference*$(A, \mathcal{G}_1, \mathcal{G}_2, \beta)$ requires that the difference of the probability of a pattern in these two networks is larger than β. In Q4, the two networks

are obtained by assigning a variable in the original network to different values (B={Age=Adolescent} and C={Age=Senior} respectively). This can be formulated over network $\mathcal{G}$ using the constraint *ev-difference(*$A, \mathcal{G}, B, C, \beta$*)*. This constraints compares the conditional probability of A given evidence B or C.

Another variation can be used for testing the correlations between a Bayesian network and an actual dataset. This constraint compares the probability of a pattern in network $\mathcal{G}$ with the relative frequency of the corresponding itemset in the database $\mathcal{D}$. We call this constraint *DB-difference(*$A, \mathcal{G}, \mathcal{D}, \beta$*)*.

3.4 Formulating BN Pattern Queries As Constraint Programming Problems

In the Bayesian network literature, typically algorithms that search in the space of assignments are developed for specific constraints and scoring functions, which limits their general applicability (see Section 7.2 for a discussion of related work). In data mining, a recent trend is the use of generic solvers for handling a wide range of constraints in a uniform way.

We observe that there is a relationship between itemsets and BN patterns, as each variable assignment $(X_i = x_i)$ can be seen as one *item*, and hence a BN pattern can be seen as an itemset. Using this insight, we adapt the constraint programming for itemset mining framework (Guns et al. 2011b) to reason over Bayesian networks. This framework has proven to support a wide range of constraints and exploratory queries over itemsets. Building on this framework, and hence the use of CP solvers, enable us to address a wide range of queries without the need to develop multiple specialized algorithms.

We first explain how Bayesian networks can be encoded in CP in the form of an arithmetic circuit. We then explain how the constraints identified in Table 3.2 can be expressed in this framework.

BN pattern in Constraint Programming (CP)

We can encode a BN pattern $A = \{(X_1 = x_1), \ldots, (X_m = x_m)\}$ in CP by introducing a CP variable Q_i for every network variable X_i. The domain of the CP variable Q_i consists of $|D(X_i)|+1$ values, where $D(X_i)$ is the set of possible values the BN variable X_i can take: value 0 to represent that X_i is not part of the pattern, e.g. it is marginalized over, and values $1 \ldots |D(X_i)|$ that each represent a possible assignment to the BN variable X_i.

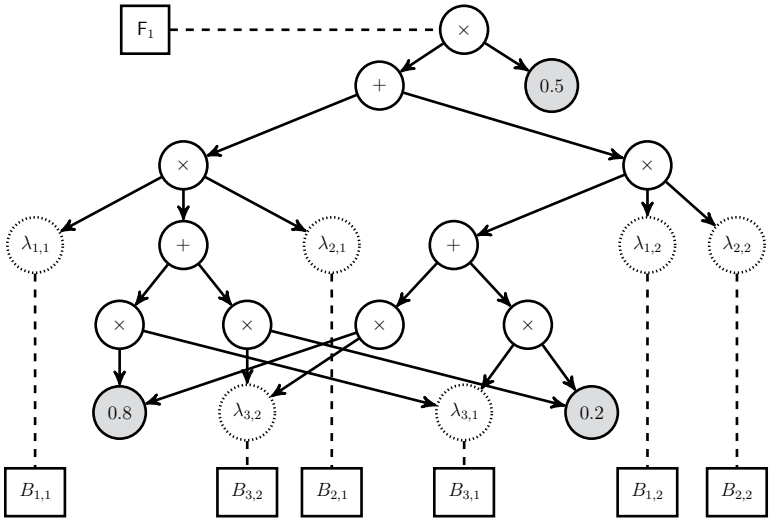


Figure 3.2: Arithmetic circuit for a BN with 3 variables with domain $\{1, 2\}$ with X_1 the parent of X_2 and X_3. Square boxes represent CP variables.

BN pattern queries in CP

Each of the constraints $\mathcal{C}$ of a BN pattern query $(\mathcal{P}_\mathcal{G}, \mathcal{C})$ can be formulated through CP constraints over the Q_i variables. We discuss this for each of the constraints in turn.

Probability constraint We will need to repeatedly compute the probability of a pattern, hence, we want this computation to be fast and ideally incremental. For this reason, we choose to first compile the BN into an Arithmetic Circuit (AC) (Darwiche 2003). Computing the probability of a partial assignment takes time polynomial to the size of the AC, though that size is exponential to the BN size in the worst case. Nevertheless, using ACs is generally recognized as one of the most effective techniques for exact computation of probabilities (Darwiche 2003), especially when doing so repeatedly.

An arbitrary AC can be encoded in CP: for each indicator variable $\lambda_{i,j}$ in the AC, we introduce a Boolean CP variable $\mathsf{B}_{i,j}$ (see Fig. 3.2); the relation between the indicator variables and the CP variables Q_i is then modeled by the following

constraints (recall that $\mathsf{Q}_i = 0$ means variable X_i is not in the pattern):

$$\mathsf{Q}_i = 0 \rightarrow \wedge_j \, (\mathsf{B}_{i,j} = 1) \qquad \forall i$$

$$\mathsf{Q}_i = k \rightarrow (\mathsf{B}_{i,k} = 1) \wedge (\wedge_{j \neq k} (\mathsf{B}_{i,j} = 0)) \qquad \forall i, \forall k \neq 0$$

We then introduce real-valued variable P, which will represent the computed probability. For this, we introduce an auxiliary real-valued variable F_v for each node in the circuit (round circles in Fig. 3.2). Assume each node has a unique identifier v, with the root node having identifier 1. Leaf nodes are either constants or indicator variables. The constants assign their corresponding F_v variable to a fixed value. For the indicator variables $\lambda_{i,j}$, the corresponding F_v variables are *channeled* to their Boolean counterparts $\mathsf{B}_{i,j}$ meaning they must take the same value (either 0 or 1). The internal nodes are then simply encoded by their operation, namely constraint $\mathsf{F}_v = \prod_{w \in \mathrm{Ch}(v)} \mathsf{F}_w$ for product nodes and constraint $\mathsf{F}_v = \sum_{w \in \mathrm{Ch}(v)} \mathsf{F}_w$ for sum nodes, where $\mathrm{Ch}(v)$ are the identifiers of the children of node v in the AC.

Because of these constraints, when all Q_i (and hence B_i) variables are assigned, each F_v represents the value of that node of the AC, and the root node F_1 is the probability of the BN pattern. F_1 can then be used in a minimum probability constraint, see Table 3.2, right column.

Subset, superset and exclusion constraints Including evidence and excluding assignments in the pattern is done by constraining the relevant Q_i variables appropriately, as indicated in Table 3.2.

Freeness constraint To enforce this constraint, as explained in Section 3.3, we need to reason over the probability of subsets of a pattern. To do so, we use the observation that for an assignment $(X_i = k) \in A$: $P_{\mathcal{G}}(A \backslash \{(X_i = k)\}) = \sum_j P_{\mathcal{G}}((A \backslash \{(X_i = k)\}) \cup \{(X_i = j)\})$. Fortunately, using ACs we can efficiently compute these terms, as they correspond to *derivatives* of the function f encoded by the AC (Darwiche 2003). The latter work shows that for partial assignment A we have $P_{\mathcal{G}}((A \backslash \{(X_i = k)\}) \cup \{(X_i = j)\}) = \frac{\partial f}{\partial \lambda_{i,j}}(A)$. It was also shown that this can be computed for all nodes (and hence variables X) simultaneously using the derivatives of its parents in the AC, together with the values that we store in F_v variables.

To compute these derivatives, we introduce a real-valued CP variable D_v for every node v in the circuit. The value of D_v's corresponding to leaves $\lambda_{i,j}$, denoted by $\mathsf{D}_{i,j}$ for ease of notation, will represent the derivative of AC w.r.t $\lambda_{i,j}$: $\forall i, j \; \mathsf{D}_{i,j} = \frac{\partial f}{\partial \lambda_{i,j}}(A)$. Hence $\mathsf{D}_{i,j} = P_{\mathcal{G}}((A \backslash \{(X_i = k)\}) \cup \{(X_i = j)\})$.

Following the formulation in (Darwiche 2003), the constraints below encode the computation of the D variables, where we denote by $\mathrm{Pa}^{+}(v)$ the identifiers of summation parents and by $\mathrm{Pa}^{*}(v)$ those of multiplication parents;

$$\mathsf{D}_v = \sum_{w \in \mathrm{Pa}^{+}(v)} \mathsf{D}_w + \sum_{w \in \mathrm{Pa}^{*}(v)} (\mathsf{D}_w \prod_{\substack{v' \in Ch(w) \\ v' \neq v}} \mathsf{F}_v) \qquad \forall v$$

$$\mathsf{D}_1 = 1$$

To formulate the *free* constraint from Table 3.2 over the CP variables, we use the fact that given $(X_i = k) \in A$: $P_{\mathcal{G}}(A \backslash (X_i = k)) = \sum_j \mathsf{D}_{i,j}$ and that $P_{\mathcal{G}}(A) = \mathsf{F}_1$.

Maximality and closedness constraints can be formulated using the same building blocks (c.f. table 3.2).

Difference constraints Comparing the probability of two networks over the same variables can be done by encoding the two ACs and formulating a mathematical constraint over the respective F_1 root node variables (Table 3.2).

Using CP allows us to easily mix different problems, such as combining the constraints of itemset mining in databases and BN's in a single CP model. The variable F_1 can be computed as before, while the relative frequency of a database over the same variables can be computed using a constraint programming for itemset mining formulation (Guns et al. 2011b). In Table 3.2 we materialize the relative frequency through a CP variable R.

As we have shown, many constraints over the pattern and the network can be readily formulated in CP. Furthermore, as these are standard CP constraints, existing CP solvers can be used to enumerate the satisfying BN patterns.

3.5 Experiments

We used the *ACE*[1] compiler (version 2) for generating arithmetic circuits from Bayesian networks. The networks were compiled with parameters "`-noTabular -cd06 -dtBnMinfill`". We used the *Gecode* [2] CP solver version 4.2.1. Experiments were run on Linux PCs with Intel 2.83GHz processors and 8GB of RAM.

[1] http://reasoning.cs.ucla.edu/ace/
[2] http://www.gecode.org

Network	#BN-n	c.Time(s)	#AC-n	#AC-e	θ	#Sols	s.Time(s)
HeparII	70	0.701	6963	13272	0.9	664	9.96
					0.8	24025	341.83
Win95pts	76	0.528	2786	6184	0.99	65	0.61
					0.95	214645	444.1
Insurance	27	0.374	34742	113788	0.9	12	2.76
					0.4	6662	383.66

Table 3.3: Probability queries over three benchmarks network, with **#BN-n**: number of BN nodes; **c.Time**: compilation time; **#AC-n/e**: number of AC nodes/edges; θ: probability threshold; **#Sols**: number of solutions, and **s.Time**: solving time.

Execution times for example queries

To give an indication of execution times, we report the runtimes for the example queries of section 3.2 in Table 3.4a. The value of β for queries Q4 and Q5 was 0.08 and 0.25, respectively. The compilation time (not included in the reported runtimes) was 0.374 seconds.

To investigate the influence of size of BN and AC, we ran a simple query with only a $probability(A, \mathcal{G}, \theta)$ constraint on three benchmark networks[3]. Table 3.3 reports BN and AC size, θ threshold and runtimes. AC compilation time is small. Observe that in Table 3.3 the two larger networks have smaller AC's, because of their other structural properties (see (Darwiche 2003) for more details). While bigger ACs require more runtime, the number of solutions has a major impact on runtime too. This can be controlled up to some extent by adding extra constraints.

Comparison with sampling

An obvious alternative to our proposed method for executing itemset queries is to first sample a database from the joint distribution and then perform constraint-based itemset mining queries on the sampled database. Using this approach, one can execute the BN pattern queries using a constraint-based itemset mining system such as (Guns et al. 2011b).

We investigate how this compares to our proposed method. We used two BNs: the first was the *insurance* network, which we will call BN1. The network BN2 is a naïve Bayes version of BN1 (With **PropertyCost** as root, and all non-cost observed variables as children) which we trained on 10000 samples from BN1. Compilation time for BN1 was 0.374 and 0.212 seconds for BN2. We then sampled a database of size 500 from BN2, which we call DB2.

[3]available at http://www.bnlearn.com/bnrepository/

Query	Q1	Q2	Q3
Time(s)	1.63	11.7	11.67
Query	Q4	Q5	
Time(s)	58.41	12.11	

(a)

#Samples	Precision	Recall	Time(s)
100	0.39	0.76	7.59
1000	0.73	0.94	20.08
10000	0.97	0.93	375.95

(b)

Table 3.4: (a) Execution times of example queries, and (b) Quality of results of sampling method as compared against the solutions of exact method.

In the approximate method, we sampled databases of varying sizes from BN1. We then searched for itemsets for which the relative frequency in the database and DB2 had a difference larger than 0.1. Table 3.4b presents the precision and recall of BN patterns found by the approximate method, compared to those found by our exact method. The results indicate that for a decent approximation, one needs to sample a large database which in turn leads to high computational costs. In comparison, the runtime of the exact method was 5.63 seconds.

3.6 Related work

Much attention in the Bayesian network literature has gone to the problem of finding explanations given some evidence. These *explanation queries* typically use a scoring function to find the best explanation. In contrast to queries like MAP and MPE, we do *not* fix which variables must be in or not in the pattern, instead we conceptually search over all possible marginalizations. There are other explanation queries that share this feature. These typically use specific scoring functions, such as the generalized Bayes factor of (C. Yuan, Lim, et al. 2011). The explanation queries are constrained optimisation problems instead of enumeration problems. Our framework on the other hand is made for exploration queries and enumerates all satisfying BN patterns instead of computing the 'optimal' one.

There is also a body of work on discarding irrelevant variables from explanations (Shimony 1993; Campos et al. 2001; C. Yuan, Lim, et al. 2011; Kwisthout 2013), as the *free* constraint does in our framework. In (Campos et al. 2001) each explanation found by a K-MPE algorithm is simplified by removing assignments that are considered irrelevant; (Kwisthout 2013) makes a trade-off between high probability and specificity. (Shimony 1993) proposes a definition for *relevance* and gives an algorithm that excludes irrelevant variables from the MAP assignments. This is a specific optimization query which is solved by a best-first-search algorithm.

Related to discriminating a BN network from a database, in (Jaroszewicz et al.

2009) the authors search for subsets of variables rather than partial assignments. These *attribute sets* are then used to modify the BN to better reflects the correlations present in the data. In other studies, a Bayesian network is used to *filter* itemsets or association rules found in a database. In (Fauré et al. 2006), first an itemset mining algorithm is applied to a database to find a number of association rules, and then these rules are scored using the probability in the Bayesian and the concept of D-separation. In (Malhas and Aghbari 2009) the itemsets found by the well-known *apriori* algorithm are scored according to a Bayesian network, and the itemsets and attribute sets with highest scores are obtained in a post-processing step. The main difference with the discriminative setting considered in our work is that we compare patterns in the database and the network during search instead of post-processing them.

Our framework combines constraints with probabilistic computations. In similar spirit, there has been work on combining (deterministic) constraint networks with probabilistic networks (Mateescu and Dechter 2008). The main difference is that in the resulting networks, all satisfying assignments are aggregated to compute a single probability value; on the other hand, we enumerate all possible *partial* assignments and compute their (marginal) probability.

3.7 Conclusions

We have investigated the problem of *exploring* Bayesian networks by querying for BN patterns (partial assignments) under constraints. The work is inspired by all the work on exploring data using constraint-based pattern mining techniques. We have shown that similar queries and constraints as used in the constraint-based pattern mining community can be used. This results in novel querying abilities for BNs. The proposed execution strategy is to compile the BN into an arithmetic circuit, and formulate and reason over that in a constraint programming framework. Such an approach supports a wide range of queries and constraints in a flexible and declarative manner.

Our work currently focusses on enumeration queries, as is typical in pattern mining. However, it could also be used in an optimisation setting over a scoring function, where its generality would allow one to add arbitrary constraints on top of the scoring function. In future work, the approach could also be adapted to problems beyond enumerating BN pattern queries, such as verifying monotonicity of Bayesian networks (Rietbergen et al. 2014) or computing same-decision probability (Chen et al. 2014). Our method may also be valuable for mining patterns over data, when evaluating the interestingness of the patterns using a BN (in our case, *during* search). Given the generality of the method, efficiency can be a concern though. Efficiency could be improved by using *global*

constraints that can reason over the AC more efficiently, instead of using a decomposition over auxiliary variables F.

Chapter 4

Stochastic Constraint Programming with And-Or Branch-and-Bound

Complex multi-stage decision making problems often involve uncertainty, for example, regarding demand or processing times. Stochastic constraint programming was proposed as a way to formulate and solve such decision problems, involving arbitrary constraints over both decision and random variables. What stochastic constraint programming still lacks is support for the use of factorized probabilistic models that are popular in the graphical model community. In this chapter we show how a state-of-the-art probabilistic inference engine can be integrated into standard constraint solvers. This chapter is based on the following publication:

- Behrouz Babaki, Tias Guns, and Luc De Raedt. "Stochastic constraint programming with and-or branch-and-bound". In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017,* pages 539–545. ijcai.org, 2017.

4.1 Introduction

Increasingly, complex decision making requires one to make *decisions* under *constraints* while taking into account the *uncertainty* of the environment. Each of these aspects has intensively been studied by different communities within artificial intelligence. Indeed, constraint programming has focused on solving constraint satisfaction problems and making decisions while the field uncertainty in artificial intelligence is concerned with probabilistic graphical models and inference. For each of these problems, advanced solutions have been developed and solvers exist that can tackle substantial problems. But today, there is a growing awareness that in many real-life applications, these aspects cannot be addressed in isolation, but rather need to be tackled by an integrated approach. Stochastic constraint programming (Walsh 2002; A. Tarim et al. 2006) covers all three aspects as it extends constraint programming with decision making under uncertainty. However, such methods do not yet support standard probabilistic techniques from the graphical model community (Koller and N. Friedman 2009). It is well-known in probabilistic graphical models that factorizing the joint probability distribution is beneficial for modeling, inference and for learning (Koller and N. Friedman 2009). Stochastic constraint programming currently uses trivial factorizations, assuming either that all random variables are marginally independent (Walsh 2002), or using the joint as the only factor (A. Tarim et al. 2006). The latter corresponds to enumerating all possible worlds, also called scenarios. On the other hand, (Mateescu and Dechter 2008) have integrated constraint programming and probabilistic graphical models, but do not deal with decisions and utilities; and influence diagrams (F. Jensen et al. 1994) integrate probabilistic graphical models with decision theory, but do not handle constraints.

Our contribution is as follows:

- We support stochastic constraint programming with factorized joint probability distributions (as in Bayesian networks) and integrate state-of-the-art inference engines for such graphical models.
- We use a generic constraint solver both for the deterministic constraints and for doing constrained branch-and-bound search over an and-or tree.
- We develop and exploit a novel bound for *expected* utility in the search. The key is that we use a probabilistic inference engine to compute marginal probabilities and interval arithmetic for the utility.

We now introduce the problem and review the two standard approaches, after which we explain and evaluate our method.

4.2 Stochastic Constraint Programming

We consider multi-stage decision problems where a number of decisions can be taken (such as the production amount) after which external factors are observed (such as the demand) followed by new decisions etc. The goal is to assign in a stage-wise manner the decision variables so that the expected utility over all possible instantiations of the random variables is maximized. To model the stochastic aspect of the external factors we use random variables with a (joint) discrete probability distribution P.

Example 6. *Consider a simple stochastic production problem from (Walsh 2002) where each stage corresponds to one quarter of the year, and the decisions V_i are how many books to produce at the start of quarter i, and the random variables S_i represent how many books were sold at the end of quarter i. The company is conservative so shortages are not allowed, yet the goal is to minimize the sum of surpluses in each quarter due to stocking costs.*

More formally for the 2 stage variant:

$$\text{minimize} \quad \min_{V_1} \sum_{S_1} \min_{V_2} \sum_{S_2} P(\mathcal{S}) \times U(\mathcal{V}, \mathcal{S})$$

$$s.t. \quad \sum_{j=1}^{i} (V_j - S_j) \geq 0 \qquad \forall i = 1..2$$

where $U(\mathcal{V}, \mathcal{S}) = V_1 - S_1 + V_2 - S_2$

Factored distributions

We will assume that the probability distribution P is specified as a factored distribution, that is, the probability can be computed as a product of individual factors (Koller and N. Friedman 2009). One popular such representation is a Bayesian network.

An example Bayesian network is shown in Figure 4.1, left. It has two observed variables (factors) S_1 and S_2, which would typically be two random variables present in the stochastic problem formulation. The Bayesian network is a hidden Markov model with 2 hidden variables (factors) where the probability of observation is influenced by the hidden variables (e.g. market sentiment), and the second hidden variable is influenced by the hidden variable of the previous stage. Such a rich structure models complex interactions, and supports learning them from observations.

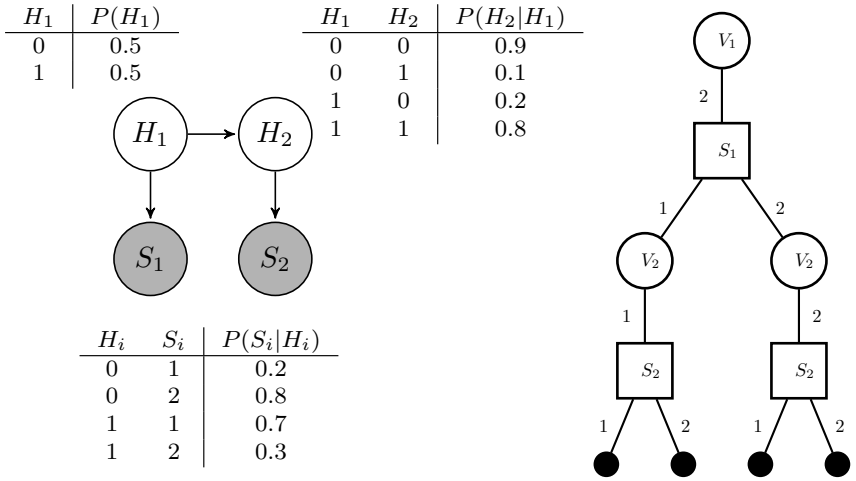


Figure 4.1: Left: Bayesian network with 2 observed and 2 hidden variables. Right: a policy tree for Example 6 with $\mathcal{D}(V_1) = \mathcal{D}(S_1) = \mathcal{D}(V_2) = \mathcal{D}(S_2) = \{1, 2\}$.

Problem Description

We define a multi-stage *Factored Stochastic Constraint Problem* (FSCP) as a 7-tuple $\mathcal{P} = \langle \mathcal{V}, \mathcal{S}, \mathcal{D}, \mathcal{P}, U, \mathcal{C}, \prec \rangle$ where:

- $\mathcal{V}$ and $\mathcal{S}$ are decision variables and random (*s*tochastic) variables, respectively;
- $\mathcal{D}$ is the domain of variables in $\mathcal{V} \cup \mathcal{S}$, namely a mapping from variable to the set of values it can take;
- $\mathcal{P}$ is a factored discrete distribution over $\mathcal{S}$, i.e. $P(\mathcal{S}) = \prod_{\mathcal{S}_i \subset \mathcal{S}} \phi(\mathcal{S}_i)$ with each factor $\phi(\mathcal{S}_i)$ over a subset of $\mathcal{S}$. As in previous works, we assume that the decision variables have no measurable impact on the probability distribution.
- $U(\mathcal{V}, \mathcal{S})$ is a function that computes the *utility* of an instantiation of the decision and random variables.
- $\mathcal{C}$ is a set of deterministic constraints. Each constraint is specified over a non-empty subset of $\mathcal{V}$ and a (possibly empty) subset of $\mathcal{S}$.
- $\prec$ is a partial ordering over $\mathcal{V} \cup \mathcal{S}$ that orders the variables by stage, and within each stage the decision variables before the random variables.

For notational convenience and without loss of generality we will assume one decision variable V_i and one random variable S_i per stage i: $V_1 \prec S_1 \prec \ldots \prec V_T \prec S_T$.

The objective is to maximize (or similarly minimize) the *expected utility* of the multi-stage problem according to $\prec$:

$$\max_{V_1} \sum_{S_1} \max_{V_2} \sum_{S_2} \ldots \max_{V_T} \sum_{S_T} P(S_1, \ldots, S_T) \times U(V_1, \ldots, V_T, S_1, \ldots, S_T) \tag{4.1}$$

where we note that sum and max are not transitive and hence can not be reordered in this formula. Constraints can range over random variables but are deterministic: they must be satisfied for all possible (non-0 probability) instantiations of the random variables.

An assignment to the variables $\mathcal{V} \cup \mathcal{S}$ that satisfies all constraints is not a solution to the FSCP, rather, it holds only with probability $P(\mathcal{S})$ and hence contributes $P(\mathcal{S}) * u(\mathcal{V}, S)$ to the expected utility. Indeed, the solution is a *policy tree* (Walsh 2002) where each node corresponds to a variable and for each path in the tree the variables satisfy the ordering $\prec$. Each decision variable V_i has just one child (corresponding to an assignment of this variable) and each random variable $\mathcal{S}_i$ has a child for *each* value in its domain. Figure 4.1 (right) shows a policy tree for the problem of Example 6, where 1 or 2 thousand books can be sold per stage.

An *optimal* policy tree is a policy tree where each decision variable in the tree is assigned to a value such that Eq. (4.1) is maximized, while always satisfying all constraints.

Scenario-based Search

One approach (A. Tarim et al. 2006; Hemmi et al. 2017) is to ground out each of the possible worlds and compute their probability, and at the same time flatten the policy tree by creating copies of each decision variable for all instantiations of the random variables preceding it. One assignment to this set of decision variables then corresponds to a policy tree.

For example for the policy tree in Figure 4.1 (right) we would create a decision variable for every decision node in the tree, so 3 variables in total: one for V_1 and two different ones for V_2, corresponding to cases $S_1 = 1$ and $S_1 = 2$.

Constraints are added over these decision variables as needed, and the expected utility function becomes a linear sum over all possible worlds $\sum_s P(s) * u(\mathcal{V}, s)$

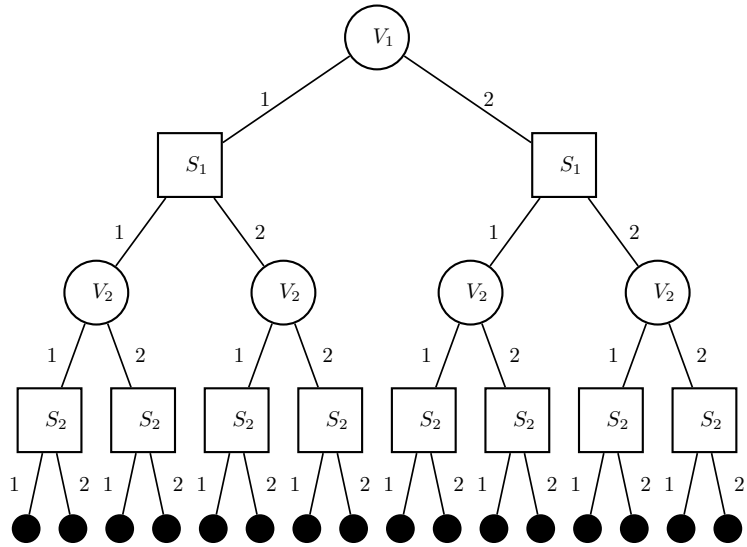


Figure 4.2: The And-Or search tree for the problem of Example 6.

with s a possible world (also called scenario). For the example in Figure 4.1 (right), there would be $2 * 2$ scenarios corresponding to the possible worlds.

An obvious downside of this approach is that the number of decision variables and the number of scenarios grows exponentially in the number of random variables, with the base of the exponent determined by the number of possible values for the random variables.

And-Or Search

Initially, a simple And-Or search algorithm (plain backtracking and forward checking) for stochastic constraint programming was proposed (Walsh 2002).

The And-Or search tree has two types of internal nodes: the AND nodes correspond to random variables, and the OR nodes correspond to the decision variables. The leaves do not correspond to a variable. An outgoing edge from an internal node represents the assignment of a value to the variable associated with that node. Figure 4.2 shows the And-Or search tree for the problem of Example 6.

Every path from the root to a node corresponds to a partial assignment to $\mathcal{V} \cup \mathcal{S}$, and must respect the ordering $\prec$. Given an assignment, we denote by

v_i the value of variable $V_i \in \mathcal{V}$ in the assignment. We label each node n by the partial assignment $(v_1, s_1, \ldots, v_k, s_k)$ represented by the path from the root to this node and denote it by $label(n)$. The *value* of a labeled node n is then defined as follows:

$$val(v_1,s_1,\ldots,v_k,s_k) = \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} \tag{4.2}$$

$$P(s_{1:k}, S_{k+1:T}) \times U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$$

where we use $v_{1:k}$ as shorthand for $(v_1, \ldots, v_k)$. The value of a label $(v_1, s_1, \ldots, v_k)$ ending in a decision variable is defined analogously. The notation follows (F. Jensen et al. 1994).

Observe how the expression in Eq. (4.1) corresponds to the above value function when none of the variables are assigned, that is, the value of the root node of the And-Or tree.

The value of any node n in the And-Or tree can be computed recursively as follows:

1. The value of a leaf with $label(n) = (v_{1:T}, s_{1:T})$ is $P(s_{1:T}) \times U(v_{1:T}, s_{1:T})$
2. If n corresponds to a random variable, then $val(label(n)) = \sum_{n' \in \text{children}(n)} val(label(n'))$
3. If n corresponds to a decision variable, then $val(label(n)) = \max_{n' \in \text{children}(n)} val(label(n'))$

A generic depth-first search procedure to recursively compute this function, while also ensuring satisfaction of all constraints in non-0 probability worlds, is shown in Algorithm 2. The symbol $\mathcal{D}$ represents the *domain*, that is, a mapping from variables to the values they can take.

First, the AndOr() procedure on line 2 does *propagation*, which is the act of removing those values from the domain that would violate a constraint. If all variables are assigned (their domain has size 1), then the value of this leaf is computed and returned. Next, on line 6, the variable to expand in this node is selected in such a way that the order $\prec$ is respected and the value for each of the children in the domain is recursively computed. In case of an AND node, first one has to verify that all children (not just those with a value in the domain) were visited and did not fail (line 13) because all possible worlds must be possible in a policy tree. If so, the sum of child values is returned. For OR nodes the maximum of the child values is returned.

Algorithm 2 And-Or search over domain $\mathcal{D}'$ following $\prec$

```
1: procedure AndOr(D')
2:     if propagate(D') == failure and probability > 0 then return failure
3:     if ∀x ∈ V ∪ S : |D'(x)|= 1 then                          ▷ In leaf
4:         return val(label(D'))
5:     end if
6:     Select unassigned variable X according to ≺
7:                                  ▷ Expand this node by assigning values to X
8:     for x ∈ D'(X) do                          ▷ For all children of this node
9:         D'' := D' and set D''(X) := {x}
10:        childval_x := AndOr(D'')
11:    end for
12:    if X ∈ S then                                              ▷ AND node
13:        if one of the children failed then return failure
14:        else return Σ_{x∈D(X)} childval_x
15:    else                                                       ▷ OR node
16:        return max_{x∈D(X)} childval_x
17:    end if
18: end procedure
```

4.3 Method: branch-and-bound And-Or search

We improve on the above And-Or search in the following two ways, which requires the probabilities of partial assignments:

- before exploring a node in the tree, we verify that the probability of the assignment to the stochastic variables explored so far (a partial assignment) is not 0. If it is, then all leaves that are descendants of this node will have 0 probability and hence the value of this node will always be 0 and it should not be explored. This was studied in (Qi and Poole 1995) where it was shown that even a naive and-or search can be sped up significantly in case of determinism (0 probabilities) in the graphical model.
- we compute upper bounds on the expected utility achievable by a node in the and-or tree to prune the search.

Querying the probability of partial assignments

The probability of a partial assignment can be obtained through marginalization; this has been studied for many years by the uncertainty in AI community (Pearl

1989). Given a distribution over T variables, the marginal probability of a subset of k variables is obtained by marginalizing out all other variables: $P(S_1, ..., S_k) = \sum_{S_{k+1}} \dots \sum_{S_T} P(S_1, ..., S_T)$.

Probabilistic inference methods can efficiently exploit structure and determinism in the factored distribution when queried for a marginal probability. To take advantage of this during search, we integrate an existing query inference engine into our approach.

The characteristics of our queries are that: 1) we will query the engine many times during search (at every node in the tree); 2) our queries will always contain random variables satisfying the order $\prec$, following the depth-first search; 3) we wish to obtain the partial probabilities of all possible values of a random variable at once. Motivated by this, we chose the ACE engine (Chavira and Darwiche 2008) as query engine, because 1) it first compiles the Bayesian network into an arithmetic circuit (AC). While this circuit may have worst-case exponential size, once the AC is compiled computing a (marginal) probability takes time linear in the size of the AC. In practice, the size of the AC is often reasonably small making this approach very attractive to many applications (Chavira and Darwiche 2005); 2) the ACE engine allows to incrementally *commit* and *retract* assignments which fits the depth-first search procedure well and prevents the engine from traversing the AC from scratch each time; 3) it can return, at low extra cost, the probabilities for all values of a random variable. For more details on the inference engine, see (Darwiche 2003).

Interval arithmetic for the utility

The computation of our novel bounds during the branch-and-bound And-Or search relies on the following key ideas: to query the inference engine for an upper bound on the probability of a partial assignment, to use *interval arithmetic* (Moore 1966) to get an upper-bound on the utility of a partial assignment and to combine this into an upper bound on the value of any node in the And-Or tree.

The basic idea of interval arithmetic is that, given an equation over intervals, using just the upper and lower bounds one can derive an upper and lower bound on the outcome of the equation (Moore 1995). For example given $f(X, Y) = 3X - Y$, then using interval arithmetic we can derive that $f([X_{min}, X_{max}], [Y_{min}, Y_{max}]) = [3X_{min} - Y_{max}, 3X_{max} - Y_{min}]$ where $[\cdot, \cdot]$ represents the minimum and maximum value of an interval. Interval arithmetic is a technique that constraint solvers often use internally to obtain bounds-consistency (Choi et al. 2006).

Given a partial assignment $(v_1, s_1, \ldots, v_k, s_k)$, we wish to derive an upper bound on the value of $U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$ over all possible assignments to $V_{k+1:T}$ and $S_{k+1:T}$. We define the upper bound on the utility as: $\bar{U}(v_{1:k}, s_{1:k}) = \max_{V_{k+1}} \max_{S_{k+1}} \ldots \max_{S_T} U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T})$. The bound is obtained by applying interval arithmetic to U with interval $[v_i, v_i]$ for each assigned variable and $[min(\mathcal{D}(V_i)), max(\mathcal{D}(V_i))]$ for each unassigned variable V_i; likewise for the S_i. For simple utilities, as is the case here, this interval bound computation can be provided by hand.

Shallow upper bound

Theorem 1. *For each partial assignment* $label(n) = (v_1, s_1, \ldots, v_k, s_k)$*, we have:*

$$val(v_1, s_1, \ldots, v_k, s_k) \leq P(s_{1:k}) \times \bar{U}(v_{1:k}, s_{1:k}) \tag{4.3}$$

Proof.

$$\begin{aligned}
&\max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&\qquad\qquad \times U(v_{1:k}, V_{k+1:T}, s_{1:k}, S_{k+1:T}) \\
&\leq \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \times \bar{U}(v_{1:k}, s_{1:k}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times \max_{V_{k+1}} \sum_{S_{k+1}} \ldots \max_{V_T} \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times \sum_{S_{k+1}} \ldots \sum_{S_T} P(s_{1:k}, S_{k+1:T}) \\
&= \bar{U}(v_{1:k}, s_{1:k}) \times P(s_{1:k})
\end{aligned}$$

□

Hence, by efficiently querying for the probability $P(s_{1:k})$ and computing $\bar{U}(v_{1:k}, s_{1:k})$, we obtain an upper bound on the value of this node.

Pruning OR nodes. This upper bound can be used to prune children of an OR node, as only the child with the maximum value is sought. In the search, every OR node will store as lower bound the value of its best child so far. If the

upper bound of the next child to explore is below this lower bound, the child does not need to be visited and can hence be pruned.

Pruning AND nodes. We observed that sometimes it is possible to prune an AND node before all its children have been explored. In those cases, the values of the children already visited are too low for the AND node to improve its closest parent OR node.

Assume that every AND node can receive from its parent what the minimum value is that it should achieve, that is, its lower bound LB. Let 'Ch' be the set of children of an AND node and let 'Pre' be the children already explored during search. Then, the following needs to hold:

$$LB \leq \sum_{i \in \text{Ch}} val(label(i)) \tag{4.4}$$

$$\rightarrow LB \leq \sum_{i \in \text{Pre}} val(label(i)) + \sum_{i \in \text{Ch} \setminus \text{Pre}} UB(i) \tag{4.5}$$

where $UB(i)$ is the upper bound on child i computed using Theorem 1. Hence, after exploring a child of an AND node, we can verify whether Eq. (4.5) holds and if it doesn't we do not need to visit the remaining children.

For this to work, all nodes must be able to pass a lower bound LB to its children. For an OR node, the lower bound of a child is simply the lower bound of the OR node itself. For AND nodes, we can derive from Eq. (4.5) the following lower bound on an unvisited child c:

$$LB - \sum_{i \in \text{Pre}} val(label(i)) - \sum_{i \in \text{Ch} \setminus (\text{Pre} \cup \{c\})} UB(i) \leq val(label(c))$$

hence the value of child node c needs to be larger than the left-hand side of this equation.

Deep upper bound

A tighter upper bound on the value of a node can be computed by realizing that the summation of an AND node S_k corresponds to a weighted sum of the children's value, weighted by their (conditional) probability. Hence, we can obtain a tighter bound by computing the probability and upper bound on the utility for each child of this AND node separately:

$$val(v_1, s_1, \ldots, v_k) \leq \sum_{S_k} P(s_{1:k-1}, S_k) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_k)$$

The same reasoning is valid for any sequence of unassigned random variables, that is, up to any depth:

Theorem 2. *Given the partial assignment* $(v_1, s_1, \ldots, v_k)$ *and an arbitrary* depth d *such that* $k + d \leq T$*, we have:*

$$val(v_1, s_1, \ldots, v_k)$$

$$\leq \sum_{S_k} \cdots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})$$

Proof.

$$\sum_{S_k} \max_{V_{k+1}} \cdots \sum_{S_T} P(s_{1:k-1}, S_{k:T}) \times U(v_{1:k}, V_{k+1:T}, s_{1:k-1}, S_{k:T})$$

$$\leq \sum_{S_k} \max_{V_{k+1}} \cdots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, V_{k+1:k+d}, s_{1:k-1}, S_{k:k+d})$$

$$\leq \sum_{S_k} \max_{V_{k+1}} \cdots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})$$

$$= \sum_{S_k} \sum_{S_{k+1}} \cdots \sum_{S_{k+d}} P(s_{1:k-1}, S_{k:k+d}) \times \bar{U}(v_{1:k}, s_{1:k-1}, S_{k:k+d})$$

the last step is possible because the remaining P and $\bar{U}$ computation is independent of the assignments to $V_{k+1:T}$, as it was taken out of $\bar{U}$ in the step before. □

This bound can be recursively computed by a simple depth-first search over the assignments to the next d unassigned random variables. At depth d of the recursive computation, the probability $P(s_{1:k-1}, S_{k:k+d})$ is queried and the upper bound on the utility is computed. Even if all probabilities of all random variables are equal, this bound will be tighter than the shallow bound thanks to the utility being computed for the actual value of the random variables explored. The complexity of this computation is exponential with respect to the depth.

4.4 And-Or search in a constraint solver

A strong argument in favor of scenario-based search in (A. Tarim et al. 2006) is the possibility to use any existing constraint solver, and hence the expressivity

and the constraints available in such solvers. We also support this argument and use a generic constraint solver for our And-Or search and the constraints. The key issue is that such a solver performs depth-first backtracking search but is oblivious to nodes being AND or OR nodes. What it considers a *solution*, namely an assignment to all of the variables, is just a leaf in the And-Or tree.

However this is not a fundamental issue, as constraint solvers are inherently *modular* depth-first search engines. To obtain a correctly working And-Or search, we added three modules to a constraint solver, which can be added to most if not all modern CP solvers:

1. a *global* constraint that verifies whether no values are removed from the domain of random variables, except by the search method, and otherwise fails because the constraints should be satisfied in all possible worlds;
2. a variable and value ordering module that respects $\prec$, that will fail the remaining children of an AND node if one of its children failed, and that also computes and prunes using the upper bounds, as well as pushing the lower bound to the child nodes;
3. a module that is called each time a complete assignment is found, and that updates the expected utility values in its ancestor nodes appropriately.

At a technical level, we maintain the state of the policy tree in an object that is shared by all three modules and that is not backtracked over by the regular backtracking mechanism. The variable and value ordering module stores in it the value choices that it made, for the global constraint to check; the 'leaf' module updates the values of the policy tree whenever it is called, and the variable and value ordering module resets these values for all nodes in a subtree before exploring that subtree. This policy tree object also provides an interface to the probabilistic inference engine and caches all queries to increase efficiency. The cached values are stored in a hash table without replacement.

4.5 Experiments

We investigate the following questions: **Q1:** Does our proposed method improve over existing approaches? **Q2:** What is the impact of bounding depth on the efficiency of search? **Q3:** What is the interplay between bounding and constraint propagation?

We used two problems in our experiments:

#stages	scen-vars	scen-**CP**	scen-**ILP**	**AOB&B**
1	1	0.000089	0.000416	0.000120
2	16	0.001410	0.002101	0.001291
3	241	T(S)	0.013641	0.016774
4	3616	T(S)	0.202595	0.19052
5	54241	T(S)	4.092760	4.02639
6	813616	T(S)	117.2034	75.9437
7	12204241	M(G)	M(G)	1494.54

Table 4.1: Comparing the runtime (s) of our method (AOB&B) with scenario-based approaches (CP and ILP) on the knapsack problem. The unsuccessful cases either ran out of time (T) or memory (M) during generation of scenario-based problem (G) or solving the problem (S).

#stages	scen-vars	scen-**CP**	scen-**ILP**	**AOB&B**
1	2	0.000134	0.000392	0.000213
2	34	0.000667	0.001199	0.001141
3	546	T(S)	0.023160	0.015718
4	8738	T(S)	0.724038	0.220448
5	139810	M(S)	34.09311	5.84844
6	2236962	T(S)	779.4099	162.892
7	35791394	M(G)	M(G)	T(S)

Table 4.2: Comparing the runtime (s) of our method (AOB&B) with scenario-based approaches (CP and ILP) on the investment problem. The unsuccessful cases either ran out of time (T) or memory (M) during generation of scenario-based problem (G) or solving the problem (S).

Knapsack (based on an example from (Hnich, R. Rossi, S. A. Tarim, and S. Prestwich 2011)) As items arrive, we must decide whether to pick or leave the item. The weight and cost of an item are stochastic, and only revealed immediately after a decision is made for that item. The weight (5 possibilities) and cost (3 possibilities) in each stage depend on a hidden variable (2 possibilities), which itself depends on the hidden variable of the previous stage. The goal is to maximize the expected sum of values under the constraint that the total weight does not exceed the capacity.

Investment At the start of each season, a company can invest in option **A** or **B**. The stochastic return is revealed at the end of the season. The two return values in each season (4 possibilities for each option) depend on the market sentiment (2 possibilities), which itself depends on the market sentiment in the previous season. Option **A** has a higher return on average, but a major tax relaxation is applied at the end of the horizon if the majority of income comes from investment in **B**. The goal is to maximize the expected returns under the

constraint that the tax relaxation applies.

The Bayesian Networks are HMMs similar to Figure 4.1 but with two observed variables per hidden variable (corresponding to each stage). The hidden variables have 0.9 probability for staying in the same state. Other distribution parameters for both problems were generated randomly, such that at each stage for Knapsack the weight and cost have positive correlation and for Investment the returns have negative correlation. Higher-order HMMs did not impact compilation or runtime much because of the small number of variables/depth.

We ran the experiments on Linux machines with 32 GB of memory. The time-out used was 1800 seconds. The CP solver used is Gecode-4.4.0 and the MIP solver is Gurobi-6.5[1]. We used the the ACE[2] package to compile the Bayesian networks into arithmetic circuits, and ported the inference library to C++ for integration with Gecode. The code and data are available online [3].

Results

To answer *Q1*, we compare our method with a scenario-based approach that copies the decision variables, using both a CP solver and a MIP solver. The latter is possible because both problems only have linear constraints, though our method can handle any CP constraint. Tables 4.1 and 4.2 show the results. Standard CP quickly fails due to the huge search space and weak pruning. ILP is more effective thanks to its presolving ($> 50\%$ reductions in variables) and cutting planes. However, our native method is faster and can handle larger number of stages.

To answer *Q2*, we compare the runtime of our method for different depths of the bound. Figure 4.3 shows that even the shallow bound is much better than no bound. Deeper bounds have a marginal gain in runtime for knapsack and a marginal overhead for investment but always lead to smaller search spaces (not shown). On the investment problem, it is clear that also bounding the AND nodes (or both) is better than only the OR nodes.

To answer *Q3*, we gradually tighten the capacity constraint in the *knapsack* problem and observe the effect on runtime, number of nodes, and number of failures. Figure 4.4 shows that in the absence of bounds, tightening the constraint leads to more failures and fewer nodes; meaning that the search space becomes smaller. When bounding is employed, tighter constraints increase both the number of nodes and failures. This indicates fewer solutions and weaker

[1] http://www.gecode.org and http://www.gurobi.com
[2] http://reasoning.cs.ucla.edu/ace
[3] https://github.com/Behrouz-Babaki/FactoredSCP

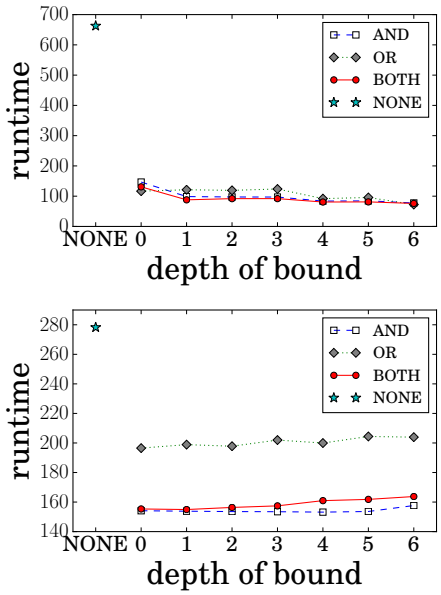


Figure 4.3: Effect of depth of bounds on instances of the *knapsack* problem (top) and *investment* problem (bottom).

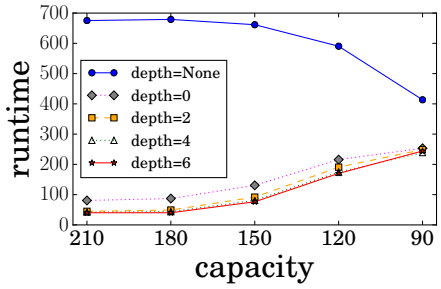


Figure 4.4: The effect of tightening the *knapsack* constraint on runtime in a 6-stage problem.

bounding. Such interactions demonstrate the need for approaches that can both handle constraints and prune with bounds.

	no bound		depth=6	
C	**#failures**	**#nodes**	**#failures**	**#nodes**
210	0.0E+00	1.1E+09	8.1E+05	1.8E+07
180	5.9E+04	1.1E+09	8.1E+05	1.8E+07
150	7.6E+05	1.0E+09	2.3E+06	5.4E+07
120	3.2E+06	9.3E+08	6.3E+06	1.5E+08
90	5.7E+06	6.4E+08	9.1E+06	2.4E+08

Table 4.3: The effect of tightening the *knapsack* constraint (C) on the number of nodes and failures in a 6-stage problem.

4.6 Related work

(Walsh 2002) also employed And-Or search (backtracking and forward checking) but no bounds, independent random variables and one global stochastic constraint. Nested constraint programming (Chu and Stuckey 2014) is a related framework in which stochastic CP as well as other *nested* problems can be expressed; their clause learning solver performs and-or search and caches the valuation of identical subproblems/nodes, but it assumes independent random variables. Quantified constraint optimization (Benedetti et al. 2008) nests existential and universal quantification but does not consider probability distributions. Arbitrary non-factored probability distributions have been considered before in a scenario-based setting (A. Tarim et al. 2006). Global chance constraints have been investigated in that setting too (Hnich, R. Rossi, S. A. Tarim, and S. D. Prestwich 2012). Another work investigates relaxation methods for convex expected utility functions (R. Rossi et al. 2008); see (Hnich, R. Rossi, S. A. Tarim, and S. Prestwich 2011) for a survey on such methods.

FSCPs are also related to influence diagrams (F. Jensen et al. 1994) but are different in that the utility function in influence diagrams is assumed to be additive and that (F)SCPs support arbitrary complex constraints over both decision and random variables; The typical jointree algorithms (F. Jensen et al. 1994) can have prohibitive memory requirements. A depth-first branch-and-bound algorithm was investigated (C. Yuan, X. Wu, et al. 2010) but the bound uses (simplified) influence diagrams itself. Other connections between constraint programs and probabilistic graphical exist. Notably probabilistic queries on *mixed deterministic and probabilistic networks* (Mateescu and Dechter 2008) and a more theoretical, unifying framework of algebraic graphical models (Pralet et al. 2007).

4.7 Conclusion and future work

We presented a novel stochastic constraint programming method with three distinguishing features: a novel bound that works on the And-Or search space directly; the use of (non-trivial) factorized probability distributions and querying during search using a state-of-the-art inference engine from the UAI community; and within a generic constraint solver meaning that any existing global constraint can be used. This allows to reason over larger problems for which grounding out all possible worlds is not feasible or incurs too much overhead. Our CP-based method can handle complex constraints and not just linear/convex constraints as MIP solvers do.

In the future, we aim to investigate a generic mechanism for global *chance constraints* (Hnich, R. Rossi, S. A. Tarim, and S. D. Prestwich 2012), which can be violated in a small amount of possible worlds (Walsh 2002). An option is also to to find approximate solutions by not exploring worlds with a very small probability/expected utility. A last promising avenue is to reason over probability distributions that are influenced by the *decisions* too, where our method has the advantage that it reasons over the (non-ground) problem structure directly.

Part II

Constrained Clustering using Integer Linear Programming

Chapter 5

Constrained Clustering using Column Generation

In this chapter, we investigate the problem of constraint-based clustering from an optimization point of view. We propose a new solution for minimum-sum-of-squares clustering under constraints, where the constraints considered are must-link constraints, cannot-link constraints and anti-monotone constraints on individual clusters. This chapter is based on the following publication:

- Behrouz Babaki, Tias Guns, and Siegfried Nijssen. "Constrained clustering using column generation". In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings,* volume 8451 of *Lecture Notes in Computer Science,* pages 438–454. Springer, 2014.

5.1 Introduction

One of the core problems studied in the data mining and machine learning literature is that of *clustering*. Given a database of examples, the clustering task involves identifying groups of similar examples; such groups are for instance indicative for patients with similar clinical observations, customers with similar purchase behaviour, or website visitors with similar click behaviour.

While the clustering problem is common in the data mining literature, it is only recently realized in the data mining community that this problem is closely related to problems studied in the optimization literature, and hence that open problems in clustering may be solved using generic optimization tools. In this chapter, we study one such open problem: *Optimal Constrained Minimum Sum-of-Squares Clustering* (MSSC). We show that a generic optimization strategy can be used to address this problem.

Many types of clustering problems are known in the literature; however, MSS clustering is arguably one of the most popular clustering settings. In MSS clustering, the task is to find a clustering in which each example is put into *exactly* one cluster. Clusters do not overlap and together they cover all the available data. Clusters should be chosen such that points within a cluster have small sum-of-squared distances.

The popularity of the MSS clustering setting is partially due to the *k-means* algorithm. K-means is a heuristic algorithm which quickly converges to a local minimum and is included in most data mining toolkits. Even though successful, basic k-means has several disadvantages. One is its randomized nature: each run of the algorithm may yield a different clustering. Another is its lacking ability to take into account prior knowledge of a user.

There are many types of prior knowledge that a user may have. A common perspective is to formalize prior knowledge in terms of *constraints* on the clusters one wishes to find (Basu et al. 2008), where the most popular constraints are *must-link* and *cannot-link* constraints. A must-link constraint enforces that examples that are known to be related, are part of the same cluster. A cannot-link constraint, on the other hand, enforces that examples that are not related are not part of the same cluster. These constraints are popular as many other constraints can be transformed into must-link and cannot-link constraints (Davidson and Ravi 2007). The maximum cluster diameter constraint, for instance, requires that each cluster must have a diameter of at least distance α; hence, any two points that are further than α apart cannot link together. The minimum cluster separation constraint requires that clusters must be separated by at least distance β; hence, any two points that are less than β apart must link together.

Other clustering settings that can be seen as constraint-based clustering problems are problems in which clusters need to have a minimum or maximum size, or where one is looking for alternative clusterings (Gondek and Hofmann 2004).

An important question is how to find a clustering that satisfies constraints. Here, most algorithms in the data mining literature take a heuristic approach. Arguably, the most well-known example is the *COP-k-means* algorithm (Wagstaff

and Cardie 2000), which modifies the *k-means* algorithm to deal with must-link and cannot-link constraints. Unfortunately, even though the algorithm is fast, it may not find a solution that satisfies all constraints even if such a solution exists (Davidson and Ravi 2007). In itself, this is not surprising as the problem is known to be NP hard and hence a polynomial solution is not likely to exist (Aloise, Deshpande, et al. 2009; Davidson and Ravi 2007). As a result, the problem of how to solve the MSSC problem under constraints is still open.

This chapter addresses this challenge and develops a generic approach that can find an optimal solution to constrained MSSC problems. While we will focus on must-link and cannot-link constraints, the approach allows for the inclusion of several other constraints as well; we will show that the approach works for all constraints that are *anti-monotone*.

Our approach builds on earlier work that showed the feasibility of *unconstrained* optimal MSS clustering (Merle et al. 1999; Aloise, Hansen, and Liberti 2012) by using *column generation* in an *integer linear programming* setting. The column generation process is here responsible for identifying candidate clusters that can be put into a clustering. We will show that most clustering constraints can be dealt with by *pushing* the constraints in a branch-and-bound algorithm for column generation.

This chapter is organized as follows. Section 5.2 introduces MSS clustering and MSS clustering under constraints. Section 5.3 gives an overview of how to find a solution using a column generation process, building on the earlier work of (Merle et al. 1999; Aloise, Hansen, and Liberti 2012). In Section 5.4 we introduce a branch-and-bound approach for generating columns under constraints. Section 5.5 discusses practical considerations in the implementation of this algorithm. Section 5.6 provides experiments, Section 5.7 discusses related work and Section 5.8 concludes.

5.2 MSSC

Assumed given is a dataset D with n data points. Each example in the dataset is a point p in an m-dimensional space and is represented by a vector with m values. One cluster is defined as a set of data points $C \subseteq D$. A cluster*ing* consists of k such clusters, and corresponds to a partitioning of the data into k groups. The number of clusters k is typically given upfront by the user. In MSS clustering, the clusters in a clustering are usually non-overlapping, that is, each data point belongs to exactly one cluster.

Given a cluster $C \subseteq D$, the cluster center or *centroid* is the mean of the data points that belong to that cluster:

$$z_C = \text{mean}(C) = \frac{\sum_{p \in C} p}{|C|} \tag{5.1}$$

The quality of a clustering can be measured in many different ways. In MSS clustering, the quality of a clustering is measured using the sum of squared distances between each point in a cluster and the centroid of the cluster: $SSC(\mathcal{C}) = \sum_{C \in \mathcal{C}} \sum_{p \in C} d^2(p, z_C)$, where $d(\cdot, \cdot)$ calculates the distance between two points, for example, the Euclidean distance.

Note that for a cluster C, the sum of squared distances to its centroid equals the sum of all pairwise distances between the points of that cluster, divided by the size of the cluster:

$$\sum_{p \in C} d^2(p, z_C) = \frac{\sum_{p_1, p_2 \in C} d^2(p_1, p_2)}{|C|} \tag{5.2}$$

For simplicity of notation, when we write $p_1, p_2 \in C$ we assume that every pair of two points in C is included in the sum exactly once. To summarize the MSSC problem, a mathematical programming formulation is given in Table 5.1.

The best known clustering algorithm that uses sum of squared distances is the k-means algorithm. It is an approximate algorithm that starts with an initial random clustering and iteratively minimizes the sum-of-squares using the following two steps: 1) add each data point to the cluster with closest cluster centre; 2) compute the new cluster centre of the resulting clusters. These two steps are iterated until convergence, that is, the cluster centres do not change any more. This procedure can get stuck in local minima and it is not uncommon that two different runs (e.g. with different initial clusters) produce different clusterings.

Constraints. The most well-known constraints are must-link and cannot-link constraints. Let ML and CL be subsets of $D \times D$. Then a cluster C satisfies a must-link constraint $(p_1, p_2) \in ML$ iff $|\{p_1, p_2\} \cap C| \neq 1$; it satisfies a cannot-link constraint $(p_1, p_2) \in CL$ iff $|\{p_1, p_2\} \cap C| \leq 1$.

Note that both constraints can be evaluated on the individual clusters in a clustering. This is a key observation for our work.

In a seminal paper by Wagstaff et al (Wagstaff and Cardie 2000), the COP-k-means algorithm is proposed. COP-k-means is an extension of the k-means algorithm towards must-link and cannot-link constraints. It modifies the k-means algorithm by not assigning each point to its closest cluster centre, but

rather to the closest centre that satisfies all constraints. If no such centre exists, the algorithm terminates. An alternative approach is to continue running the algorithm, even though the final solution might then not satisfy all constraints; in any case, the algorithm is not guaranteed to find a solution even if there exists one.

Many other constraints are possible. We will not give a complete overview here (see Section 5.7 and (Basu et al. 2008)). For this work it is however important to observe that many problems can be formalized using constraints that are *anti-monotone*. We call a boolean constraint $\varphi(C)$ on a cluster C of data points *anti-monotone* iff $\varphi(C)$ implies $\varphi(C')$ for all $C' \subseteq C$. The cannot-link constraint is anti-monotone: if a cluster C satisfies a cannot-link constraint, every subset also satisfies this constraint. There are many other anti-monotone constraints:

- a maximum cluster size constraint on clusters $|C| \leq \theta$, which can be used to avoid that one cluster dominates a clustering;
- a maximum overlap constraint $|C \cap X| \leq \theta$, which can be used to avoid that any cluster found is too similar to a given set of points X; this generalizes the cannot-link constraint;
- a minimum difference constraint $|C \backslash X| \leq \theta$, which requires a certain similarity to cluster X;
- a soft cannot-link constraint, which requires that the number of pairs of points in a cluster that have a cannot-link constraint among them is bounded;
- conjunctions or disjunctions of anti-monotone constraints.

A conjunction of anti-monotone constraints can for instance be used to find an alternative clustering: starting from a clustering $\mathcal{C}$, we can enforce that in a new clustering every cluster is different from all clusters in the earlier clustering.

Must-link constraints are an example of constraints that are not anti-monotone.

In the following sections, we will show how to solve the MSS problem under a combination of anti-monotone constraints and must-link constraints, by adapting a state-of-the-art unconstrained optimal clustering algorithm. A feature of the algorithm is that it exploits the anti-monotonicity of cluster constraints.

$$\underset{\mathcal{C}}{\text{minimize}} \quad \sum_{C\in\mathcal{C}} \sum_{p\in C} d^2(p, z_C), \tag{5.3}$$

s.t.

$$C_1 \cap C_2 = \emptyset \qquad \forall C_1, C_2 \in \mathcal{C} \tag{5.4}$$

$$|\bigcup_{C\in\mathcal{C}} C| = n \tag{5.5}$$

$$|\mathcal{C}| = k \tag{5.6}$$

Table 5.1: MSS clustering

$$\underset{x}{\text{minimize}} \quad \sum_{t\in T} c_t x_t, \tag{5.7}$$

s.t.

$$\sum_{t\in T} x_t a_{it} = 1 \qquad \forall i \in \{1, \ldots, n\} \tag{5.8}$$

$$\sum_{t\in T} x_t = k \tag{5.9}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{5.10}$$

Table 5.2: An ILP model for MSS clustering

5.3 Column generation framework

In this section we give a brief overview of an ILP formulation of MSSC and a column generation method for solving it, based on the (unconstrained) MSSC column generation framework of Aloise et al. (Aloise, Hansen, and Liberti 2012). The next section will introduce our proposed approach for taking constraints into account.

An ILP formulation of MSSC. Given a dataset with n data points, the number of possible clusters is 2^n. In principle, we can hence reformulate the clustering problem using a Boolean n by 2^n matrix A that represents all possible clusters: each column is a cluster where $a_{it} = 1$ if data point p_i is in cluster t and $a_{it} = 0$ otherwise. We define the *cost* of a cluster (column) as the sum of squared distances of the points in the cluster to its mean: $c_t = \sum_{i=1}^{n} d^2(p_i, z_t)a_{it}$.

The problem in equations 5.3-5.6 can then be formulated as an Integer Linear Program as in Table 5.2 (Merle et al. 1999), where $T = \{1, \ldots, 2^n\}$ denotes all possible clusters. Equation 5.7 corresponds to the SSC criterion. Equation 5.8 states that each data point must be covered exactly once. Hence it enforces both that the clusters are not overlapping and that all points are covered. Equation 5.9 finally ensures that exactly k clusters are found. Note that the k-means (and COP-k-means) algorithm can return empty clusters and hence less than k clusters in some occasions. This can not arise in the above formulation.

For even moderate sizes of n the number of clusters will be too large to solve the above ILP by first materializing A. However, we can use a column generation approach in which the master problem (Eq. 5.7-5.10) is *restricted* to a smaller set $T' \subseteq T$ and columns (clusters) are incrementally added until the optimal solution is provably found.

Column Generation iterates between solving the restricted master problem and adding one or multiple columns. A column is a candidate for being added to the restricted master problem if adding it can improve the objective function. If no such column can be found, one is certain that the optimal solution of the restricted master problem is also the optimal solution of the full master problem. Whether a column can improve on the objective can be derived from the dual.

The dual of the master problem (Table 5.2) is given in Table 5.3.

$$\underset{\lambda,\sigma}{\text{maximize}} \quad -k\sigma + \sum_{i=1}^{n} \lambda_i \tag{5.11}$$

s.t.

$$-\sigma + \sum_{i=1}^{n} a_{it}\lambda_i \leq c_t \quad \forall t \in T \tag{5.12}$$

$$\lambda_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \tag{5.13}$$

$$\sigma \geq 0 \tag{5.14}$$

Table 5.3: Dual of the optimization problem.

$$\underset{x}{\text{minimize}} \quad \sum_{t \in T} c_t x_t + \sum_{i=1}^{n} \theta_i y_i, \tag{5.15}$$

s.t.

$$\sum_{t \in T} x_t a_{it} + y_i = 1 \quad \forall i \in N \tag{5.16}$$

$$\sum_{t \in T} x_t = k \tag{5.17}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{5.18}$$

$$-\mu \leq y_i \leq \mu \qquad \forall i \in N \tag{5.19}$$

Table 5.4: Model with stabilization included ($N = \{1, \ldots, n\}$).

Here λ_i indicates a dual value corresponding to the constraint in equation 5.8 and σ a dual value corresponding to equation 5.9. One column in the master problem corresponds to one constraint in the dual (Equation 5.12).

Given values for λ and σ, obtained by solving a restricted master problem, we need to determine whether there are columns for which $\sigma - \sum_{i=1}^{n} a_{it}\lambda_i + c_t < 0$, that is, whether there are columns with a *negative reduced cost*. If no such column can be found, the current solution is optimal.

Finding a column with negative reduced cost is called *pricing*. While a pricing routine can return any column with a negative reduced cost, one typically searches for the smallest one; hence we are interested in finding:

$$\arg\min_{t \in T} \quad \sigma - \sum_{i=1}^{n} a_{it}\lambda_i + c_t. \tag{5.20}$$

Solving this pricing problem is not trivial, given the large number of columns. The details of solving the pricing subproblem will be discussed in more detail in Section 5.4.

When solving the restricted master problem, it is possible that it has no feasible solution. In this case, Farkas' Lemma (Schrijver 2003) can be used to add columns that gradually move the solutions of the restricted master problems closer to the feasible region, or to prove infeasibility of the master problem. This Farkas pricing is similar to the regular pricing explained above. In this case, the problem to optimize is:

$$\arg\min_{t \in T} \quad \sigma' - \sum_{i=1}^{n} a_{it}\lambda'_i \tag{5.21}$$

where σ' and λ' are the dual Farkas values. Note that this is the same problem as the regular pricing problem above, with the exception that the cost c_t of the cluster does not need to be taken into account.

5.4 Column generation with constraints.

Given the earlier observations, one can see that enforcing constraints on clusters C amounts to removing from the cluster matrix A all clusters that do not satisfy these constraints. In a column generation scheme, this means that it is sufficient to add these constraints to the subproblem solver; they do not need to be added to the master problem. The rest of this section explains our proposed branch-and-bound method for solving the (constrained) subproblem.

5.4.1 Subproblem solving

Essentially, in each iteration of the column generation process we need to solve a constrained minimisation problem. The objective function to minimize is given by equation 5.20 (equation 5.21 in case of infeasibility). By removing the constant σ and using equation 5.2, we can rewrite the objective as:

$$\arg\min_{t\in T} \quad \sum_{i=1}^{n} d^2(p_i, z_t)a_{it} + \sigma - \sum_{i=1}^{n} a_{it}\lambda_i \tag{5.22}$$

$$= \arg\min_{t\in T} \quad \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} d^2(p_i, p_j)a_{it}a_{jt}}{\sum_{i=1}^{n} a_{it}} - \sum_{i=1}^{n} a_{it}\lambda_i \tag{5.23}$$

Let us represent the cluster $t \in T$ and its corresponding column $a_{\cdot t}$ as a set X. We define $d(X) = \sum_{i,j\in X} d^2(p_i, p_j)$, where every pair is only considered once in the sum, $d(X, Y) = \sum_{i\in X, j\in Y} d^2(p_i, p_j)$ and $\lambda(X) = \sum_{i\in X} \lambda_i$. We can now rephrase our problem as that we wish to search for a cluster X:

$$\arg\min_{X} \quad \frac{d(X)}{|X|} - \lambda(X) \tag{5.24}$$

and such that all constraints on clusters are satisfied.

Blocks. A first simple observation is that the must-link constraints are transitive and hence the must-link relation is an equivalence relation. We will refer to the equivalence classes as *blocks*. We can rephrase our optimization problem as an optimization problem over the blocks. Let $X = [p_i]_{ML}$ denote the block that point $p_i \in D$ belongs to (a point can never belong to two blocks) and let $D/ML = \{[p_i]_{ML} \,|\, p_i \in D\}$ denote the blocks in the data. We are looking for a subset of the blocks $\bar{X} \subseteq D/ML$ such that the following criterion is minimized:

$$f(\bar{X}) = \left(\sum_{X\in\bar{X}} d(X) + \sum_{X,Y\in\bar{X}} d(X, Y) \right) / \sum_{X\in\bar{X}} |X| - \sum_{X\in\bar{X}} \lambda(X). \tag{5.25}$$

Note that we can precompute the terms $d(X)$, $d(X, Y)$, $|X|$ and $\lambda(X)$ for all $X, Y \in D/ML$. Note furthermore that if $ML = \emptyset$ then $\forall X \in \bar{X} : |X| = 1$ and this formula is identical to the one without constraints.

In addition, the choice of $\bar{X}$ has to satisfy the cannot-link constraint: for no two $X, Y \in \bar{X}$ it may be the case that $i \in X, j \in Y, (i, j) \in CL$.

Algorithm 3 Branch-and-bound(Set: $\bar{X}$, Set: $\bar{C}$)

$\bar{X}$ is the current set of blocks under consideration, $\bar{C}$ the possible extensions to $\bar{X}$.

```
1: C̄ := reduce-candidates(X̄, C̄)
2: if not prunable(X̄, C̄) then
3:     Store C̄ in a stack
4:     Process X̄ as candidate cluster
5:     while C̄ is not empty do
6:         C := C̄.pop ()
7:         Branch-and-bound (X̄ ∪ {C}, C̄)
8:     end while
9: end if
```

Algorithm. We propose to use a branch-and-bound algorithm to solve this problem. This algorithm performs a set-enumeration and is given in Algorithm 3 (initialized with Branch-and-bound($\{\}$, D/ML)). It uses newly developed pruning strategies to make the search feasible and is easily extended to include a wide range of constraints. In order to prune candidates, we either remove some candidates from consideration (line 1) or discard a branch of the search tree using bounds on the objective function (line 2).

The removal of candidates in line 1 corresponds to propagation in a constraint programming setting (Dao et al. 2013). However, we will show that the proposed bound used in line 2 is not valid in the presence of arbitrary constraints and hence cannot be used in general.

5.4.2 Reducing the number of candidates

We employ three strategies to reduce the set of candidates in line 1 of Algorithm 3:

Cannot-link constraints. The cannot-link constraint is easily taken into account: when there is a cannot-link constraint between a block in $\bar{C}$ and a block in $\bar{X}$, the block is removed from $\bar{C}$.

Anti-monotone constraints other than cannot-link constraints are easily included as well: if a set $\bar{X} \cup \{C\}$ does not satisfy an anti-monotone constraint, the candidate C can be removed in line 1.

Block compatibility. Assume that we have a block $C_1 \in \bar{X}$ and a block $C_2 \in \bar{C}$ and the following holds:

$$\frac{d(C_1) + d(C_2) + d(C_1, C_2)}{|C_1|+|C_2|} - \lambda(C_1) - \lambda(C_2) > 0,$$

then any cluster $\bar{X}'$ we could build that includes both C_1 and C_2 can be improved by removing both C_1 and C_2:

$$f(\bar{X}) = \sum_{p_i \in \cup \bar{X}} d(p_i, z_{\cup \bar{X}})^2 - \sum_{X \in \bar{X}} \lambda(X) \geq$$

$$\sum_{p_i \in \cup \bar{X} \setminus \{C_1, C_2\}} d(p_i, z_{\cup \bar{X} \setminus \{C_1, C_2\}})^2 + \sum_{p_i \in C_1 \cup C_2} d(p_i, z_{C_1 \cup C_2})^2 - \sum_{X \in \bar{X}} \lambda(X) \geq$$

$$\sum_{p_i \in \cup \bar{X} \setminus \{C_1, C_2\}} d(p_i, z_{\cup \bar{X} \setminus \{C_1, C_2\}})^2 + - \sum_{X \in \bar{X} \setminus \{C_1, C_2\}} \lambda(X). \quad (5.26)$$

Note that this argument is only valid in the presence of anti-monotone constraints in combination with must-link constraints. We refer to this test as a *compatibility test*. When a block in $\bar{C}$ is incompatible with a block in $\bar{X}$, the block is removed from $\bar{C}$.

5.4.3 Pruning using a bound on the objective function

For the remaining set of candidates, a more elaborate test is carried out to determine whether to continue the search (line 2). This test consists of calculating a bound on achievable solutions and comparing it with the best solution found so far. A key feature of this bound is that it can be calculated efficiently.

The key idea is as follows. Let $\bar{X}'$ be a set that is found below a set $\bar{X}$ in the search tree, that is, $\bar{X}' \subseteq \bar{C} \cup \bar{X}$. We can write its quality as follows:

$$(\underbrace{d(\cup \bar{X})}_{\text{old}} + \underbrace{\sum_{X \in \bar{X}' \setminus \bar{X}} \beta(\bar{X}, X)}_{\text{(1) between old and new}} + \underbrace{\sum_{X, Y \in \bar{X}' \setminus \bar{X}} d(X, Y)}_{\text{(2) between new blocks}})/ \underbrace{\sum_{X \in \bar{X}'} |X|}_{\text{(3) sizes}} - \underbrace{\sum_{X \in \bar{X}'} \lambda(X)}_{\text{(4) lambdas}},$$

where $\beta(\bar{X}, X) = d(X) + \sum_{Y \in \bar{X}} d(X, Y)$.

Essentially, we need to have a bound on the best $\bar{X}'$. An important first concern is that we do not know the size of the best $\bar{X}'$ and hence we do not

know term (3). We simplify this problem by iterating over all cluster sizes $\sum_{X\in\bar{X}}|X| \leq s \leq \sum_{X\in\bar{X}}|X| + \sum_{C\in\bar{C}}|C|$ and calculating a bound on the quality assuming the best cluster has size s, i.e., we calculate a bound on the above formula assuming part (3) is iteratively fixed. The overall bound is the best bound among all the sizes considered.

Calculating a lower bound for a fixed value s of (3) requires a lower bound on (1) and (2), and an upper bound on (4). We discuss each in turn. A lower bound on part (1) for a given size s is obtained as follows:

- sort all $C \in \bar{C}$ increasing in their $\beta(\bar{X}, C)/|C|$ values, yielding order $C_1, \ldots, C_m$;
- determine the largest value k such that $\sum_{i=1}^{k}|C_i| \leq s$;
- determine $\sum_{i=1}^{k}\beta(\bar{X}, C_i)$ as bound.

The argument for this is as follows. All additional points that are selected by the algorithm above in $C_1, \ldots, C_k$ are characterized by the $\beta(\bar{X}, C)/|C|$ value their corresponding block has. If we sum these characteristic values over all points, the result is $\sum_{i=1}^{k}\beta(\bar{X}, C_i)$. Choosing the lowest possible characteristic values is a lower bound as the sum of characteristic values of the points in the optimum $\bar{X}^*$, and hence also the value $\sum_{X\in\bar{X}^*\setminus\bar{X}}\beta(\bar{X}, X)$, can never be better.

A similar algorithm can be used to determine an upper bound for term (4):

- sort all $C \in \bar{C}$ decreasing in their $\lambda(C)/|C|$ values, yielding order $C_1, \ldots, C_m$;
- determine the smallest value k such that $\sum_{i=1}^{k}|C_i| \geq s$;
- determine $\sum_{i=1}^{k}\lambda(C_i)$ as bound.

A simple lower bound on term (2) is that it is always higher than zero. Calculating a good bound is hard, as we essentially need to solve an edge-weighted clique problem.

While the overall bound obtained is not very tight, also because term (1) and term (4) are sorted independently, it has important computational advantages. First, we can sort the $\lambda(C)/|C|$ and $\beta(\bar{X}, C)/|C|$ values before iterating over potential sizes; hence, we can avoid doing this repeatedly for each size s. Second, we do not need to consider all sizes s indicated earlier. If we consider the sorted ranges of λ and β values, there are ranges of sizes in which the bound does not

change; the bound only changes when either a lambda value changes or a β value changes. It hence suffices to consider $2|\bar{C}|$ different sizes for s. Finally, we can maintain the bounds incrementally.

As a result, the overall bound over all sizes s can be calculated in $O(|\bar{C}|\log|\bar{C}|)$ time. As furthermore all required counts can be maintained incrementally in $O(|\bar{C}|)$ time, the overall time spent in one call of the Branch-and-bound algorithm (excluding recursive calls) is $O(|\bar{C}|\log|\bar{C}|)$; in other words, the complexity of the algorithm is *not* dependent on the number of points in the data, but *only* on the number of blocks that the must-link constraints identify in it.

5.5 Practical considerations

The column generation approach, in combination with the branch-and-bound algorithm, provides a fundamental approach for finding optimal solutions under constraints. However, several practical considerations are of importance when implementing the column generation approach.

5.5.1 Initialisation

Initially, there are no columns in the restricted master problem. This means that Farkas pricing needs to be performed until a feasible solution is found, which can be time consuming. However, assuming a heuristic solver such as COP-k-means finds a solution, one can initialize the restricted master problem with this known (sub-optimal) solution. This avoids the need for Farkas pricing, provides a number of good initial columns (cuts to the dual problem) as well as an upper bound for the master problem.

5.5.2 Branching

Integer linear programs are typically solved by solving a number of LP relaxations and using branching to enforce integrality. So far, we have described how we employ the column-generation method for solving the LP relaxations. In theory, if the solution to the linear program is fractional any type of branching can be used. In previous work (Merle et al. 1999) a Ryan-Foster branching scheme was employed. In this scheme, in the restricted problem two columns are determined that have a corresponding fractional value and that cover the same data point (p_1). Branching will enforce that in subsequent problems only one of these two

Figure 5.1: Run times on the Iris data set.

columns can cover that point. Observe that no two columns cover exactly the same data points and hence they must differ in at least one data point (p_2). We can now branch by enforcing that in one branch points p_1 and p_2 are in the same cluster and that in the other branch p_1 and p_2 are not in the same cluster.

This type of branching naturally fits our approach as it corresponds to adding a must-link or cannot-link constraint. Compared to (Merle et al. 1999), the proposed approach can hence handle both constrained and unconstrained cases in the same principled manner.

5.5.3 Slow convergence

Many large-scale column generation approaches suffer from slow convergence. Similar to (Merle et al. 1999), we also observed degeneracy in our experiments: even when given the optimal solution, a large number of column generation iterations is required before the optimality is proved. We implemented a dual stabilisation scheme similar to the one of (Merle et al. 1999): adding a linear penalisation to the dual objective corresponds to adding a *perturbation* variable to each of the constraints in equation 5.8 and adding them to the objective function, given in Table 5.4. Here y_i are the perturbation variables, $+/-\mu$ its bounds and θ_i its coefficients in the objective function. The θ_i form a stabilisation centre in the dual that will penalize duals that are too far from it. A good choice for θ is the dual λ values from the best known solution so far. The value of μ has to be progressively decreased until 0. At this point, all perturbation variables are 0 and the problem is identical to the original restricted master problem.

We employ a scheme where the θ_i are given an equal initial value and μ is set to 0.99. Each time an optimal solution to the perturbed restricted master problem is found, the θ_i values are changed to the duals of that optimal solution and μ is divided by 2^ℓ where ℓ is a counter of the number of such updates.

5.6 Experiments

Data was obtained from the UCI machine learning repository (Bache and Lichman 2013). Table 5.5 lists the properties of the datasets.

name	# points	dimensions	# labels
Iris	150	4	3
Wine	178	13	3
Soybean	47	35	4

Table 5.5: Description of datasets

We used the open-source SCIP (Achterberg 2009) system as column generation framework. The branch-and-bound pricer is written in C++. Source code is available at `http://dtai.cs.kuleuven.be/CP4IM/cccg/`. All experiments were run on quad-core Intel 64 bit computers with 16GB of RAM running Ubuntu Linux 12.04.3.

Constraints were generated according to the common methodology of (Wagstaff and Cardie 2000): two data points are repeatedly sampled randomly from labelled data; if they have the same label a ML constraint is generated, otherwise a CL constraint. This is repeated until the required number of constraints is generated. The code for generating these constraints and for the COP-k-means algorithm were obtained from `http://www.cs.ucdavis.edu/~davidson/constrained-clustering/` .

It is common practice to run (COP-)k-means multiple times to avoid that it is stuck in a local minimum. For each setting, we ran COP-k-means 500 times. The implementation obtained continues until convergence and is not guaranteed to satisfy all constraints. We will report on the number of runs that satisfy all constraint (COP sat). Only when at least one solution is found that satisfies all constraints will we report on its quality (COP max).

We initialized our column generation method with the *best* solution found by COP-k-means. Best is here defined by the clustering with the largest number of clusters satisfying all constraints. Among these clusterings, the one with the lowest MSS is selected. Note that in case COP-k-means did not find a solution satisfying all constraints, our column generation method started with the *best* infeasible solution. The stabilisation parameter μ was set to 0.99. Initial perturbation values θ_i can be set to any value; the update mechanism is explained in Section 5.3. In case a feasible solution is at hand, a good initial value for θ_i can be obtained from bounds on the dual variables. These bounds are calculated as in (Merle et al. 1999), and we used the lower bounds of the dual variables to initialize the corresponding θ_i.

The branch-and-bound method for solving the subproblem maintains a list of all clusters that improve the bound during search (including the final best one). All these clusters are added as columns to the restricted master problem.

k=3			
#c	**COP sat**	**COP max**	**CG best**
2	100.00%	90.3725	90.3725
60	100.00%	83.6675	83.6675
100	37.20%	87.2082	87.2082
140	0%	-	87.8750*
200	0%	-	89.1496*
240	0%	-	85.2477*
300	0%	-	89.3868*
340	31.40%	89.3868	89.3868*
400	0%	-	89.3868*
440	31.00%	88.6409	88.6409*
500	0%	-	89.3868*

Table 5.6: Clustering with 3 clusters and '#c' constraints, Iris dataset. *optimality proven

Results. We compare the result of our column generation approach to that of repeated runs of COP-k-means. Our column generation approach is initialized as explained above, and a time-out of 30 minutes is used.

Tables 5.6 and 5.7 shows the quality of the results for the Iris dataset, for $k = 3$ (the true number of class labels) and $k = 5$, respectively; Figure 5.1 gives an impression for the amount of run time it took to calculate these results.

A first observation is that in case of $k = 3$, and a low number of clusters, COP-k-means easily finds clusterings that satisfy the constraints (indicated by "COP sat"). For higher numbers of constraints, COP-k-means encounters more problems finding clusterings satisfying all constraints. In multiple cases none of the 500 runs finds a clustering satisfying all constraints. When we increase the number of clusters to $k = 5$, the constrained clustering problem becomes easier (Davidson and Ravi 2007); as a consequence, COP-k-means can find satisfying solutions more easily. Even when COP-k-means can not find a solution, our method finds acceptable clusterings; even optimal ones are found for higher numbers of constraints. The case of 140 constraints is an exception. For $k = 5$ and higher numbers of constraints, our method can find the optimal constrained clustering.

Tables **??** and 5.9 showsthe results for the bigger Wine dataset. This dataset is much harder, both for COP-k-means and for the column generation approach. In case of $k = 3$, the true number of class labels, COP-k-means is again rarely able to find a solution satisfying all constraints. The CG approach is able to find solutions for some cases, but can not prove them optimal within the time-out.

k=5				
#c	**COP sat**	**COP max**	**CG best**	**impr.**
2	100%	46.5616	46.5616	0%
60	100%	53.399	53.399	0%
100	100%	57.3827	57.3804*	0.004%
140	100%	63.1699	62.2115*	1.5%
200	100%	71.1401	69.3154*	2.56%
240	100%	72.7078	69.9776*	3.76%
300	83.6%	82.0819	81.9792*	0.13%
340	100%	85.9036	82.9945*	3.39%
400	100%	84.0495	84.0357*	0.02%
440	100%	82.6373	82.6373*	0%
500	100%	85.8908	85.8719*	0.02%

Table 5.7: Clustering with 5 clusters and '#c' constraints, Iris dataset. *optimality proven

k=3			
#c	**COP sat**	**max**	**CG best**
240	0%	-	4860250*
300	0%	-	5133144*
340	0%	-	5214981*
380	0%	-	5220299*
420	0%	-	5232632*
460	0%	-	5232632*
500	0%	-	5232632*

Table 5.8: Clustering with 3 clusters and '#c' constraints, Wine dataset. *optimality proven

In case of $k = 5$ the problem becomes easier, as was the case on Iris. We can see that the CG approach can sometimes greatly improve the best solution found in 500 COP-k-means runs, even without being able to prove its optimality.

Table 5.10 shows results on the Soybean dataset, a smaller dataset of higher dimensionality; its true number of labels is 4. Observe that for $k = 3$ and 80 constraints, CG is able to **prove** that this problem is infeasible. The heuristic COP-k-means simply does not find a solution, as happens for 40 and 60 constraints. We further note that in contrast to $k = 4$, for $k = 5$ COP-k-means is often not able to find the optimal solution.

k=5				
#c	**COP sat**	**COP max**	**CG best**	**impr.**
240	100%	4021090	3327908*	17.24%
300	0%	-	4077296*	+
340	16.6%	4659910	4329603*	7.09%
380	66.6%	4729860	4450036*	5.92%
420	59.6%	4740180	4537678*	4.27%
460	94.2%	4819200	4540041*	5.79%
500	15%	4922560	4684355*	4.84%

Table 5.9: Clustering with 5 clusters and '#c' constraints, Wine dataset. *optimality proven

	k=3		k=4		k=5	
# cons	COP sat.	CG gap	COP sat.	CG gap	COP sat.	CG gap
2	100.00%	0	100.00%	0	100.00%	0.00%
10	100.00%	0	100.00%	0*	100.00%	4.56%*
20	100.00%	0*	100.00%	0.12%*	100.00%	0.29%*
40	0.00%	339*	100.00%	0*	100.00%	1.25%*
60	0.00%	418*	52.60%	0*	81.20%	0.24%*
80	0.00%	INF	74.00%	0*	27.00%	0.38%*

Table 5.10: Soybean, different k and number of clusters (#c); GC gap = difference between best solution quality of cop-kmeans and the solution of CG, INF = infeasible.

5.7 Related work

We build on a column generation approach first described in (Merle et al. 1999) and improved in (Aloise, Hansen, and Liberti 2012). This earlier work only studies *unconstrained* clustering settings. We show that with modifications it can also be used in the presence of constraints. The main necessary modification is in the subproblem solver. We use a branch-and-bound approach that directly solves the subproblem and can be used in the presence of any constraint that is anti-monotone.

A feature of the first approach (Merle et al. 1999) is that it uses a heuristic *Variable Neighborhoud Search* method to solve a subproblem, and only when a solution can not be found in this way an exact method is used. The exact method uses Dinkelbach's lemma (Dinkelbach 1967) to solve equation 5.23 through a series of unconstrained quadratic 0-1 problems. The latter are solved using a heuristic VNS combined with an exact branch-and-bound algorithm for

verifying the stopping criterion of the Dinkelbach method.

This method is improved in (Aloise, Hansen, and Liberti 2012). One of these improvements is the introduction of a compatibility test. We adapted this test for use in the presence of must-link constraints.

Other exact methods for MSSC are branch and bound methods (Koontz et al. 1975; Diehr 1985; Brusco and Stahl 2005), a cutting plane algorithm that starts from the observation that MSSC is a concave optimisation problem (Xia and Peng 2005), dynamic programming (R. E. Jensen 1969; Os and Meulman 2004) and a branch-and-cut semi-definite programming algorithm (Aloise and Hansen 2009). These methods do not consider the addition of extra constraints.

Exact methods for constrained-based clustering have been studied before. Typical is that they do not use MSS as optimisation criterion, but rather a function that is linear or quadratic. Saglam et al. (Saglam et al. 2006) use an integer linear programming approach for minimizing the maximum cluster diameter. More recently, constraint programming has been used for solving constrained clustering tasks (Dao et al. 2013). A range of constraints is supported including instance-level constraints, size of cluster constraints and constraints on the separation between clusters and maximum diameter of a cluster. As objective function the (non-normalized) sum of squared distances between clusters or maximum diameter is supported.

A large class of clustering methods are those that evaluate the quality of a cluster based on a cut-value. Also in such methods the use of column generation has been proposed (Johnson et al. 1993). The inclusion of constraints in this method may be a topic for further research.

Exact methods are also used as part of approximate constraint-based clustering methods. Demiriz et al. (Demiriz et al. 2008) propose to modify k-means such that the assignment step, where points are assigned to their nearest feasible cluster, corresponds to solving an LP. Constraints on minimum cluster size can be taken into account, as well as instance level constraints. Davidson et al. studied the use of SAT solvers, also using diameter as optimization criterion (Davidson, Ravi, and Shamis 2010). Müller and Kramer (Mueller and Kramer 2010) use integer linear programming to solve constrained clustering tasks where a fixed number of candidate clusters is given upfront. The problem consists of selecting the right subset of clusters, which can be compared to solving one iteration of the restricted master problem. They investigate a number of different optimisation criterion, as well as constraints at the clustering level, such as the maximum amount of overlap between clusters or logical formula over entire clusters. These methods are not guaranteed to find globally optimal solutions.

5.7.1 Recent developments

After publication of this work, the problem of exact constrained MSS clustering has been further studied and significant improvements have been achieved in this direction. (Dao et al. 2015) present a framework based on constraint programming. They introduce filtering algorithms that compute a bound on the objective function of the clustering problem. This framework is improved by using a different filtering algorithm in a later study (Guns, Dao, et al. 2016). Both studies demonstrate improvements over our method in their experiments. In general, using constraint programming for solving clustering problem is an emerging topic, which aims at offering generic systems that can deal with a range of objective functions and constraints (Dao et al. 2017).

5.8 Conclusions

We proposed a column generation strategy for solving the constrained MSS clustering problem. The main novelty is a branch and bound algorithm that directly solves the subproblem. Experiments showed its promise: in cases where the COP-k-means algorithm is not able to find a solution satisfying all constraints even in 500 runs, CG could find solutions and in several cases even prove their optimality.

Several open questions remain. Degeneracy was not a main concern in this study, however we observe that with the simple stabilisation scheme described in section 5.5 the master problem still converges very slowly. It is worth investigating if advanced stabilisation techniques work better (Merle et al. 1999). Furthermore, the pruning strategy in the branch-and-bound algorithm could be improved and the branch-and-bound could be expanded to deal with additional constraints.

Chapter 6

A Branch-and-Cut Algorithm for Constrained Graph Clustering

In this chapter we study a graph clustering problem motivated by an application in the analysis of biological data. This problem has two distinguishing characteristics: First, the subgraph induced by each cluster has to be connected. Second, the objective function is based on penalties rather than densities or other standard quality measures. In this chapter, we explain the problem including the constraints and quality measures, and propose to use generic Mixed Integer Programming to solve it. This chapter is based on the following publication:

- Behrouz Babaki, Dries Van Daele, Bram Weytjens, and Tias Guns. "A branch-and-cut algorithm for constrained graph clustering". *Data Science meets Optimization workshop (colocated with CPAIOR), Padova, Italy, 2017.*

6.1 Introduction

Clustering is the task of partitioning a set of entities into homogeneous subsets. The quality of the clustering is typically determined by the *distance* between the entities. In graph clustering (Schaeffer 2007), each entity is assumed to be

a node in a graph; this graph is typically not fully connected. The quality is then determined by the *density* of the entities within a cluster or by the *cut size* between the clusters, that is the number of edges shared between different clusters. The latter is for example frequently studied in the field of graph partitioning (Buluç et al. 2016). Graph clustering has applications in many domains including social network analysis and community detection, information networks, transportation and logistics, and bioinformatics (Schaeffer 2007).

As data mining is increasingly applied on more and more problems in different domains, it increasingly happens that existing clustering methods are not suited for the problem at hand. This is either because the problem domain imposes additional constraints that can not be expressed in these methods, or because the objective function has a non-standard form. This has lead to the field of *constrained clustering*, which studies clustering problems involving different constraints and objectives (Basu et al. 2008). An increasingly popular way to handle a broad range of constraints and objectives is to use generic optimisation tools. In other words, to cast the problem as an optimisation problem and to use generic (discrete) optimisation solvers such as constraint programming (Dao et al. 2013), mixed integer programming (Gilpin et al. 2013; Babaki et al. 2014) or maximum satisfiability solvers (Berg and Järvisalo 2017). Though scalability can be an issue, these solvers can intrinsically handle different objectives and constraints.

The problem we study in this chapter is such a graph clustering problem that does not fit existing approaches. It is part of a bigger pipeline in computational cancer research, where the goal is to find *pathways* in gene interaction networks. There is hence an *interaction network* and it is a hard constraint that all nodes belonging to a cluster must be connected in this interaction network. To evaluate the quality there is a separate weighted *co-occurence* network and the nodes belonging to a cluster should have a low co-occurence penalty. Furthermore, small pathways are biologically less meaningful and hence the size of the clusters should also be maximized. The problem is hence a bi-objective graph clustering problem with hard connectivity constraints on a separate network. Existing methods are not able to handle such a complex setting, hence we study a mixed integer programming approach.

More specifically, our contributions are as follows: 1) we identify a new bi-objective constrained graph clustering with applications in bio-informatics and present an MIP formulation of the problem; 2) in order to better handle the large number of connectivity constraints, we propose a branch-and-cut approach that adds connectivity constraints as needed using the principle of node-cut sets. Our experiments demonstrate the effectiveness of the approach.

The rest of this chapter is structured as follows: we first discuss related work. In

section 6.3 we present the application that motivates this problem. Section 6.4 introduces the formal problem definition and a MIP formulation with the connectivity requirement. In section 6.6, we introduce two methods for handling the connectivity requirement, including a cutting plane approach. Section 6.7 contains the experiments after which we conclude.

6.2 Related work

There are several studies that apply mathematical programming to clustering and graph partitioning problems (Hansen and Jaumard 1997). In addition to integer linear programming, semidefinite programming (Armbruster et al. 2008; Lisser and Rendl 2003), and quadratic programming (N. Fan and Pardalos 2010) formulations have also been used for solving these problems. Techniques such as branch-and-cut (Ferreira et al. 1998; Grötschel and Wakabayashi 1989) and branch-and-price (Mehrotra and Trick 1998; X. Ji and Mitchell 2007) have been used to improve the performance.

There are several variants of the graph partitioning problem. When the underlying graph is a complete graph, this problem is sometimes called the *clique partitioning* problem. In most variants of the graph partitioning problem, the edges, nodes, or both are weighted. The number of clusters can be specified by the user or they can be decided by the algorithm. The two most prominent types of objective functions are 1) the total weight of the edges that have endpoints in different clusters and 2) the total weight of the edges that have endpoints in the same cluster. Depending on the meaning of the weights, these functions are minimized or maximized. In either case, these objectives are meant to increase the homogeneity of the clusters.

Several types of constraints are common to the graph partitioning problem. The most widely used constraint is the *balance* constraint that requires the number of nodes in all clusters to be almost equal (Labbé and Özsoy 2010). Other types of constraints include constraints on the size of clusters (N. Fan and Pardalos 2010), and constraints on the total weight of nodes in a cluster (Ferreira et al. 1998).

The main difference of our problem with existing ones is that we have two graphs, where the edges of the co-occurrence graph are weighted, but the goal is to minimize their total value. The connectivity of the nodes in the interaction network are required and must hence be added as hard constraints. The graph partitioning problem of (Benati et al. 2017) also requires such constraints. However, they do not assume that the number of clusters is given, and their encoding of the clustering problem is quite different from ours.

6.3 Motivating application

In cancer research, tumor tissue is collected from patients for further study. Such a tumor is basically a cell in which one or more genes have mutated. Normally that cell would be destroyed by the immune system, but in case of a tumor that cell has managed to survive and may even be growing (out of control). Genes and mutated genes can be identified in a tumor by sequencing the DNA of the tissue. Nowadays, sequencing has become fairly commonplace, enabling the genome-wide measurement of mutated genes across large groups of cancer patients.

The key challenges when interpreting these data are to detect the (mutated) genes that affect the creation and development of cancer and to gain an understanding of their interaction. Initially, the main focus by the community was solely on the detection of key driver genes. However, there are typically many mutated genes making it challenging to detect rare ones with high statistical significance. For this reason, there is recently a rise in methods aiming to exploit the information contained in the human interaction network (Leiserson et al. 2015; Pulido-Tamayo et al. 2016). This network expresses which genes interact with each other. This can be used to verify that a set of genes interact with each other.

In our setting we start by considering a subgraph of the human interaction network containing only those genes and interactions that were identified as being highly relevant to breast cancer. Each node represents a gene or gene product, and each edge represents an interaction between a pair of nodes. The graph may consist of multiple connected components. It is our goal to separate distinct *pathways* from each of these components, where a pathway is a set of genes that interact with each other.

By incorporating such biological pathway information, a superior selection and understanding of genes and their interactions is possible. A difficult challenge however is how to determine that two genes are more likely to belong to the same or to a different pathway. Fortunately, it is known that the creation and growth of tumors follows a clonal *evolutionary* model. Following this model, it is considered unlikely that a tumor would disrupt and mutate different genes within a single pathway. Indeed, it is sufficient to disrupt one gene to disrupt the pathway, and hence evolutionarily there is little incentive to disrupt others as well. As a consequence, it is less common that multiple mutated genes of a single pathway are observed within the same patient. We can hence compute a co-occurrence penalty score for each pair of genes, based on the harm associated with the mutations and the number of patients for whom that pair was observed.

While obtaining pathways with a low penalty is important, one should not go

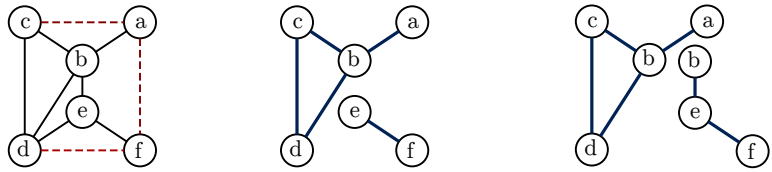


Figure 6.1: Left: A small subgraph extracted from the interaction network. Dashed lines represent the can-not-link constraints. Middle: the pathways obtained from non-overlapping clusters. Right: the pathways obtained from overlapping clusters.

as far as dividing the genes into small clusters. For this reason, we will aim to balance the size of the (smallest) pathway with the amount of penalty the pathways incur.

6.4 Problem and MIP formulation

From a computational point of view, the input to our problem are a set of genes and two graphs over those genes: an unweighted interaction graph and a weighted co-occurrence graph. We first review some basic graph concepts before formally defining the problem.

A graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. Each edge $e \in E$ is a tuple $(u, v), u, v \in V$ and the graph is *undirected* if the ordering of the edges does not matter. The graph is *weighted* if each edge $e \in E$ has a corresponding weight $w(e)$. A graph is *simple* if it does not have any loops or double edges. A graph is *connected* if there is a path between each pair of its nodes. The nodes on a path except the first and last ones are called the *intermediate nodes*. A *simple path* is a path with no cycles. Another important concept is that of an induced subgraph. Given a set of nodes $V' \subseteq V$, the *induced subgraph* of G on V' is a graph $G[V']$ that contains only the nodes in V' and only the edges of G that have both endpoints in V'.

Problem definition

Our problem is as follows: given a set of genes V, a simple undirected graph $G_i = (V, E_i)$ representing the interaction network and a simple weighted undirected graph $G_o = (V, E_o)$ with weight function w_o representing the co-occurence network. The goal is to partition V into k different groups $V_1, \ldots, V_k$, where k is assumed given. Each cluster induces a subgraph on the interaction graph G_i

and that subgraph *must be connected*. The quality of a clustering is determined by the size and co-occurence penalty of the clusters. The two are linearly combined using γ to obtain the following objective that must be maximized:

$$f(V_1, \ldots, V_k) = min_{c=1}^{k} |V_c| - \gamma \sum_{c=1}^{k} \sum_{e \in G_o[V_c]} w_o(e) \tag{6.1}$$

The first component represents the size of the smallest cluster while the second component represents the sum of weighted edges in the induced subgraph of the co-occurence graph G_o on the cluster V_c. γ can be used to balance the size component to the co-occurence penalty computed and is assumed given.

Mixed integer programming

We can formulate this problem as an integer linear program. The main decision variables in this formulation are those that determine the assignment of nodes to the clusters. We will use binary variables x_{ic} to indicate whether or not node i is included in cluster c. Each cluster must be assigned to exactly one cluster. This can be enforced by the following set of constraints:

$$\sum_{c=1}^{k} x_{ic} = 1 \quad \forall i \in \{1, \ldots, |V|\} \tag{6.2}$$

To model the first component of the objective in Eq. 6.1 we introduce an integer variable s which represents the size of the smallest cluster. The domain of s is $\{0, \ldots, \lfloor \frac{|V|}{k} \rfloor\}$. The following constraints ensure that s is smaller or equal than the size of each cluster.

$$s \leq \sum_{i=1}^{|V|} x_{ic} \quad \forall c \in \{1, \ldots, k\} \tag{6.3}$$

As s is included in the objective function it will be maximized and hence take the value of the size of the smallest cluster during optimisation.

For the second component, we introduce additional variables to model the edges in the induced subgraph $G_o[V_c]$ of each cluster. More specifically, we introduce binary variables y_{ijc} which are equal to one if and only if nodes i and j are both included in cluster V_c. To enforce this property, we add the following constraints to the model:

$$y_{ijc} \geq x_{ic} + x_{jc} - 1 \quad \forall c \in \{1, \ldots, k\}, \forall (i,j) \in E_o \tag{6.4}$$

When both x_{ic} and x_{jc} are equal to one, this constraint forces y_{ijc} to be equal to one. Otherwise, y_{ijc} can be either zero or one. However, as we will see, y_{ijc} is included in the objective function with a negative coefficient. Hence the optimization procedure will automatically fix y_{ijc} to zero in such cases.

Our formulation so far does not ensure that the nodes in each cluster are connected. For now assume that constraint $\mathsf{connected}(x_{1c}, \ldots, x_{|V|c})$ enforces the connectivity of cluster c. We will discuss the exact formulation of this constraint in the next section.

The complete model that we defined is hence the following, where $w_o((i,j)) = w_o(e)$ for $e = (i,j)$:

$$\text{maximize} \quad s - \gamma \sum_{c=1}^{k} \sum_{(i,j) \in E_o} w_o((i,j)) * y_{ijc} \tag{6.5}$$

s.t.

$$\sum_{c=1}^{k} x_{ic} = 1 \qquad \forall i \in \{1, \ldots, |V|\} \tag{6.6}$$

$$y_{ijc} \geq x_{ic} + x_{jc} - 1 \qquad \forall c \in \{1, \ldots, k\}, \forall (i,j) \in E_o \tag{6.7}$$

$$s \leq \sum_{i=1}^{|V|} x_{ic} \qquad \forall c \in \{1, \ldots, k\} \tag{6.8}$$

$$\mathsf{connected}(x_{1c}, \ldots, x_{|V|c}) \qquad \forall c \in \{1, \ldots, k\} \tag{6.9}$$

$$x_{ic} \in \{0,1\} \qquad \forall c \in \{1, \ldots, k\}, \forall i \in \{1, \ldots, |V|\} \tag{6.10}$$

$$y_{ijc} \in \{0,1\} \qquad \forall c \in \{1, \ldots, k\}, \forall (i,j) \in E_o \tag{6.11}$$

$$s \in \{0, \ldots, \lfloor \frac{|V|}{k} \rfloor\} \tag{6.12}$$

6.5 Extensions and improvements

We choose to use a generic discrete constraint solver to solve our non-traditional clustering problem. In the following we discuss three extensions to the above formulation that are possible thanks to the use of generic solvers and their ability to handle different types of constraints.

6.5.1 Overlapping clusters

It is well known that genes can occur in multiple pathways, and hence we may wish to allow for nodes to be included in more than one cluster. To apply this modification to our formulation, we only need to replace constraints (6.2) with the following inequalities:

$$\sum_{j=1}^{k} x_{ij} \geq 1 \qquad i \in \{1, \ldots, |V|\} \tag{6.13}$$

6.5.2 Breaking symmetries

Clustering problems often have an inherent symmetry which is due to the fact that the labels of clusters are arbitrary. This means that for each solution, there are $k!$ equivalent solutions that only differ by the cluster labels. We can strengthen our formulation by breaking these symmetries. (Sherali and Desai 2005) suggests two measures to reduce these symmetries: 1) assign label 1 to the cluster that contains node 1. 2) assign labels to other clusters in increasing order of their sizes. This translates to the following constraints:

$$x_{11} = 1 \tag{6.14}$$

$$\sum_{i=1}^{|V|} x_{ic} \leq \sum_{i=1}^{|V|} x_{i(c+1)} \qquad c \in \{2, \ldots, k-1\} \tag{6.15}$$

These constraints do not eliminate all symmetries (especially in the case of overlapping clusters) but still lead to improvements in performance in practice.

6.5.3 Obtaining the set of Pareto optimal solutions

In the above formulation we used the commonly employed method of reducing a bi-objective optimisation problem to a single-objective one through the use of a balancing parameter γ. An alternative solution is to use a bi-objective optimisation approach to compute the set of *Pareto optimal* solutions. A solution of a bi-objective optimization problem is Pareto optimal if there is no other solution with a better quality with respect to both objectives. Obtaining the set of Pareto optimal solutions for other types of bi-objective constrained clustering has been studied before (Dao et al. 2017; Guns, Dao, et al. 2016). As in that work, we use a method based on the ϵ-constraint algorithm (T'Kindt

Algorithm 4 Computing the set of Pareto optimal solutions

```
1: P ← ∅                                   ▷ the set of Pareto optimal solutions
2: m ← 0                                   ▷ Minimum size of the previous solution
3: repeat
4:     solution ← MinimizePenalties(V, G_i, G_c, k, m)
5:     P ← P ∪ solution
6:     m ← size of the smallest cluster in solution
7: until no solution was found
8: return P
```

and Billaut 2006) to obtain the Pareto optimal solutions. To do this, we remove the size component from the objective leaving only the co-occurence part. Then we iteratively solve this modified problem, each time adding a constraint such that the size of the smallest cluster is larger than that found in the previous iteration (m):

$$\sum_{i=1}^{|V|} x_{ic} > m \quad \forall c \in \{1, \ldots, k\} \tag{6.16}$$

The approach is depicted in Algorithm 4.

6.6 Enforcing connectivity

The remaining issue to work out is how to represent the $\mathsf{connected}(x_{1j}, \ldots, x_{|V|j})$ constraint. We investigate two different approaches.

For a cluster to be connected, there should exist for each pair of points belonging to the cluster at least one path between these two nodes such that all nodes on this path also belong to the cluster. In section 6.6.1 we enforce this condition by explicitly enumerating all simple paths between each pair of non-adjacent nodes in the graph and adding variables and constraints for these paths. However, the total number of paths can be exponential leading to very large models to solve.

A way to avoid having to ground out constraints for all the possible paths is to incrementally add only those constraints needed. The *cutting plane* algorithm is a method that allows this exactly. In section 6.6.2 we introduce another formulation for the connectivity constraint based on node-cut sets, which also has a worst-case exponential number of constraints but which lends itself well to an incremental cutting plane method.

6.6.1 Enumerating all simple paths

Consider a simple path between the nodes u and v. Let I denote the set of indices of the intermediate nodes on this path. Assume that binary variable y_c indicates that all these nodes belong to cluster c. A standard translation of the relation $(y_c = 1) \Leftrightarrow \wedge_{i \in I}(x_{ic} = 1)$ to linear constraints gives the following inequality:

$$0 \leq \sum_{i \in I} x_{ic} - |I| y_c \leq |I| - 1$$

In general, let P_{uv} denote the set of all simple paths between nodes u and v in the interaction graph. For a path $P_r \in P_{uv}$, let I_r denote the set of indices of intermediate nodes of P_r. We introduce binary variable y_{rc} to indicate that all these nodes are assigned to cluster c. This relationship is enforced by the following constraints:

$$0 \leq \sum_{i \in I_r} x_{ic} - |I_r| y_{rc} \leq |I_r| - 1$$

$$\forall u, v \in V, (u, v) \notin E, \forall r \in \{1, \ldots, |P_{uv}|\}, \forall c \in \{1, \ldots, k\} \quad (6.17)$$

Finally, to enforce the condition $(x_{uc} = 1 \wedge x_{vc} = 1) \Rightarrow \vee_r (y_{rc} = 1)$, we add the following constraints to the model:

$$x_{uc} + x_{vc} - 1 \leq \sum_{r=1}^{|P_{uv}|} y_{rc}$$

$$\forall u, v \in V, (u, v) \notin E,$$

$$\forall c \in \{1, \ldots, k\} \quad (6.18)$$

These constraints ensure there exists at least one path in the interaction graph between every two nodes in the same cluster.

6.6.2 A cutting plane approach

In the cutting plane method two steps are iteratively repeated: 1) A model that includes only a subset of the constraints is solved. 2) A constraint that is violated by the current solution, a cut, is added to the model. These steps are

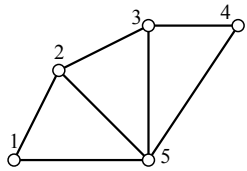


Figure 6.2: The sets {2, 5} and {3, 5} belong to $\Gamma(1,4)$, but the set {2, 3, 5} does not because it is not minimal.

repeated until no constraint is violated. To use the cutting plane algorithm, we need an oracle that given an assignment $\mathbf{x}$ can check if $\mathbf{x}$ satisfies all constraints and if not, finds a constraint that is violated by $\mathbf{x}$. Since in the latter case the added constraint separates $\mathbf{x}$ from the feasible region, the problem solved by the oracle is called the *separation* problem. In a *branch and cut* algorithm, cutting planes are added throughout the branch and bound tree.

A cutting plane approach with the formulation of connectivity from the previous section would require us to add both variables (the y_{rc} ones) and constraints (6.17) and (6.18) for each cut. Instead, we adopt the approach of (Carvajal et al. 2013) which defines the connectivity constraints in terms of *node-cut sets*. The advantage is that no extra variables need to be introduced, and that between two nodes the connectivity can be broken incrementally with individual constraints. The following definition and theorem are taken from (Carvajal et al. 2013).

Definition 6 (Node-cut set)**.** *Given nodes $u, v \in V$ that are not adjacent ($(u,v) \notin E$), a set of nodes $S \subseteq V \setminus \{u,v\}$ is a* node-cut set *separating u and v (or simply a uv-*node cut*) if all paths between u and v intersect S.*

There is hence at least one node from every path between u and v in the uv-node cut. A uv-node cut is *minimal* if it is not a uv-node cut after removing any of its nodes. For a pair of non-adjacent nodes u and v, we denote by $\Gamma(u,v)$ the set of all minimal uv-node cut sets. Figure 6.2 shows examples of minimal node-cut sets. The following theorem relates the connectivity of a graph to its minimal node-cut sets.

Theorem 3. *Given $U \subseteq V$ and a pair of non-adjacent nodes $u, v \in U$, there exists a path between u and v in the subgraph induced by U if and only if all uv-node cuts S are such that $S \cap U \neq \emptyset$.*

For every uv-node cut S this theorem states that to ensure that cluster c is connected, at least one node in the node cut must belong to the cluster:

$$(x_{uc} = 1 \wedge x_{vc} = 1) \Rightarrow \vee_{w \in S}(x_{wc} = 1)$$

By translating this condition into linear constraints we can formulate the constraint $\mathsf{connected}(x_{1c}, \ldots, x_{|V|c})$ as follows:

$$\sum_{w \in S} x_{wc} \;\geq\; x_{uc} + x_{vc} - 1 \forall (u,v) \;\in\; V, (u,v) \;\notin\; E, S \;\in\; \Gamma(u,v) \quad (6.19)$$

The constraint set (6.19) contains an exponential number of constraints. Following the cutting plane method, we will iteratively add some of the violated constraints to the model. A common practice is to add one or some of the constraints *most violated* by the current solution in each iteration.

Given a solution $\mathbf{x}^*$ of the problem, let us denote the value of x_{ic} variables in this solution by x^*_{ic} and the vector of variables corresponding to cluster c by $\mathbf{x}^*_c$. In this respect, a first observation is that $\sum_{w \in S} x^*_{wc} \geq 0$ is always true as the x^* are Boolean variables. Hence, the constraint can only be violated if $x^*_{uc} + x^*_{vc} - 1 > 0$, that is, if both variables belong to the same cluster. If that is the case, then the constraint can only be violated if $\sum_{w \in S} x^*_{wc} = 0$, that is, if none of the nodes in the cut set are in the cluster. If no such constraint can be found, then the connectivity constraint is satisfied.

The same principle can be used on real-valued solutions (as computed by the MIP solver when solving the linear relaxation). The most violated constraint for a non-adjacent pair (u, v) is now the constraint for which $x^*_{uc} + x^*_{vc} - 1 > 0$ and that minimizes $\sum_{w \in S} x^*_{wc}$. To add a cut of (u, v) in the cutting plane algorithm, the goal is hence to find the node-cut set S^*,

$$S^* = \underset{S \in \Gamma(u,v)}{argmin} \sum_{w \in S} x^*_{wc} \quad (6.20)$$

It is shown in (Carvajal et al. 2013) that the solution for equation 6.20 can be computed efficiently: If we use x^*_{ic} as the capacity of node i, the separation problem reduces to finding the minimum capacity node cut separating u and v. This problem can be solved using any standard *min-cut* algorithm. A summary of the cut generation procedure is presented in algorithm 5, where for each cluster the most violated constraint among its node-pairs is added.

6.7 Experiments

We ran experiments on quad-core Linux machines with 32 GB of memory. We implemented our branch-and-cut algorithm using the Python interface of

Algorithm 5 The cut-generation procedure

$\mathcal{C} \leftarrow \emptyset$ ▷ Set of constraints to add
for $c \in \{1, \ldots, k\}$ **do**
  for u, v such that $(u, v) \notin E_i$ and $x^*_{uc} + x^*_{vc} > 1$ **do**
    $S^* \leftarrow$ min-cut$(u, v, \mathbf{x}^*_c, G_i)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\sum_{w \in S^*} x_{wc} \geq x_{uc} + x_{vc} - 1\}$
  end for
end for
Add constraints $\mathcal{C}$ to the model

Gurobi-7[1] and allowed it to use all 4 cores.

To solve the separation problem, we used the min-cut/max-flow algorithm from the *NetworkX-1.11* library (Hagberg et al. 2008). In order to use this algorithm, we first replaced each undirected edge by two opposite directed edges of infinite capacity. Then, we replaced each node v by two nodes $v_{\text{in}}, v_{\text{out}}$ connected by two opposite edges, with capacities equal to the capacity of v. We obtain the minimum node cut separating u and v by computing the minimum cut between u_{out} and v_{in} in this graph.

We followed the recipe of (Carvajal et al. 2013) for adding the cuts: cuts to the linear relaxation were only added at the root node of the branch-and-cut tree. Moreover, when adding cuts to the relaxation, we monitored the change in the value of the objective function. If this value improved less than 5% in 10 consecutive rounds, we stopped adding cuts. Outside the root node, we only add cuts when an integer solution is found. We normalized the two components of the objective function. We multiplied the size variable by $1/n$ in overlapping clustering and by k/n in non-overlapping clustering. We divided the second component by the sum of edge weights.

The graphs used in our experiments were extracted from the HINT+HI2012 protein-protein interaction network (Das and Yu 2012; Yu et al. 2011). We consider this selection of subgraphs representative of the graphs encountered in our application domain. Table 6.1 highlights some of the properties of our selected instances. Several of these instances contain a very low number of edges compared to nodes. As a direct result of this, they contain a very small number of paths. We have also selected a few graphs that contain more paths. These graphs also have a number high-degree nodes. Such hubs are fairly common in scale-free networks, of which protein-protein interaction networks are a common example.

[1] www.gurobi.com

Table 6.1: Instance properties

instance	#simple paths	#nodes	#edges
200_1000	93	40	46
200_1250	1508	52	68
250_750	45	30	32
250_1000	1040	69	81
300_750	59	44	46
350_500	12	13	12
350_750	73	58	60
450_750	105	90	92

6.7.1 Results and discussion

Scalability of the two approaches

We compare the runtime of the model formulation using all paths (ENUM) with that of using branch-and-cut (BNC), for a number of datasets and k values and averaged over $\gamma \in \{0.1, 0.25, 0.33, 0.5, 1, 2, 3, 4, 5, 10\}$. Table 6.2 shows the results. In each instance, one of the algorithms outperforms the other one. This divides the instances into two groups. Instances for which enumerating all paths has a better performance are those which have a similar number of nodes and edges (see table 6.1). As a result, the total number of simple paths in these graphs is small and the optimization model has a reasonable size. On the other hand, for two of the instances the branch-and-cut method provides a clear advantage. The total number of simple paths for these instances is considerably larger than the others. Hence, for these instances the extra effort for solving the separation problem pays off.

Impact of γ

The γ parameter is a way of balancing the minimum size of the clusters with the total co-occurrence penalty. We showcase the effect of the parameter on these two components of the objective in Figure 6.3 for instance 250_750. As γ approaches 0 most weight is given to the size leading to increasing larger minimum clusters but also a sharp incline in the total co-occurence penalty. For too high γ values, ≥ 4 in this case, the size of the clusters can drop to values of 1 or 2, which are not meaningful. In this case, a γ between 1 and 3 seems most sensible.

Pareto optimal set

As explained in Section 6.5.3 one can also use generic solvers to compute the Pareto-optimal set directly instead of having to determine a γ parameter. In Figure 6.4 for the non-overlapping setting we can see that a few smallest cluster values are skipped because they are not optimal. We also observe that there is a gradual increase in total violations as the minimum size increases, with an increase in incline near the end (there are 30 genes in this dataset). This can be used to select an appropriate solution among the Pareto optimal ones.

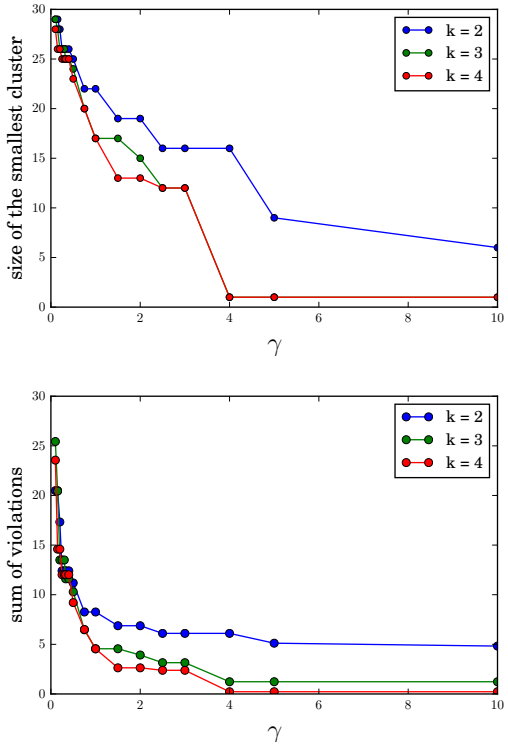


Figure 6.3: Impact of γ on smallest cluster size (left) and total co-occurrence penalty (right) for instance `250_750`.

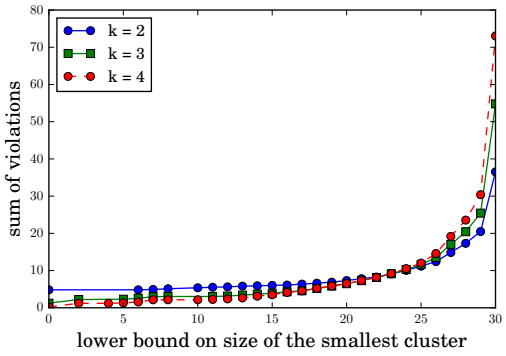


Figure 6.4: The Pareto optimal set for overlapping clustering on instance `250_750` with three values for number of clusters.

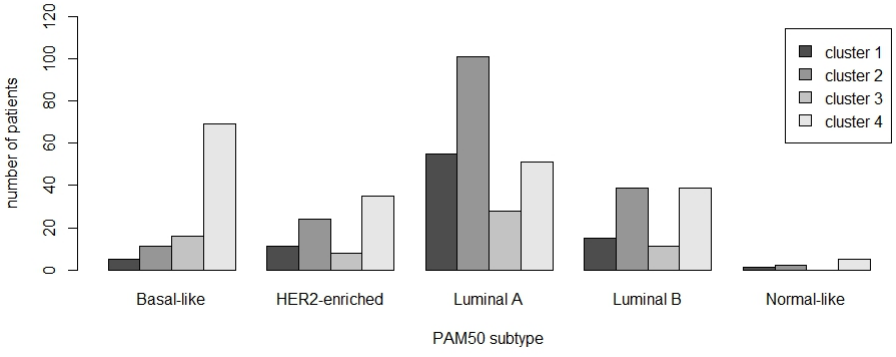


Figure 6.5: Number of patients which are a member of each cluster, per PAM50 subtype.

Table 6.2: Average runtimes of overlapping and non-overlapping clustering by enumerating all simple paths (AllPaths) and branch and cut (BnC). Timed-out experiments are counted as 600 seconds (–).

		overlapping		*non-overlapping*	
instance	k	*AllPaths*	*BnC*	*AllPaths*	*BnC*
200_1000	2	**0.830**	12.320	**0.319**	13.398
	3	**3.211**	16.255	**4.474**	15.007
	4	**14.846**	28.714	**15.578**	52.262
200_1250	2	225.735	**43.740**	113.210	**38.849**
	3	582.541	**52.950**	298.390	**72.643**
	4	–	**188.287**	595.930	**205.454**
250_1000	2	487.222	**78.243**	370.898	**69.481**
	3	587.700	**121.677**	–	**217.079**
	4	–	**206.952**	–	–
250_750	2	**0.166**	4.058	**0.025**	3.873
	3	**0.321**	6.433	**0.387**	3.965
	4	**0.737**	10.564	**0.698**	7.374
300_750	2	**0.333**	12.232	**0.209**	10.546
	3	**0.761**	20.365	**0.812**	16.924
	4	**1.728**	28.267	**3.598**	50.836
350_500	2	**0.004**	0.093	**0.003**	0.151
	3	**0.006**	0.129	**0.021**	0.257
	4	**0.007**	0.181	**0.039**	0.211
350_750	2	**0.503**	44.842	**0.591**	20.130
	3	**1.971**	113.311	**2.016**	58.417
	4	**7.892**	323.188	**9.841**	309.998
450_750	2	**1.560**	115.541	**1.334**	89.672
	3	**10.686**	361.855	**12.655**	320.706
	4	**90.516**	430.954	**75.395**	–

Biological validation

Research increasingly shows that a single type of cancer, for example breast cancer, is not one homogeneous disease. Instead, it can be divided in different *subtypes* that display different harmful effects, and in turn require different treatments. In order to validate the results of the clustering approach, we looked at the PAM50 tumor subtype classification (Parker et al. 2009), which was previously published for the patients in our records by the Cancer Genome Atlas Network (Koboldt et al. 2012). PAM50 subtypes are determined by looking at the expression levels of 50 specific genes from a breast cancer sample in order to assign an *intrinsic* subtype to the patient's tumor. As this is

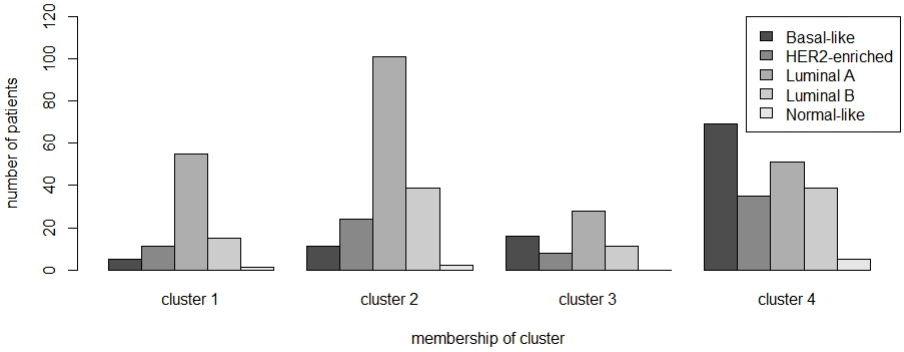


Figure 6.6: Number of patients with specific PAM50 subtype, per cluster.

based on expression data rather than the mutational gene data used in our clustering approach, correlation between the PAM50 subtypes and the identified clusters would be interesting as this would enable us to (partially) subtype breast cancer tumors based on gene mutation data. Furthermore, since the PAM50 tumor subtype classification has prognostic significance, correlation between the clusters and the PAM50 tumor subtypes would largely validate our clustering approach as being biologically relevant in a real-world cancer setting. We analyzed the data by partitioning it into 4 non-overlapping clusters with a γ of 0.5. We then assessed every patient in our dataset for membership of one or multiple clusters. A patient was considered a member of a specific cluster when that patient had at least one deleterious mutation (defined as a PHRED-scaled CADD score of at least 20 (Kircher et al. 2014)). The results of this analysis are depicted in Figure 6.5 and Figure 6.6. We performed a χ-square goodness of fit test on the distribution of the basal-like PAM50 subtype from Figure 6.5 and on both the distributions of cluster 1 and cluster 2 from Figure 6.6 to test whether they deviated from a random assignment of patients to clusters. All three distributions deviated significantly from the random case (Basal-like: $\chi - squared = 103.48$, $df = 3$, $p - value < 2.2e - 16$; cluster 1: $\chi - squared = 15.507$, $df = 4$, $p - value = 0.003757$; Cluster 2: $\chi - squared = 22.388$, $df = 4$, $p - value = 0.0001678$). Based on these results, and the observations from figure 6.5 and figure 6.6, two interesting conclusions could be drawn: 1) patients with a basal-like PAM50 tumor subtype were very likely to have a mutation in clustered pathway 4 and 2) patients with a mutation in pathway 1 or 2 were more likely to have a luminal A subtype. This shows that the identified gene clusters / pathways have at least some correlation with the

PAM50 subtypes and could thus be useful in patient subtyping and exploring subsequent treatment options, although more research would be needed to confirm this. As such, our subtyping method is able to generate meaningful biological results in a cancer subtyping setting.

6.8 conclusions and future work

Motivated by a problem in bio-informatics, we presented a novel graph clustering problem involving two graphs, a co-occurrence graph whose weighted edges are part of the objective function, and an interaction graph with hard connectivity constraints. We propose two methods for handling the (potentially exponential in number) connectivity constraints, one based on enumerating all simple paths and the other being a cutting plane approach. We also present a number of extensions such as a bi-objective Pareto optimisation method to balance minimum cluster size and total penalty. Computational experiments show the properties of the different proposed methods, and a validation experiment on a separate biological data source demonstrates the potential of our proposed approach.

Part III

Learning Taxi Passenger Demand

Chapter 7

Feature-based Taxi Request Prediction

In this chapter we investigate the problem of transportation demand prediction where the goal is to accurately predict where and when a transport need will occur. We focus in particular on the problem of taxi request prediction. We develop an approach for feature-based taxi request prediction that allows for the easy integration of additional features. We also introduce a new approach for analyzing and gaining insight in taxi demand by using non-negative matrix factorization. This chapter is based on the following work:

- Behrouz Babaki, and Anton Dries. *Feature-based Taxi Request Prediction.* (manuscript in preparation).

7.1 Introduction

Over the last few years, the study of transport management has gained a lot of attention in the political, economic and scientific communities. One of the important topics studied in this emerging field is the optimal use of existing infrastructure (e.g. roads) to accommodate the increasing demand with the aim to decrease economic losses caused by traffic jams and health and environmental issues caused by pollution.

In this context, dial-a-ride services (such as taxis, and services like Uber and Lyft) are an interesting candidate for research because (1) they are a major

contributor to the traffic in large urban regions, (2) they are demand-driven (unlike most public transport systems which are supply-driven), and (3) they are increasingly monitored digitally by means of GPS trackers.

Understanding the regularities in taxi trajectories not only gives insight about the taxi services, but also reveals the patterns of human mobility in the city. For example, one can identify the attractive regions by observing the popular destinations and study how they vary over time (Li et al. 2012; Yue, Zhuang, et al. 2009; Yue, Wang, et al. 2011; Chang et al. 2010). The information obtained through such studies can be used for improving transport management, such as determining the need for public transport or planning the locations of taxi stops (Moreira-Matias et al. 2013), or to support long-term decisions with respect to urban planning (Zheng et al. 2011).

We focus on the problem of predicting taxi requests from contextual features. The primary goal is to be able to predict taxi requests on regions for which there is no or limited historic data available. To the best of our knowledge, there are no techniques that can easily incorporate such features. We show that by using demographic information we obtain reasonable predictions for both known and unknown regions.

As an additional contribution, we show that non-negative matrix factorization can be a useful tool in the analysis of taxi data. It can offer insight in the behavioral patterns underlying the data and improve the interpretability of the prediction model by reducing the number of parameters.

This chapter is organized as follows. In Section 7.2 we give an overview of work related to travel demand forecasting. In Section 7.3, we give a formal definition of the problem of predicting taxi demand and in Section 7.4, we investigate traditional approaches to In Section 7.5 we introduce a new approach for predicting taxi requests based on demographic data and we evaluate our approach in Section 7.6. Finally, Section 7.7 concludes this chapter.

7.2 Related work

The problem of predicting taxi passenger requests has been studied before. Some of the existing methods assume a dependence between the demand at a certain timeslot and the observed demand in previous timeslots. All methods that are based on time-series analysis fall in this category. Davis et al. use a time-series model for prediction. They extend this method by grouping neighboring areas into clusters and combining the predictions for areas in each cluster (Davis et al. 2016). Several papers use different autoregressive integrated moving

average (ARIMA) models for predicting the taxi demand (Li et al. 2012; Li et al. 2012; Moreira-Matias et al. 2013). The dependence relations with previous observations can be also modeled as a Bayesian network: Li et al. start with a naive method that simply reports the observed count of passengers in the same timeslot in the past day. Then they extend this method to a simple Bayesian network (Li et al. 2012). One of the methods presented by Zhao et al. is an order-k Markov model (Zhao et al. 2016). Zhang et al. represent the relationship between the observed demand and hidden variables by a hidden Markov model (Zhang et al. 2014).

Some studies decompose the demand prediction problem into subproblems. In the work of Miller et al. there are two sources of uncertainty: the arrival of pedestrians and whether a pedestrian will make a request for a taxi trip. They model the arrivals by a Poisson distribution and the probability of making a request by a Bernoulli distribution (Miller and How 2017). Zhang et al. decompose the passenger demand into two components: the passengers who have just arrived, and those who are left behind from previous timeslots (Zhang et al. 2014).

Bayesian learning is also used for updating the distribution of demand continuously. In this method, the quantity of interest is modeled by a distribution with uncertain parameters. These parameters themselves follow a distribution that can be easily updated according to the observed values. Miller et al. use this method to update the arrival rate of pedestrians and the probability of receiving a trip request from a pedestrian (Miller and How 2017). Zhang et al. model the demand by a time-varying Poisson process which is continuously revised by Bayesian updating (Zhang et al. 2014).

Until recently, contextual features were rarely taken into account in prediction of taxi demand. However, two recent studies train predictive models that take such features as input. Zhao et al. train a neural network using these features: *temperature*, *precipitation*, *wind speed*, *day of week*, and *hour of day* (Zhao et al. 2016). Saadi et al. compare multiple machine learning algorithms: a single decision tree, bagged decision trees, random forests, gradient boosted trees, and neural networks. They use four types of features as input: *space-time*, *price*, *traffic condition*, and *weather condition* (Saadi et al. 2017). One important difference of our work with these studies is that we base our model on both contextual features and recent observations.

Predicting future demand can improve the quality of solutions obtained for optimization problems such as vehicle routing. There exist several studies on routing with uncertainty about the future demand (Ferrucci et al. 2013; Ichoua et al. 2006; Bent and Hentenryck 2004). Most of these studies assume that the requests are known in advance (e.g. for a package delivery service) or follow

a known distribution (stochastic). A notable exception is the work of Bent et al. where they discretize the demand into a number of levels and train a hidden Markov model for predicting the demand (Bent and Hentenryck 2005). Ferrucci et al. learn fixed-rate Poisson models for reach region and period to predict the demand (Ferrucci et al. 2013). Ge et al. solves a routing problem using historical taxi trip records of San Francisco. They cluster pickup points of high-performance drivers into clusters and later use the centroids of these clusters for recommending pickup points to drivers (Ge et al. 2010). Miao et al. design a tractable dispatching algorithm which takes the uncertainties in passenger demand prediction into account. They evaluate their method on four years of trip data of New York and report improvements in the supply-demand ratio and idle driving distance (Miao et al. 2016).

Analysis of historical data of taxi trips has been used for other tasks than predicting the future demand: Some studies use the trip records of taxis to identify spatial clusters based on taxi services in multiple periods during the day (Deng and M. Ji 2011; Yue, Zhuang, et al. 2009). Li et al use records of taxi trips to find regions with the highest pickup rates (Li et al. 2012). Chang et al use clustering techniques on the trip data to discover the location of landmarks in the city (Chang et al. 2010). Yuan et al use historical records of taxi trajectories to train a probabilistic model for computing the probability of picking up the next passenger. They use this model to recommend locations to both drivers and passengers (J. Yuan et al. 2011).

7.3 Problem setting

The problem of predicting taxi requests is typically modeled using the following three components:

- a time frame T (e.g. one day)
- a partitioning P of the time frame (e.g. 15-minute intervals)
- a set of geographical regions R (e.g. a rectangular grid on the map)

The request model describes the distribution of requests within one time frame (e.g. one day) and can then be used to predict the number of requests for the regions in R and the periods in P of another time frame (e.g. the next day). For each region $r_i \in R$ and each time interval $p_j \in P$ the distribution of the number of requests occurring in the given region at the given time is modeled.

Existing approaches to modeling this problem can be characterized according to three properties:

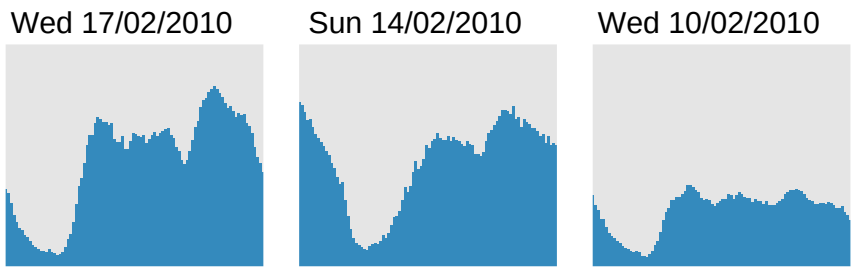


Figure 7.1: Distribution of taxi rides for three days in February 2010: a regular Wednesday (17/02), a regular Sunday (14/02) and a Wednesday during a blizzard (10/02).

1. To what extent the model uses peripheral features;
2. How specific or general the model is; and
3. How recent is the data that the model uses for making predictions.

We will now explain each of these properties in more detail.

7.3.1 Exploiting peripheral features

The problem formulation above focuses on the effect of two factors on the distribution of requests, namely the area and time interval. However, not all time frames are identical. The distribution of trips on a Wednesday can be significantly different from the distribution of trips on a Sunday, or trips can be influenced by the weather as illustrated in Figure 7.1. The distribution is influenced by external factors, that can often be described using additional features, such as calendar, weather, and demographic information. A factor that distinguishes different models is the extent to which they incorporate such peripheral information.

7.3.2 Specificity of the model

A common practice in existing methods for predicting the requests is to partition the feature space in advance, and develop one model for each partition. The simplest example of this practice is to develop one model for each geographical region. More generally, one can train a separate model for each combination of

values of features of interest. This corresponds to adding an extra dimension to the parameter space, and parameters can be noted as $\lambda_{ij}^{(d)}$ where $d \in D$, the set of possible feature combinations. Obviously, this quickly leads to an explosion in the number of parameters and, consequently, the amount of data required to accurately learn a model, because in order to obtain an accurate model, we need sufficient data for each of the parameters. This means that the number of trips in each cell (region-period) should be high enough. For example, if we have 200 regions, 15-minute intervals and a time frame of one week (for a total of 672 periods), we have $200 \times 672 = 134\,400$ parameters. Even if we make the (unrealistic) assumption that the trips are distributed evenly over the region/period cells, then for a mere 10 data points per parameter, we would require more than 1 million data points. When the trips distribution is skewed or more features are used, this number can rise quickly.

The restriction that we can only predict requests for regions and combinations of features for which we already have historic data, limits the applicability of the traditional approaches. Indeed, a model that uses individual, independent parameters $\lambda_{ij}^{(d)}$ can't make any predictions for combinations of i, j and d that do not appear in the training data.

A completely different approach is to train one general model for making all predictions. In this approach, instead of partitioning the feature space, feature values are provided as input data to the model. For example, instead of training one model for each geographical region, the demographic information of regions is provided as features to the model. This enables the model to generalize to new combinations of feature values that do not occur as such in the training data. Using this approach, we can, for example, make predictions in an area for which we have no historical data of requests based purely on its demographic information.

7.3.3 Recency of the data

The last factor concerns the recency of the information that we assume to be available at prediction time. More formally, at the time that we predict the requests for time interval p_t, how much information is available about the observed requests in the intervals $p_{t-1}, p_{t-2}, \ldots$. Some models heavily rely on the availability of very recent data. While exploiting such information can enhance the accuracy of prediction, assuming that such information is always available can be restrictive. On the other hand, as GPS technologies are more widely used in transportation services, over time this restriction might become less severe.

7.4 Traditional approaches

In this section we summarize the traditional methods for predicting requests for a transportation service. We group these methods into two classes: Poisson processes and time-series models.

7.4.1 Poisson processes

A Poisson process is a joint distribution of variables $X_1, \ldots, X_t$ such that:

$$P(X_t) = \frac{e^{-\lambda(t)}\lambda(t)^{X_t}}{X_t!} \tag{7.1}$$

The dependence between observations is encoded through the rate function $\lambda(t)$. An advantage of using Poisson distributions is that the rate parameter λ coincides with the most likely value of the distribution. Moreover, we can efficiently learn the parameter λ from the data by averaging the observed counts.

The problem of predicting requests is most commonly described as a time-space Poisson process (Ferrucci et al. 2013; Bent and Hentenryck 2005), that is, the requests in region r_i in time interval p_j are distributed according to a Poisson distribution with rate λ_{ij}. The model describes the distribution of requests over the time-frame (e.g. a day). If we want to predict requests from region r_i (e.g. near the airport) at a time p_j (e.g. between 8:00 and 8:15) in the next time-frame (i.e. the next day), we can simply look up the rate λ_{ij} and determine the distribution.

In a time-space Poisson model, it is assumed that all time frames are identical. As pointed out earlier, this assumption is not very realistic. To overcome this, one can include dimensions other than *time* and *space* into the design of the model. Moreira-Matias et al. (Moreira-Matias et al. 2013), for example, noticed a significant difference in the number of requests in different days of the week. To take this factor into account, they added *day-of-week* as an additional dimension to their model.

In principle, this approach can be extended to including other factors such as weather information. However, this requires training one specific model for each combination of features of interest. As we already stated in section 7.3.2, this is only practical if we have a few number of features.

Prediction using a Poisson process is simply performed by looking up the corresponding rate parameter. Hence there is no need for information about recently-observed requests at prediction time.

7.4.2 Time-series analysis

Moreira-Matias et al. (Moreira-Matias et al. 2013) use an ARIMA model for predicting the requests. An ARIMA model is a linear function of p past observed requests and q random error terms. The pair (p, q) is called the *order* of the model. At prediction time, one should provide the observed values of requests in the past p intervals. This means that ARIMA models require very recent information for making predictions.

An ARIMA model only involves past observations and error terms. To consider the differences between the areas, or other influencing factors, one needs to train multiple specific ARIMA models. For example, Moreira-Matias et al. (Moreira-Matias et al. 2013) trained one ARIMA model for each area.

In our work, we also propose models that exploit the information about previous observed requests. However, our model assumes a nonlinear relation between past observations and the target value. Moreover, we include other influencing factors as additional features in our models.

7.5 Feature-based taxi prediction

Traditional approaches are not scalable to many features because they require a model for each combination of feature values. In our approach, we use a global, feature-based model that can represent all parameters in a single model. We use existing learning algorithms to construct such a model, which are able to freely decide which parameters can be grouped together based on available data and can make predictions on combinations of feature values that do not occur in the training data. This global model represents a function $\Lambda(x)$ where x is an arbitrary vector of features, instead of representing the model as a fixed number of rates $\lambda_{ij}^{(d)}$. This function computes the most likely number of requests for a region and time with the characteristics given in x, which corresponds to the rate of the Poisson distribution.

With regard to features, we consider three types: time-specific (such as time of day, day of week, weather, etc.), region-specific (such as population, number of restaurants, etc) and recent observations (i.e. observed requests for previous time frames).

In this section we describe three techniques that can be fitted in this framework. First, we show that the splitting-based approach mentioned in section 7.4.1 can be described in this framework. Next, we introduce two new approaches based on regression.

7.5.1 Feature-based Poisson processes

In time-space Poisson processes, the only supported features are region identifier i, and period identifier j. To include features in the model, we can add a feature-based partition identifier d to the index pair (i, j). The partition identifier can be used to split the model based on feature values, for example, to learn one model for weekdays and another for weekends.

The function $\Lambda(i, j, d)$ corresponds to a lookup in a table that contains a rate $\lambda_{ij}^{(d)}$ for each triplet (i, j, d). These rates can be obtained by computing the average of a given set of data points $(i, j, d, count)$.

7.5.2 Direct prediction

In the first approach, we use a regression model to directly predict the rate. The input data contains a row for each combination of region, period and time-frame that occurs in the training data. The columns are region-specific features (e.g. population) and time-specific features (e.g. weather, time of day, day of week). The model is built on the features and predicts the count.

As our regression model, we use gradient boosted regression trees (GBT). Boosting methods fit additive models for predicting a target value: The learned function has the form $f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$. The functions $b(x; \gamma_m)$ are called *base learners* and are parameterized by vector γ_m. A possible base learner for the regression task is a regression tree. The regressoin tree itself has an additive form:

$$b(x; \{c_j, R_j\}_1^J) = \sum_{j=1}^{J} b_j \mathbb{1}(x \in R_j) \tag{7.2}$$

where $\{R_j\}_1^J$ are the regions in the input space corresponding to J terminal nodes of the tree, $\{c_j\}_1^J$ are the values predicted at terminal nodes, and $\mathbb{1}(.)$ is the indicator function. The tree is equivalent to the set of rules $x \in R_j \Rightarrow b(x) = c_j, \forall 1 \leq j \leq J$.

Boosting methods add base learners in several stages. In each stage a new base learner is added to the model to address its shortcomings. Traditional boosting methods like AdaBoost, represent the shortcomings of the current model by classification or regreesion error Freund and Schapire 1997. In contrast, gradient boosting method identifies the shortcomings of current model in terms of the gradient of loss function J. H. Friedman 2001.

Tree-based approaches have the ability to adapt the level of detail of a prediction based on the amount of available data in certain regions of the feature space. The advantage of using such a global model is that the learning algorithm can automatically determine the optimal number of parameters and decide which regions, periods and features are most relevant or are similar enough to be merged. By applying gradient boosting we can significantly increase their predictive performance (Freund and Schapire 1997; J. H. Friedman 2001).

If we assume that recently materialized requests are known at prediction time, we can include these observations as additional features. This approach has two advantages over ARIMA: (1) features other than past observations are included in the model, and (2) the model can encode complex nonlinear relationships between the features and the number of requests.

In this approach, the training algorithm is in charge of partitioning the feature space. However, it is also possible to perform some of this partitioning manually. For example, one can drop the region-specific features from the feature set and train one specific model for each region. Note that other features (i.e. the time-specific features and previous observations) are still used for making the predictions.

7.5.3 Decomposition-based prediction

The previous approach has the disadvantage that it does not reduce the number of potential parameters. This also makes the method's outcome hard to interpret.

In order to reduce the number of parameters, we first apply a transformation on the data. The goal of this transformation is to find a number of principal components that each describe a different aspect of the data. To this end, we will use non-negative matrix factorization (NMF) (Lee and Seung 2000; Lee and Seung 1999). This approach is commonly used in topic modeling but to our knowledge has never been applied to data of this kind.

NMF decomposes a matrix M of size $m \times n$ into a component matrix B of size $c \times n$ and a weight matrix W of size $m \times c$ such that

$$M \simeq W \cdot B$$

and c is a preset number of components. The advantage of NMF over other PCA techniques is that each value in the matrices B and W is ≥ 0. This means that the components can still be interpreted and visualized as distributions of request counts and that each of the components has an additive contribution to the distribution of requests. This makes it easier to interpret the result.

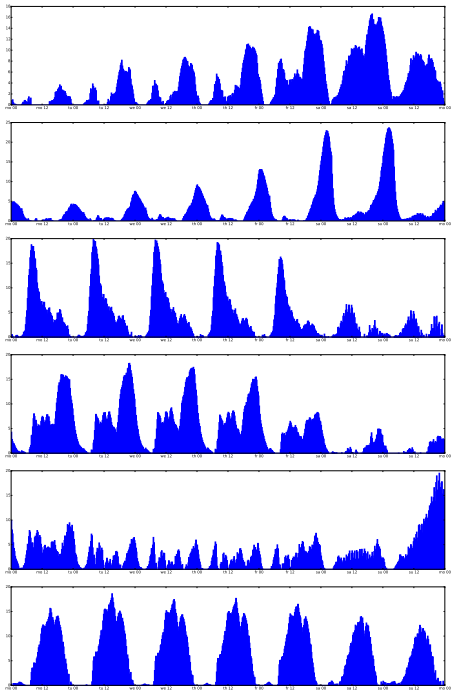


Figure 7.2: Components discovered by NMF ($c = 6$).

In our approach, the matrix M corresponds to a matrix Z_{ij} where each row represents the histogram of requests for a given region r_i over the course of one week with 15-minute bins p_j. Each component found by NMF can be interpreted as such a histogram as well describing a subset of the requests. Figure 7.2 illustrates that NMF appears very successful in finding meaningful base components.

The matrix W describes, for each region, the contribution of each of the base components to the activity in that region.

Instead of directly predicting the $m \times n$ matrix representing the number of requests, we can now predict the $m \times c$ matrix of weights.

Assume we are given a request matrix Z_{known} for a given set of periods and regions, a corresponding matrix of features F_{known}, and a set of features for a different set of periods and regions $F_{unknown}$. The request matrix for the second set of regions can then be predicted using the following four steps:

1. Compute weights and base components using NMF:

$$W_{known} \cdot B = Z_{known}$$

2. Train a regression model based on features:

$$\mathcal{M} = regr(F_{known}, W_{known})$$

3. Predict the weights of the unknown data using the regression model:

$$W'_{unknown} = \mathcal{M}(F_{unknown})$$

4. Construct the request matrix:

$$Z'_{unknown} = W'_{unknown} \cdot B$$

The function $\Lambda(x)$ is thus defined as

$$\Lambda(x) = \mathcal{M}(x) \cdot B.$$

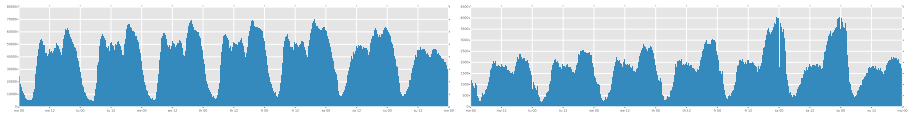
7.6 Evaluation

In this section, we empirically evaluate our approaches. We give a short description of the data and evaluation metric we use, and we look at the effect of using decomposition. Finally, we evaluate the predictive power of our approach both on predicting known regions and unknown regions.

7.6.1 Data

For our experiments we use two publicly available datasets of taxi trips for New York[1] and San Francisco[2]. We base our results primarily on the New York dataset due to the low quality of the San Francisco data witnessed by a high variability in number of recorded trips and large chunks of missing data. Since Gradient boosting is inherently capable of handling missing values, we did not need to remove or replace the missing counts in the data. We learned the parameter of Poisson distributions by taking the corresponding average excluding the missing entries.

[1] http://nyc.gov/html/tlc/html/about/trip_record_data.shtml
[2] http://stamen.com/work/cabspotting



(a) New York (26 744 433 trips) (b) San Francisco (1 130 078 trips)

Figure 7.3: Distribution of all trips between 5 April and 30 May 2010. Trips are aggregated to show distribution over a one week period.

As region features, we use data from the LEHD department of the US Census bureau[3]. We therefore use census tracts as regions. These datasets describe characteristics of working people. For each region there are two sets of features, one set describes the job characteristics of the people living in that region (RAC), the other set describes the job characteristics of people working in that region (WAC). These characteristics include total number of jobs, number of people working in given sectors (e.g. healthcare, entertainment, food services, etc), earning certain wages, having a certain education level, etc.

For weather information, we collected the measurements from multiple weather stations using MesoWest API[4]. The weather information is available for each 15-minute interval. For each attribute, we use the average over measurements of multiple stations and hence it is the same for all regions within the same city. The attributes consist of temperature, humidity, precipitation and wind speed.

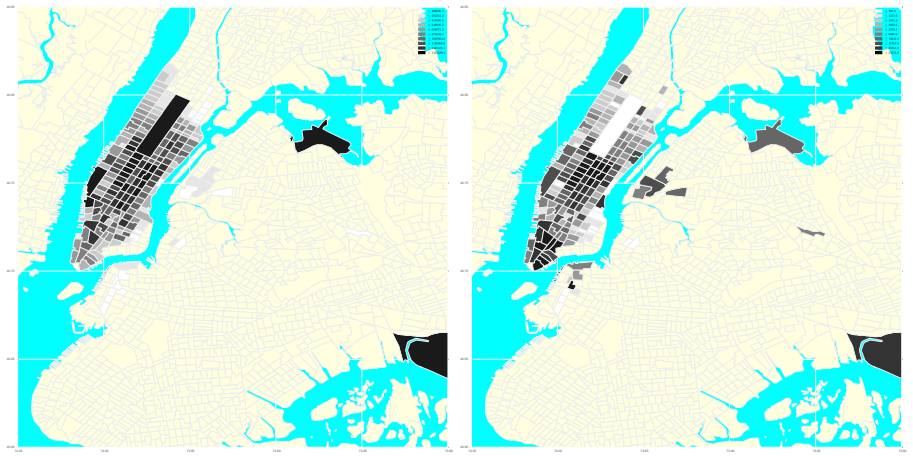
Figures 7.3 and 7.4 show the overall distribution of trips for both cities, and the used regions in New York.

7.6.2 Evaluation metric

To measure the quality of predictions, we compare the symmetric mean absolute percentage error (sMAPE) of the predicted values against the actual number of requests. Assume that Z_{ij} is the actual demand for region i and period j at time t, and Z'_{ij} is the predicted value, i.e. $\lfloor \lambda'_{ij} \rfloor$. Then for each region r_i the sMAPE error is defined as $\text{sMAPE}_i = (1/|P|) \sum_j \text{Err}_j$, where:

$$\text{Err}_j = \begin{cases} 0 & \text{if } Z'_{ij} = Z_{ij} = 0 \\ \dfrac{|Z'_{ij} - Z_{ij}|}{Z'_{ij} + Z_{ij}} & \text{otherwise} \end{cases} \tag{7.3}$$

[3] http://lehd.ces.census.gov/
[4] http://mesowest.org/api/



(a) Number of taxi pickups (b) Total number of jobs

Figure 7.4: Map of New York regions with high activity.

Moreira-Matias et al. argue that sMAPE is too sensitive to small deviations when the actual number of requests are small. They suggest adding a constant $c = 1$ to the denominator of Err_j to alleviate this effect (Moreira-Matias et al. 2013). The modified sMAPE measure is hence defined as:

$$\text{sMAPE}_i = \sum_j \frac{|Z'_{ij} - Z_{ij}|}{Z'_{ij} + Z_{ij} + 1} \tag{7.4}$$

To compute the error over all regions, we take a weighted average of errors based on the number of requests in each region

$$\text{sMAPE} = \sum_{r_i \in R} \frac{\text{sMAPE}_i \psi_i}{\Psi} \tag{7.5}$$

where $\psi_i = \sum_{r_i \in R} z_{ij}$ and $\Psi = \sum_{r_i \in R} \psi_i$.

All the errors reported in the remainder of this chapter are computed in this way. We compare different methods based on these errors. To evaluate the significance of the difference between the errors of two methods, we use the independent two-sample t-test with a 0.05 significance level.

7.6.3 Decomposition

Before we evaluate the prediction components, we first analyze the performance of the decomposition. We want to answer the following two questions:

Q1 How much information do we lose by decomposing the profiles?

Q2 Are the components for one location informative when reused in another location?

To answer these effect, we take three datasets

- **NYC10** New York 2010 (200 most active regions)
- **NYC14** New York 2014 (same regions at NYC10)
- **SFC10** San Francisco 2010 (50 most active regions)

and we compute the decomposition for each of them. We then use the components to transform and reconstruct on all three datasets.

The results are shown in Table 7.1 for decomposition with 6 and 16 components respectively. This table shows that the reconstruction errors between the two datasets from New York are very similar. This indicates that the basic behavioral patterns did not change much between 2010 and 2014. Figure 7.5 shows the

		to		
		NYC10	**NYC14**	**SFC10**
from	**NYC10**	0.1358	0.1408	0.2774
	NYC14	0.1402	0.1406	0.2851
	SFC10	0.1758	0.1814	0.2450

(a) 6 components

		to		
		NYC10	**NYC14**	**SFC10**
from	**NYC10**	0.1238	0.1292	0.2655
	NYC14	0.1270	0.1267	0.2658
	SFC10	0.1530	0.1573	0.2245

(b) 16 components

Table 7.1: Reconstruction error across datasets. Rows indicate components used. Columns indicate reconstructed datasets.

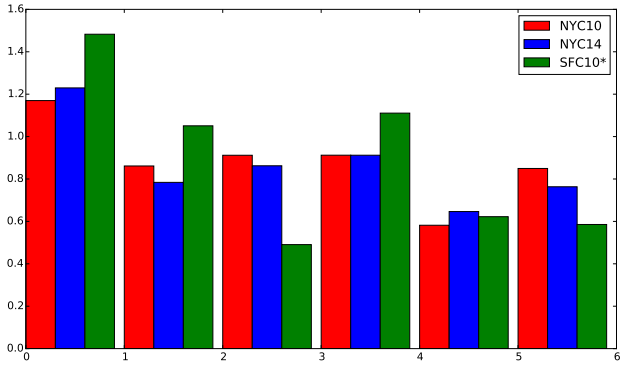


Figure 7.5: Average component weights for three datasets for 6 components obtained from NYC10 (components shown in Figure 7.2). SFC10 scaled up by a factor of 10.

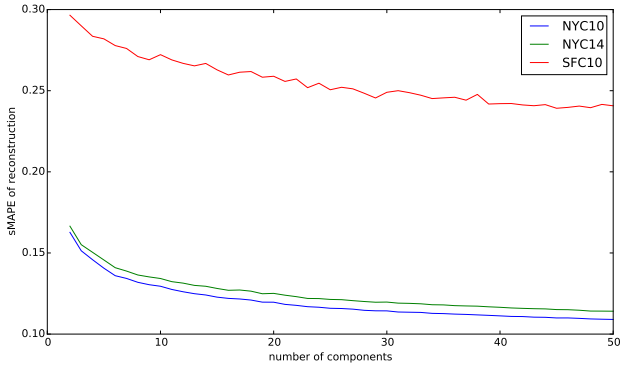


Figure 7.6: Reconstruction error of NYC10 components by number of components.

average component weights for 6 components of NYC 2010 when reconstructing the three datasets. The reconstruction errors for San Francisco are significantly worse, which may be due to the high number of zeros (potentially missing entries) in this dataset. By increasing the number of components we can reduce the reconstruction error as illustrated in Figure 7.6 (it will eventually reach zero).

7.6.4 Prediction using historic data

In this experiment we examine the performance of our approach in the traditional request prediction setting: given historic data from a given set of regions, predict the future demand for the *same set of regions*. In these experiments we assume that no information about recently-materialized requests is available. We want to answer this question:

Q3 How does our proposed method compare with the traditional methods when recent observations are *not* available?

We perform this experiment in two settings: predict two weeks in the future, and four years in the future. For this experiment, we take all the data for New York City in 2010 and we split it into blocks of 8 weeks. The first 6 weeks in the block are training data, the 8th week is the test data. (The 7th week is left out as a buffer.) We repeat this for blocks starting at week $1, 5, 9, \ldots, 41$. In the second stage of the experiment we predict the 8th week for each block but four years later (2014).

The comparison contains the following approaches:

Naive: Predict the average over the whole training set for all examples in the test set. This method will serve as a reference point in the evaluation.

Poisson: Predict the requests using Poisson processes. For each time, region, and day of week, a rate is learned from the historic data.

Poisson_W: Predict the requests using *weighted time-varying Poisson models* (Moreira-Matias et al. 2013). The difference with the previous setting is that for learning the rates, instead of simple averaging, a weighted average is used. This way, later observations get larger weights than the earlier ones.

GBT: Predict the requests using a single gradient boosted regression model.

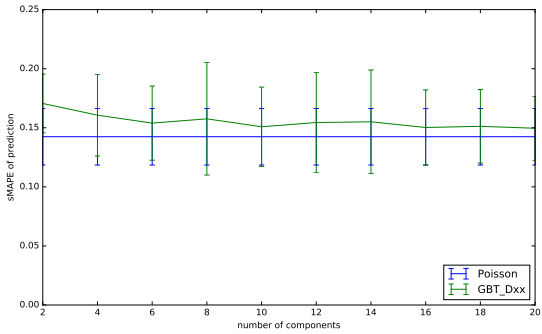


Figure 7.7: Prediction error for components by number of components for NYC10 to NYC10. Error bars indicate one standard deviation.

GBT_S: Predict the requests using region-specific GBT models.

GBT_D6 and **GBT_D8**: Predict the requests through decomposition, using 6 and 8 components respectively.

The approach **GBT** uses calendar information, demographic data (WAC and RAC) and weather information. In **GBT_S** the demographic data are excluded. The approaches based on decomposition don't use the full weather information (with information every 15 minutes), but instead uses snapshots at every four hours. The results are summarized in Table 7.2.

method	**2010 → 2010**	**2010 → 2014**
Naive	0.3694±0.0181	0.3720±0.0104
Poisson	0.1402±0.0227	0.1605±0.0212
Poisson_W	0.1413±0.0245	0.1618±0.0223
GBT	0.1499±0.0324	0.2173±0.0319
GBT_S	0.1463±0.0272	0.1702±0.0279
GBT_D6	0.1539±0.0314	0.2229±0.023
GBT_D8	0.1576±0.0476	0.2194±0.0214

Table 7.2: sMAPE score for prediction with historic data. Results are averaged over 11 time blocks.

Discussion The errors are presented in Table 7.2. According to the statistical tests, in short-term prediction except for the method **GBT_D8**, other learning

methods are not significantly different. In long-term prediction, the methods **Poisson**, **Poisson_W** and **GBT_S** are significantly better than others. All methods outperform the naive averaging in both settings.

In results of short-term prediction two observations are worth noting: (1) the method **GBT** which uses a single global model competes with region-specific models, and (2) the region-specific method **GBT_S** is not significantly better than the **GBT** method. These observations confirm the ability of the GBT models to adapt to the properties of feature space. Overall, the results indicate that in this setting the feature-based methods do not offer an improvement in terms of accuracy over the traditional methods.

Figure 7.7 shows the effect of the number of components on the prediction error. Whereas Figure 7.6 showed us that increasing the number of components decreases the reconstruction error, the same does not necessarily hold for the prediction error. This is due to the prediction task becoming harder when there are more components (in the limit it will correspond to **GBT**).

7.6.5 Prediction using historic data and recent observations

In this experiment we examine the ability of our method to exploit the information about recent observations. This is the question that we want to answer:

Q4 How does our proposed method compare with the traditional methods when historic data and recent observations are available?

The data that we use for this experiment is the same as in section 7.6.4, except that each row has ten additional features indicating the counts at 15-minute intervals for the period $[t-180, t-30]$ for a prediction made at time t. Note that we have excluded the two observations that immediately precede the prediction time. This allows for a delay in collection and processing of the real-time data.

The comparison contains the following approaches:

Naive: The average over the whole training set.

ARIMA: Predict the requests using region-specific ARIMA models.

GBT_O: Predict the requests using a single GBT model, trained on the data that contains recent observations as additional features.

GBT_S_O: Predict the requests using region-specific GBT models, trained with the ten additional features.

For training the ARIMA models, we used the same setting as the one used by Moreira-Matias et al. (Moreira-Matias et al. 2013). To enforce the two-period delay between the last observation and prediction, we performed three-step-ahead prediction using the ARIMA models. The results are summarized in Table 7.3.

method	**2010 → 2010**	**2010 → 2014**
ARIMA	0.1694±0.0123	0.1765±0.0067
GBT_O	0.1187±0.0067	0.1403±0.0071
GBT_S_O	0.1236±0.0082	0.1365±0.0118

Table 7.3: sMAPE score for prediction with historic data and recent observations. Results are averaged over 11 time blocks.

Discussion Error values presented in Table 7.3 and the statistical tests show that both versions of the feature-based method perform significantly better than ARIMA in both settings (For the sake of consistency, we will keep calling these settings short-term and long-term). As expected, ARIMA performs similarly in short-term and long-term predictions, while the accuracy of feature-based methods decreases in long-term prediction. A comparison with Table 7.2 shows that by using both contextual information and recent observations, we can outperform all traditional methods (which at best use only one of these two types of information).

7.6.6 Prediction on unseen regions

Some of the traditional methods (e.g. Poisson models) require records of requests over a sufficiently long period of time for training a model. Some other methods (e.g. ARIMA models) rely solely on the most recent observations. Our proposed method can use both recent observations from the target region and historic data from other regions. In this experiment we examine this scheme and answer the following question:

Q5 How does the proposed method compare to the traditional methods when no historic data is available?

We ran this experiment in two settings. In the first setting we use the same time-based partitioning (6 + 1 weeks), but we use different regions for training

and testing. We use 5-fold cross-validation on the regions (i.e. 80% of regions for training and 20% of regions for testing).

In the second setting we use the model learned on all regions of New York to predict the taxi demand in San Francisco. Because there is a large discrepancy between the number of trips in both cities, we rescale the request counts for San Francisco for each week separately. In this way, we can determine whether the prediction captures the relative distribution of trips over the week. Three weeks were omitted from the test set because no data was available for those weeks.

We compare the methods **ARIMA**, **GBT**, and **GBT_O**. This experiment is not applicable to other methods, as they all require historic data from the target region. In previous experiments we used the last part of historic data to do the model selection for ARIMA models. In this experiment we start using the ARIMA model from the second day. At each day, we use all previous days for model selection. In the first day, to predict the counts at time t we take the average of observations at times $t-3$ and $t-4$, which are the most recent available observations. The results are summarized in table 7.4.

method	**NYC → NYC**	**NYC → SFC**
Naive	0.3693±0.0180	0.6908±0.0719
ARIMA	0.1764±0.0123	0.2449±0.0158
GBT	0.2338±0.0224	0.6272±0.0772
GBT_O	0.1429±0.0057	0.3558±0.0966

Table 7.4: sMAPE scores for predicting on unseen regions.

Discussion In the first setting (using historic data from another region in the same city), all differences are statistically significant. In this setting, the simple feature-based method performs poorly. However, when the recent observations are added to the features, it outperforms the **ARIMA** method. In the second setting (using historic data from another city), there is no difference between **GBT** and naive averaging. In general, in this setting all methods perform poorly.

7.7 Conclusions and future work

We introduced two novel approaches for feature-based taxi request prediction. To our knowledge this is the first work that specifically aims at using background

information for making predictions and to have the ability to make predictions in regions for which no historic data is available. Our experiments suggest that models that combine contextual features and recent observations outperform the models which take only one of these two into account. These models also perform significantly better in the task of making predictions for unseen regions in the same city. For the task of making predictions for unseen regions in a different city, none of the studied methods produced satisfactory results.

We also contributed a method for analyzing data of this type by showing that non-negative matrix factorization can be used to discover behavioral patterns in the data. The main advantage of this method is that it offers *interpretable* predictions.

Prediction of taxi trips is usually a component of a larger pipeline that solves an optimization problem (e.g. routing or scheduling problems). For future work, we want to investigate whether our predictions will improve the quality of the final solutions in such a pipeline. Another extension is to use methods such as *Poisson dependency networks* for modeling the dependencies among Poisson variables. Another interesting direction is the use of more relevant features. For example, it would be interesting to combine our work with that of Chang et al. (Chang et al. 2010) which focuses on detecting landmark in a city based on taxi data. By using those techniques we can develop a model about which type of landmarks draw most traffic and use that information as features in our prediction model.

Chapter 8

Conclusions and Future Work

8.1 Summary and Conclusions

Probabilistic inference and constraint satisfaction and optimization have been studied extensively for decades. The two fields are known to have connections, and their intersection has been studied before. However, these connections have not been fully exploited in solving problems that involve both constraint satisfaction and probabilistic inference. An example of such problems is maximization of the expected utility which is a natural extension of deterministic CSPs to a situation where there is only probabilistic knowledge about some of the problem parameters. Another class of such problems are those that involve constraining or optimizing the probability values themselves. Our contribution in this regard was to present two mechanisms for solving these two classes of problems. Both these methods build on existing constraint programming solvers. This means that a lot of facilities of these solvers (e.g. the complex constraints that they support) can be used by our methods.

The second contribution of this thesis was motivated by the recent interest in integrating data mining with constraint satisfaction and optimization. In particular, we formulated and solved two clustering problems as integer linear programming models.

Advances in optimization under uncertainty allow the user to formulate more accurate models by specifying a distribution over random variables. A problem that is rarely addressed in existing work on stochastic optimization is how to accurately approximate the real-world uncertainties using a probability distribution. This question has been extensively studied in the statistical

machine learning domain. Our third contribution was to use techniques from this domain to learn probability distributions for taxi passenger demand.

In the first chapter we introduced the three main research question in this thesis. We will now review these questions again and summarize the answers that this thesis provides for them:

Q1. *How can we combine principles of constraint satisfaction and probabilistic inference to solve problems that involve both tasks?*

We introduced two mechanisms for combining probabilistic inference and constraint satisfaction and optimization. The first mechanism models the computational steps of a probabilistic inference engine in terms of constraints. The second mechanism uses a novel depth-first search algorithm and calls an external probabilistic inference engine.

In the first approach, we compile a Bayesian network into a d-DNNF, and formulate and reason over this structure using a constraint programming solver. We show that using this approach we can support a wide range of queries and constraints in a flexible and declarative manner. We used this approach for pattern mining in Bayesian networks.

In the second approach, we presented a new stochastic constraint programming method. Existing works on stochastic constraint programming made at least one of the following assumptions: 1) the random variables are independent, 2) the probability distribution should be first converted to a list of possible worlds. Instead, we assumed a non-trivial factored joint distribution over the random variables. We introduced an And-Or search algorithm to combine constraint satisfaction and probabilistic inference. We introduced a novel bound that works directly on this tree. To compute this bound we used a state-of-the-art probabilistic inference engine. We implemented this mechanism within a generic constraint solver. So our method supports existing complex constraints.

Q2. *What are the potentials of formulating constrained clustering as integer linear programming?*

We also formulated two clustering problems using integer linear programming: constrained minimum sum-of-squares (MSS) clustering and constrained graph clustering. To solve the first problem we used a formulation in which each possible cluster is represented by a variable. This leads to an exponential number of variables. We used a column-generation algorithm to incrementally add a significant subset of these variables to the model. We presented a novel branch and bound algorithm to solve the *pricing* subproblem, i.e. the problem of finding the next variable to be added to the model. This hybrid approach allows us to take care of the constraints when solving the subproblem. Using

this approach we obtained the optimal solution for a number of constraint clustering instances for the first time.

We also presented two formulations for a biologically-inspired graph clustering problem. The first formulation was based on enumerating all simple paths in the graph. In dense graphs, this formulation leads to a large number of constraints. The second formulation also included a worst-case exponential number of constraints. But we used a cutting-plane algorithm to include only a sufficient subset of these constraints in the model. We also introduced a bi-objective Pareto optimization method to balance the two components of the objective function. Our experiments showed that each formulation performs better on a certain type of problems. When the graph is sparse and the total number of simple paths is low, the first formulation is more efficient. For denser graphs, the overhead of solving the cutting-plane subproblem pays off and the second formulation performs better.

Q3 *How can we use data mining techniques to learn the distribution of passenger requests from records of taxi trips?*

We introduced two novel approaches for learning distributions for taxi passenger demand. The main novelty of our method is using background information for making predictions and to have the ability to make predictions in regions for which no historic data is available. Our experiments suggest that models that combine contextual features and recent observations outperform the models which take only one of these two into account. These models also perform significantly better in the task of making predictions for unseen regions in the same city.

8.2 Discussion and Future Work

We will now shortly discuss a number of directions for future research. We divide our discussion into three parts. In the first part we review the open questions and future directions on the topic of combining CSP(O) and probabilistic inference. The second part deals with possibilities for future work in the application of integer programming to constrained clustering problems. We conclude by mentioning opportunities for further research on the topic of learning taxi passenger demand.

8.2.1 Probabilistic Models in Constraint Satisfaction and Optimization

In one of our studies we represented an arithmetic circuit by a set of constraints. It has been shown that using a dedicated propagator for an s-DNNF outperforms the method that represents the circuit as a set of constraints (Gange and Stuckey 2012). A dedicated propagator for the arithmetic circuit obtained from the d-DNNF might improve the efficiency of the search. Designing such a propagator is a direction for future research.

In our work on stochastic constraint programming using And-Or search, there are several directions for future research. One component missing from our work which is common in the existing research on stochastic constraint programming is the notion of *chance constraints*. A chance constraint is a constraint that is allowed to be violated in a certain fraction of possible worlds (Walsh 2002). Introducing chance constraints to our method is a topic for future work.

Stochastic constraint programming is a combination of two difficult problems. This motivates designing approximate methods that aim at solving large problems. An example of approximate methods for stochastic constraint programming is to use reinforcement learning for solving these problems (S. D. Prestwich et al. 2017). Another possible approach which is closer to an exact algorithm is to use the same And-Or branch and bound method, while ignoring the branches that lead to a small probability (or expected utility) in such a way that the difference with the exact objective is guaranteed to be less than a certain threshold.

The And-Or branch and bound algorithm shares many principles with nested constraint programming (NCP) (Chu and Stuckey 2014). In principle, factored stochastic constraint programs can be solved as special cases of NCP. This would make it possible to take advantage of improvements that are built in this framework. Currently, when a stochastic constraint programs is modeled as NCP it is assumed that the random variables are independent. Modeling factored stochastic programs as NCP requires designing special propagators that reason over the joint probability distribution. This can be done in the same direction as designing propagators that reason over arithmetic circuits.

In our FSCP framework, the problems of constraint satisfaction/optimization (search) and probabilistic inference (counting) were decoupled. The search component repeatedly communicates with an external inference engine to answer probability queries. A direction for future research is to perform these tasks jointly within a single mechanism. This can facilitate exploiting the joint structure of deterministic and probabilistic factors. DPLL search is a possible candidate for unifying search and counting. On one hand, it has been used for

probabilistic inference (Sang et al. 2004), and on the other hand it can be used for propagation in constraint satisfaction/optimization (Ohrimenko et al. 2007).

Finally, it is interesting to allow the probability distribution to be influenced by the decisions. Standard scenario-based methods sample the scenarios in a phase prior to the decision-making step and hence can not deal with such problems. The advantage of our method is that it directly reasons over the problem structure and hence it might be easier to adapt to such problems.

8.2.2 Constrained Clustering using Integer Linear Programming

There are a number of open questions about our work on clustering by column generation. The performance of this method is affected by the efficiency of solving the master problem and the subproblem. An interesting direction for future work is to improve the bounding method in the branch and bound algorithm that solves the subproblem. It might also be beneficial to use an approximate algorithm for solving the subproblem and use the exact method only when this approximate method fails. This technique has been used in the column generation algorithm for the unconstrained version of our clustering problem (Aloise, Hansen, and Liberti 2012). In our algorithm, the constraints are enforced in the subproblem. Another interesting future work is to extend the branch and bound algorithm so that it supports other types of constraints.

In our work on graph clustering, we based our formulation on encoding assignments of nodes to clusters as decision variables. Another common encoding in graph clustering is based on co-membership of pairs of nodes in a cluster. Using this approach, the number of clusters can be decided automatically (Benati et al. 2017). Formulating the graph clustering problem using this encoding and comparing it with the current two formulations is an interesting direction for future work. Moreover, in our formulations of the graph clustering problem we did not use any redundant constraints. Finding valid inequalities that can strengthen the current formulations is another topic that needs further investigation.

In both formulations of clustering problems, we had to develop specialized algorithms for solving the subproblems. This at least partially contradicts the declarative nature of constraint solving which was one of the motivations for formulating DM/ML tasks as CSP(O) models. A possible direction for future research is to automatically detect and solve the subproblems. There is a framework called *Generic Column Generation* (GCG) which converts a MILP formulation into an equivalent model that can be solved using the

column-generation method. It then formulates the subproblem as another MILP model and solves the problem using the branch-and-price method (Gamrath and Lübbecke 2010). However, our MSS clustering problem can not easily be modeled as a compact MILP problem. Moreover, the subproblem in our formulation involves non-linear constraints and hence can not be directly solved by the MILP solver. This suggests that new representations and solving techniques are required to extend existing methods such as GCG to general clustering problems.

8.2.3 Learning Taxi Passenger Demand

In our work we assumed that the random variables representing the passenger demand in different areas are independent given the contextual features (e.g. weather, demographic attributes, calendar information, etc.). This assumption simplifies both learning the distributions and sampling from them. However, there are methods in spatio-temporal data analysis that take the dependencies between variables into account (Diggle 2013). Another type of models that encode such dependencies between count variables are *Poisson dependency networks* (Hadiji et al. 2015). Using these richer models might lead to more accurate distributions and more realistic samples.

An emerging topic in machine learning is *probabilistic programming*. In this declarative approach, the user specifies a probabilistic model, and inference (also learning, as a special type of inference) is performed automatically. Several probabilistic programming frameworks rely on sampling as their approximate inference mechanism. It might be beneficial to use these frameworks for learning distributions and sampling scenarios. A next step will be extending probabilistic programming frameworks with decision making capabilities. This will offer a fully-declarative approach to (multi-stage) stochastic optimization. Since the declarative specification might include constraints over the decision variables, enforcing such constraints in this setting is a challenge that needs to be addressed.

8.3 Concluding Remarks

The three domains of AI that we studied in this thesis – i.e. DM, CSP(O), and probabilistic inference – share common underlying principles. On one hand, CSP(O) and probabilistic inference both deal with exploring combinatorial spaces. On the other hand, a significant number of DM problems involve searching over a combinatorial hypothesis space. This similarity allows using methods from one domain to solve problems in another one. Following this

perspective, we solved a DM problem (clustering) using CSP(O) mechanisms (integer linear programming). Sometimes a problem lies at an intersection of these domains. An approach for solving these problems is to combine the existing methods in each domain into a new solving mechanism. We took this approach for combining CSP(O) and probabilistic graphical models.

Our work in both directions led to improvements over the state-of-the-art. The author hopes that this work inspires others to work along similar intersections and to contribute to bridging the gap between these subfields.

Bibliography

Achterberg, T. (2007). "Constraint integer programming". PhD thesis. Berlin Institute of Technology.

– (2009). "SCIP: solving constraint integer programs". In: *Math. Program. Comput.* 1.1, pp. 1–41.

Aggarwal, C. C., M. Bhuiyan, and M. A. Hasan (2014). "Frequent Pattern Mining Algorithms: A Survey". In: *Frequent Pattern Mining*. Springer, pp. 19–64.

Agrawal, R., T. Imielinski, and A. N. Swami (1993). "Mining Association Rules between Sets of Items in Large Databases". In: *SIGMOD Conference*. ACM Press, pp. 207–216.

Agrawal, R. and R. Srikant (1994). "Fast Algorithms for Mining Association Rules in Large Databases". In: *VLDB*. Morgan Kaufmann, pp. 487–499.

Aloise, D., A. Deshpande, P. Hansen, and P. Popat (2009). "NP-hardness of Euclidean sum-of-squares clustering". In: *Machine Learning* 75.2, pp. 245–248.

Aloise, D. and P. Hansen (2009). "A branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering". In: *Pesquisa Operacional* 29.3, pp. 503–516.

Aloise, D., P. Hansen, and L. Liberti (2012). "An improved column generation algorithm for minimum sum-of-squares clustering". In: *Math. Program.* 131.1-2, pp. 195–220.

Armbruster, M., M. Fügenschuh, C. Helmberg, and A. Martin (2008). "A Comparative Study of Linear and Semidefinite Branch-and-Cut Methods for Solving the Minimum Graph Bisection Problem". In: *IPCO*. Vol. 5035. Lecture Notes in Computer Science. Springer, pp. 112–124.

Babaki, B., T. Guns, and S. Nijssen (2014). "Constrained Clustering Using Column Generation". In: *CPAIOR*. Vol. 8451. Lecture Notes in Computer Science. Springer, pp. 438–454.

Bache, K. and M. Lichman (2013). *UCI Machine Learning Repository*.

Bartlett, M. and J. Cussens (2017). "Integer Linear Programming for the Bayesian network structure learning problem". In: *Artif. Intell.* 244, pp. 258–271.

Basu, S., I. Davidson, and K. Wagstaff (2008). *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC Press.

Benati, S., J. Puerto, and A. M. Rodríguez-Chía (2017). "Clustering data that are graph connected". In: *European Journal of Operational Research* 261.1, pp. 43–53.

Benedetti, M., A. Lallouet, and J. Vautard (2008). "Quantified Constraint Optimization". In: *CP*. Vol. 5202. Lecture Notes in Computer Science. Springer, pp. 463–477.

Bent, R. and P. V. Hentenryck (2004). "Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers". In: *Operations Research* 52.6, pp. 977–987.

– (2005). "Online Stochastic Optimization Without Distributions". In: *ICAPS*. AAAI, pp. 171–180.

Berg, J. and M. Järvisalo (2017). "Cost-optimal constrained correlation clustering via weighted partial Maximum Satisfiability". In: *Artif. Intell.* 244, pp. 110–142.

Binder, J., D. Koller, S. J. Russell, and K. Kanazawa (1997). "Adaptive Probabilistic Networks with Hidden Variables". In: *Machine Learning* 29.2-3, pp. 213–244.

Boulicaut, J. and B. Jeudy (2001). "Mining Free Itemsets under Constraints". In: *IDEAS*. IEEE Computer Society, pp. 322–329.

Brusco, M. J. and S. Stahl (2005). "Minimum Within-Cluster Sums of Squares Partitioning". In: *Branch-and-Bound Applications in Combinatorial Data Analysis*. Springer.

Buluç, A., H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz (2016). "Recent Advances in Graph Partitioning". In: *Algorithm Engineering*. Vol. 9220. Lecture Notes in Computer Science, pp. 117–158.

Campos, L. M. de, J. A. Gámez, and S. Moral (2001). "Simplifying Explanations in Bayesian Belief Networks". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 9.4, pp. 461–490.

Carvajal, R., M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub (2013). "Imposing Connectivity Constraints in Forest Planning Models". In: *Operations Research* 61.4, pp. 824–836.

Chang, H., Y. Tai, and J. Y. Hsu (2010). "Context-aware taxi demand hotspots prediction". In: *IJBIDM* 5.1, pp. 3–18.

Chavira, M. and A. Darwiche (2005). "Compiling Bayesian Networks with Local Structure". In: *IJCAI*. Professional Book Center, pp. 1306–1312.

– (2008). "On probabilistic inference by weighted model counting". In: *Artif. Intell.* 172.6-7, pp. 772–799.

Chen, S. J., A. Choi, and A. Darwiche (2014). "Algorithms and Applications for the Same-Decision Probability". In: *J. Artif. Intell. Res.* 49, pp. 601–633.

Choi, C. W., W. Harvey, J. H. M. Lee, and P. J. Stuckey (2006). "Finite Domain Bounds Consistency Revisited". In: *Australian Conference on Artificial Intelligence*. Vol. 4304. Lecture Notes in Computer Science. Springer, pp. 49–58.

Chu, G. and P. J. Stuckey (2014). "Nested Constraint Programs". In: *CP*. Vol. 8656. Lecture Notes in Computer Science. Springer, pp. 240–255.

Dao, T., K. Duong, and C. Vrain (2013). "A Declarative Framework for Constrained Clustering". In: *ECML/PKDD (3)*. Vol. 8190. Lecture Notes in Computer Science. Springer, pp. 419–434.

– (2015). "Constrained Minimum Sum of Squares Clustering by Constraint Programming". In: *CP*. Vol. 9255. Lecture Notes in Computer Science. Springer, pp. 557–573.

– (2017). "Constrained clustering by constraint programming". In: *Artif. Intell.* 244, pp. 70–94.

Darwiche, A. (2003). "A differential approach to inference in Bayesian networks". In: *J. ACM* 50.3, pp. 280–305.

– (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Das, J. and H. Yu (2012). "HINT: High-quality protein interactomes and their applications in understanding human disease". In: *BMC Systems Biology* 6, p. 92.

Davidson, I. and S. S. Ravi (2007). "The complexity of non-hierarchical clustering with instance and cluster level constraints". In: *Data Min. Knowl. Discov.* 14.1, pp. 25–61.

Davidson, I., S. S. Ravi, and L. Shamis (2010). "A SAT-based Framework for Efficient Constrained Clustering". In: *SDM*. SIAM, pp. 94–105.

Davis, N., G. Raina, and K. Jagannathan (2016). "A multi-level clustering approach for forecasting taxi travel demand". In: *ITSC*. IEEE, pp. 223–228.

Demiriz, A., K. Bennett, and P. Bradley (2008). "Using assignment constraints to avoid empty clusters in k-means clustering". In: *Constrained Clustering: Algorithms, Applications and Theory*. Chapman & Hall/CRC.

Deng, Z. and M. Ji (2011). "Spatiotemporal structure of taxi services in Shanghai: Using exploratory spatial data analysis". In: *Proceedings of the 19th International Conference on Geoinformatics*, pp. 1–5.

Diehr, G. (1985). "Evaluation of a branch and bound algorithm for clustering". In: *SIAM Journal on Scientific and Statistical Computing* 6.2, pp. 268–284.

Diggle, P. J. (2013). *Statistical analysis of spatial and spatio-temporal point patterns*. CRC Press.

Dinkelbach, W. (1967). "On nonlinear fractional programming". In: *Management science* 13.7, pp. 492–498.

Druzdzel, M. J. and H. J. Suermondt (1994). "Relevance in probabilistic models:"Backyards" in a "small world"". In: *Working notes of the AAAI–1994 Fall Symposium Series: Relevance*, pp. 60–63.

Fan, N. and P. M. Pardalos (2010). "Linear and quadratic programming approaches for the general graph partitioning problem". In: *J. Global Optimization* 48.1, pp. 57–71.

Fauré, C., S. Delprat, J. Boulicaut, and A. Mille (2006). "Iterative Bayesian Network Implementation by Using Annotated Association Rules". In: *EKAW*. Vol. 4248. Lecture Notes in Computer Science. Springer, pp. 326–333.

Feillet, D. (2010). "A tutorial on column generation and branch-and-price for vehicle routing problems". In: *4OR* 8.4, pp. 407–424.

Ferreira, C. E., A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey (1998). "The node capacitated graph partitioning problem: A computational study". In: *Math. Program.* 81, pp. 229–256.

Ferrucci, F., S. Bock, and M. Gendreau (2013). "A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods". In: *European Journal of Operational Research* 225.1, pp. 130–141.

Freund, Y. and R. E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *J. Comput. Syst. Sci.* 55.1, pp. 119–139.

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5, pp. 1189–1232.

Gamrath, G. and M. E. Lübbecke (2010). "Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs". In: *SEA*. Vol. 6049. Lecture Notes in Computer Science. Springer, pp. 239–252.

Gange, G. and P. J. Stuckey (2012). "Explaining Propagators for s-DNNF Circuits". In: *CPAIOR*. Vol. 7298. Lecture Notes in Computer Science. Springer, pp. 195–210.

Ge, Y., H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. J. Pazzani (2010). "An energy-efficient mobile recommender system". In: *KDD*. ACM, pp. 899–908.

Gilpin, S., S. Nijssen, and I. N. Davidson (2013). "Formalizing Hierarchical Clustering as Integer Linear Programming". In: *AAAI*. AAAI Press.

Gondek, D. and T. Hofmann (2004). "Non-Redundant Data Clustering". In: *ICDM*. IEEE Computer Society, pp. 75–82.

Grötschel, M. and Y. Wakabayashi (1989). "A cutting plane algorithm for a clustering problem". In: *Math. Program.* 45.1-3, pp. 59–96.

Guns, T., T. Dao, C. Vrain, and K. Duong (2016). "Repetitive Branch-and-Bound Using Constraint Programming for Constrained Minimum Sum-of-Squares Clustering". In: *ECAI*. Vol. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 462–470.

Guns, T., S. Nijssen, and L. D. Raedt (2011a). “Evaluating Pattern Set Mining Strategies in a Constraint Programming Framework”. In: *PAKDD (2)*. Vol. 6635. Lecture Notes in Computer Science. Springer, pp. 382–394.

– (2011b). “Itemset mining: A constraint programming perspective”. In: *Artif. Intell.* 175.12-13, pp. 1951–1983.

Hadiji, F., A. Molina, S. Natarajan, and K. Kersting (2015). “Poisson Dependency Networks: Gradient Boosted Models for Multivariate Count Data”. In: *Machine Learning* 100.2-3, pp. 477–507.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). “Exploring network structure, dynamics, and function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15.

Hansen, P. and B. Jaumard (1997). “Cluster analysis and mathematical programming”. In: *Math. Program.* 79, pp. 191–215.

Hemmi, D., G. Tack, and M. Wallace (2017). “Scenario-Based Learning for Stochastic Combinatorial Optimisation”. In: *CPAIOR*. Vol. 10335. Lecture Notes in Computer Science. Springer, pp. 277–292.

Hnich, B., R. Rossi, S. A. Tarim, and S. Prestwich (2011). “A Survey on CP-AI-OR Hybrids for Decision Making Under Uncertainty”. In: *Hybrid Optimization: The Ten Years of CPAIOR*. Springer, pp. 227–270.

Hnich, B., R. Rossi, S. A. Tarim, and S. D. Prestwich (2012). “Filtering algorithms for global chance constraints”. In: *Artif. Intell.* 189, pp. 69–94.

Ichoua, S., M. Gendreau, and J. Potvin (2006). “Exploiting Knowledge About Future Demands for Real-Time Vehicle Dispatching”. In: *Transportation Science* 40.2, pp. 211–225.

Jaroszewicz, S., T. Scheffer, and D. A. Simovici (2009). “Scalable pattern mining with Bayesian networks as background knowledge”. In: *Data Min. Knowl. Discov.* 18.1, pp. 56–100.

Jensen, F., F. V. Jensen, and S. L. Dittmer (1994). “From Influence Diagrams to junction Trees”. In: *UAI*. Morgan Kaufmann, pp. 367–373.

Jensen, R. E. (1969). “A Dynamic Programming Algorithm for Cluster Analysis”. In: *Operations Research* 17.6, pp. 1034–1057.

Ji, X. and J. E. Mitchell (2007). “Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement”. In: *Discrete Optimization* 4.1, pp. 87–102.

Johnson, E. L., A. Mehrotra, and G. L. Nemhauser (1993). “Min-cut clustering”. In: *Math. Program.* 62, pp. 133–151.

Kircher, M., D. M. Witten, P. Jain, B. J. O’roak, G. M. Cooper, and J. Shendure (2014). “A general framework for estimating the relative pathogenicity of human genetic variants”. In: *Nature genetics* 46.3, pp. 310–315.

Koboldt, D. C. et al. (2012). “Comprehensive molecular portraits of human breast tumors”. In: *Nature* 490.7418, pp. 61–70.

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models - Principles and Techniques*. MIT Press.

Koontz, W. L. G., P. M. Narendra, and K. Fukunaga (1975). "A Branch and Bound Clustering Algorithm". In: *IEEE Trans. Computers* 24.9, pp. 908–915.

Kwisthout, J. (2013). "Most Inforbable Explanations: Finding Explanations in Bayesian Networks That Are Both Probable and Informative". In: *ECSQARU*. Vol. 7958. Lecture Notes in Computer Science. Springer, pp. 328–339.

Labbé, M. and F. A. Özsoy (2010). "Size-constrained graph partitioning polytopes". In: *Discrete Mathematics* 310.24, pp. 3473–3493.

Lee, D. D. and H. S. Seung (1999). "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755, pp. 788–791.

Lee, D. D. and H. S. Seung (2000). "Algorithms for Non-negative Matrix Factorization". In: *NIPS*. MIT Press, pp. 556–562.

Leiserson, M. D., F. Vandin, H.-T. Wu, J. R. Dobson, J. V. Eldridge, J. L. Thomas, A. Papoutsaki, Y. Kim, B. Niu, M. McLellan, et al. (2015). "Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes". In: *Nature genetics* 47.2, pp. 106–114.

Li, X., G. Pan, Z. Wu, G. Qi, S. Li, D. Zhang, W. Zhang, and Z. Wang (2012). "Prediction of urban human mobility using large-scale taxi traces and its applications". In: *Frontiers of Computer Science in China* 6.1, pp. 111–121.

Lisser, A. and F. Rendl (2003). "Graph partitioning using linear and semidefinite programming". In: *Math. Program.* 95.1, pp. 91–101.

Malhas, R. and Z. A. Aghbari (2009). "Interestingness filtering engine: Mining Bayesian networks for interesting patterns". In: *Expert Syst. Appl.* 36.3, pp. 5137–5145.

Manandhar, S., A. Tarim, and T. Walsh (2003). "Scenario-based Stochastic Constraint Programming". In: *IJCAI*. Morgan Kaufmann, pp. 257–262.

Mannila, H. and H. Toivonen (1997). "Levelwise Search and Borders of Theories in Knowledge Discovery". In: *Data Min. Knowl. Discov.* 1.3, pp. 241–258.

Mateescu, R. and R. Dechter (2008). "Mixed deterministic and probabilistic networks". In: *Ann. Math. Artif. Intell.* 54.1-3, pp. 3–51.

Mehrotra, A. and M. A. Trick (1998). "Cliques and clustering: A combinatorial approach". In: *Oper. Res. Lett.* 22.1, pp. 1–12.

Merle, O. du, P. Hansen, B. Jaumard, and N. Mladenovic (1999). "An Interior Point Algorithm for Minimum Sum-of-Squares Clustering". In: *SIAM J. Scientific Computing* 21.4, pp. 1485–1505.

Miao, F., S. Han, S. Lin, Q. Wang, J. A. Stankovic, A. M. Hendawi, D. Zhang, T. He, and G. J. Pappas (2016). "Data-Driven Robust Taxi Dispatch under Demand Uncertainties". In: *CoRR* abs/1603.06263.

Miller, J. and J. P. How (2017). "Demand Estimation and Chance-Constrained Fleet Management for Ride Hailing". In: *CoRR* abs/1703.02130.

Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J.

– (1995). *Methods and applications of interval analysis*. SIAM studies in applied mathematics. SIAM.

Moreira-Matias, L., J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas (2013). "Predicting Taxi-Passenger Demand Using Streaming Data". In: *IEEE Trans. Intelligent Transportation Systems* 14.3, pp. 1393–1402.

Mueller, M. and S. Kramer (2010). "Integer Linear Programming Models for Constrained Clustering". In: *Discovery Science*. Vol. 6332. Lecture Notes in Computer Science. Springer, pp. 159–173.

Négrevergne, B. and T. Guns (2015). "Constraint-Based Sequence Mining Using Constraint Programming". In: *CPAIOR*. Vol. 9075. Lecture Notes in Computer Science. Springer, pp. 288–305.

Nijssen, S. and A. Zimmermann (2014). "Constraint-Based Pattern Mining". In: *Frequent Pattern Mining*. Springer, pp. 147–163.

Ohrimenko, O., P. J. Stuckey, and M. Codish (2007). "Propagation = Lazy Clause Generation". In: *CP*. Vol. 4741. Lecture Notes in Computer Science. Springer, pp. 544–558.

Os, B. J. van and J. J. Meulman (2004). "Improving Dynamic Programming Strategies for Partitioning". In: *J. Classification* 21.2, pp. 207–230.

Parker, J. S., M. Mullins, M. C. Cheang, S. Leung, D. Voduc, T. Vickery, S. Davies, C. Fauron, X. He, Z. Hu, et al. (2009). "Supervised risk predictor of breast cancer based on intrinsic subtypes". In: *Journal of clinical oncology* 27.8, pp. 1160–1167.

Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann.

Pralet, C., G. Verfaillie, and T. Schiex (2007). "An Algebraic Graphical Model for Decision with Uncertainties, Feasibilities, and Utilities". In: *J. Artif. Intell. Res.* 29, pp. 421–489.

Prestwich, S. D., R. Rossi, and A. Tarim (2017). "Stochastic Constraint Programming as Reinforcement Learning". In: *CoRR* abs/1704.07183.

Przytula, K. W., D. Dash, and D. Thompson (2003). "Evaluation of Bayesian networks used for diagnostics". In: 60, pp. 1–12.

Pulido-Tamayo, S., B. Weytjens, D. De Maeyer, and K. Marchal (2016). "SSA-ME Detection of cancer driver genes using mutual exclusivity by small subnetwork analysis". In: *Scientific Reports* 6.

Qi, R. and D. L. Poole (1995). "A New Method for Influence Diagram Evaluation". In: *Computational Intelligence* 11, pp. 498–528.

Riedel, S. (2008). "Improving the Accuracy and Efficiency of MAP Inference for Markov Logic". In: *UAI*. AUAI Press, pp. 468–475.

Rietbergen, M. T., L. C. van der Gaag, and H. L. Bodlaender (2014). "Provisional Propagation for Verifying Monotonicity of Bayesian Networks". In: *ECAI*.

Vol. 263. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 759–764.

Rossi, F., P. van Beek, and T. Walsh, eds. (2006). *Handbook of Constraint Programming*. Vol. 2. Foundations of Artificial Intelligence. Elsevier.

Rossi, R., A. Tarim, B. Hnich, and S. D. Prestwich (2008). "Cost-Based Domain Filtering for Stochastic Constraint Programming". In: *CP*. Vol. 5202. Lecture Notes in Computer Science. Springer, pp. 235–250.

Saadi, I., M. Wong, B. Farooq, J. Teller, and M. Cools (2017). "An investigation into machine learning approaches for forecasting spatio-temporal demand in ride-hailing service". In: *CoRR* abs/1703.02433.

Saglam, B., F. S. Salman, S. Sayin, and M. Türkay (2006). "A mixed-integer programming approach to the clustering problem with an application in customer segmentation". In: *European Journal of Operational Research* 173.3, pp. 866–879.

Sang, T., F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi (2004). "Combining Component Caching and Clause Learning for Effective Model Counting". In: *SAT*.

Schaeffer, S. E. (2007). "Graph clustering". In: *Computer Science Review* 1.1, pp. 27–64.

Schölkopf, B., A. J. Smola, R. C. Williamson, and P. L. Bartlett (2000). "New Support Vector Algorithms". In: *Neural Computation* 12.5, pp. 1207–1245.

Schrijver, A. (2003). *Combinatorial Optimization – Polyhedra and Efficiency*. Springer.

Sherali, H. D. and J. Desai (2005). "A Global Optimization RLT-based Approach for Solving the Hard Clustering Problem". In: *J. Global Optimization* 32.2, pp. 281–306.

Shimony, S. E. (1993). "The role of relevance in explanation I: Irrelevance as statistical independence". In: *Int. J. Approx. Reasoning* 8.4, pp. 281–324.

Tarim, A., S. Manandhar, and T. Walsh (2006). "Stochastic Constraint Programming: A Scenario-Based Approach". In: *Constraints* 11.1, pp. 53–80.

Teso, S., R. Sebastiani, and A. Passerini (2017). "Structured learning modulo theories". In: *Artif. Intell.* 244, pp. 166–187.

T'Kindt, V. and J. Billaut (2006). *Multicriteria Scheduling - Theory, Models and Algorithms (2. ed.)* Springer.

Wagstaff, K. and C. Cardie (2000). "Clustering with Instance-level Constraints". In: *ICML*. Morgan Kaufmann, pp. 1103–1110.

Walsh, T. (2002). "Stochastic Constraint Programming". In: *ECAI*. IOS Press, pp. 111–115.

Wolsey, L. A. (1998). *Integer programming*. New York, NY, USA: Wiley-Interscience.

Xia, Y. and J. Peng (2005). "A Cutting Algorithm for the Minimum Sum-of-Squared Error Clustering". In: *SDM*. SIAM, pp. 150–160.

Yu, H., L. Tardivo, S. Tam, E. Weiner, F. Gebreab, C. Fan, N. Svrzikapa, T. Hirozane-Kishikawa, E. Rietman, X. Yang, et al. (2011). "Next-generation sequencing to generate interactome datasets". In: *Nature methods* 8.6, pp. 478–480.

Yuan, C., H. Lim, and T. Lu (2011). "Most Relevant Explanation in Bayesian Networks". In: *J. Artif. Intell. Res. (JAIR)* 42, pp. 309–352.

Yuan, C., X. Wu, and E. A. Hansen (2010). "Solving Multistage Influence Diagrams using Branch-and-Bound Search". In: *UAI*. AUAI Press, pp. 691–700.

Yuan, J., Y. Zheng, L. Zhang, X. Xie, and G. Sun (2011). "Where to find my next passenger". In: *UbiComp*. ACM, pp. 109–118.

Yue, Y., H. d. Wang, B. Hu, and Q. q. Li (2011). "Identifying Shopping Center Attractiveness Using Taxi Trajectory Data". In: *Proceedings of the 2011 International Workshop on Trajectory Data Mining and Analysis*. New York, NY, USA: ACM, pp. 31–36.

Yue, Y., Y. Zhuang, Q. Li, and Q. Mao (2009). "Mining time-dependent attractive areas and movement patterns from taxi trajectory data". In: *Proceedings of the 17th International Conference on Geoinformatics*, pp. 1–6.

Zhang, D., T. He, S. Lin, S. Munir, and J. A. Stankovic (2014). "Dmodel: Online Taxicab Demand Model from Big Sensor Data in a Roving Sensor Network". In: *BigData Congress*. IEEE, pp. 152–159.

Zhao, K., D. Khryashchev, J. Freire, C. T. Silva, and H. T. Vo (2016). "Predicting taxi demand at high spatial resolution: Approaching the limit of predictability". In: *BigData*. IEEE, pp. 833–842.

Zheng, Y., Y. Liu, J. Yuan, and X. Xie (2011). "Urban computing with taxicabs". In: *UbiComp*. ACM, pp. 89–98.

List of publications

Journal article

- Behrouz Babaki, and Anton Dries. *Feature-based Taxi Request Prediction.* (manuscript in preparation).

Conference papers

- Behrouz Babaki, Tias Guns, and Siegfried Nijssen. "Constrained clustering using column generation". In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings,* volume 8451 of *Lecture Notes in Computer Science,* pages 438–454. Springer, 2014.

- Behrouz Babaki, Tias Guns, Siegfried Nijssen, and Luc De Raedt. "Constraint-based querying for bayesian network exploration". In Élisa Fromont, Tijl De Bie, and Matthijs van Leeuwen, editors, *Advances in Intel- ligent Data Analysis XIV - 14th International Symposium, IDA 2015, Saint Etienne, France, October 22-24, 2015, Proceedings,* volume 9385 of *Lecture Notes in Computer Science,* pages 13–24. Springer, 2015.

- Behrouz Babaki, Tias Guns, and Luc De Raedt. "Stochastic constraint programming with and-or branch-and-bound". In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017,* pages 539–545. ijcai.org, 2017.

Workshop paper

- Behrouz Babaki, Dries Van Daele, Bram Weytjens, and Tias Guns. "A branch-and-cut algorithm for constrained graph clustering". *Data Science meets Optimization workshop (colocated with CPAIOR), Padova, Italy, 2017.*

Curriculum Vitae

Behrouz Babaki Studied Industrial Engineering at Iran University of Science and Technology, Tehran, and received his Bachelor degree in September 2005. He obtained a Master of Science in Industrial Engineering from Sharif University of Technology, Tehran, in January 2009. His thesis was titled *"Dynamic Pricing of Perishable Assets using Demand Learning"*.

In September 2011, he joined KU Leuven, Belgium, for a Master of Science in Artificial Intelligence. His thesis was titled *"Preposition Disambiguation using Relational Learning"*. He graduated with great honors in September 2012. Afterwards, he started his PhD studies under the supervision of professor Luc De Raedt and professor Tias Guns at the DTAI lab of KU Leuven. In September 2017, he will defend his thesis titled *"On Constraints, Optimisation, Probability and Data Mining"*.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven