Partition Strategies For Incremental Mini-Bucket

Alicia Nicas Miquel

Director: Alexander Ihler - UC Irvine
Internal examiner: Lluís Belanche Muñoz - Computer Science

October 13, 2016
UC Irvine
Informatics Engineering - Computation
Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Barcelona (UPC) – BarcelonaTech

Abstract

Probabilistic graphical models such as Markov random fields and Bayesian networks provide powerful frameworks for knowledge representation and reasoning over models with large numbers of variables. Unfortunately, exact inference problems on graphical models are generally NP-hard, which has led to significant interest in approximate inference algorithms.

Incremental mini-bucket is a framework for approximate inference that provides upper and lower bounds on the exact partition function by, starting from a model with completely relaxed constraints, i.e. with the smallest possible regions, incrementally adding larger regions to the approximation. Current approximate inference algorithms provide tight upper bounds on the exact partition function but loose or trivial lower bounds.

This project focuses on researching partitioning strategies that improve the lower bounds obtained with mini-bucket elimination, working within the framework of incremental mini-bucket.

We start from the idea that variables that are highly correlated should be reasoned about together, and we develop a strategy for region selection based on that idea. We implement the strategy and explore ways to improve it, and finally we measure the results obtained using the strategy and compare them to several baselines.

We find that our strategy performs better than both of our baselines. We also rule out several possible explanations for the improvement.

Resumen

Los modelos en grafo probabilísticos, tales como los campos aleatorios de Markov y las redes bayesianas, ofrecen poderosos marcos de tabajo para la representación de conocimiento y el razonamiento en modelos con gran número de variables. Sin embargo, los problemas de inferencia exacta en modelos de grafos son NP-hard en general, lo que ha causado que se produzca bastante interés en métodos de inferencia aproximados.

El mini-bucket incremental es un marco de trabajo para inferencia aproximada que produce como resultado límites aproximados inferior y superior de la función de partición exacta, a base de -empezando a partir de un modelo con todos los constraints relajados, es decir, con las regiones más pequeñas posible- incrementalmente añadir regiones más grandes a la aproximación. Los métodos de inferencia aproximada que existen actualmente producen límites superiores ajustados de la función de partición, pero los límites inferiores suelen ser demasiado imprecisos o incluso triviales.

El objetivo de este proyecto es investigar estrategias de partición que mejoren los límites inferiores obtenidos con el algoritmo de mini-bucket, trabajando dentro del marco de trabajo de mini-bucket incremental.

Empezamos a partir de la idea de que creemos que debería ser beneficioso razonar conjuntamente con las variables de un modelo que tienen una alta correlación, y desarrollamos una estrategia para la selección de regiones basada en esa idea. Posteriormente, implementamos nuestra estrategia y exploramos formas de mejorarla, y finalmente medimos los resultados obtenidos usando nuestra estrategia y los comparamos con varios métodos de referencia.

Nuestros resultados indican que nuestra estrategia obtiene límites inferiores más ajustados que nuestros dos métodos de referencia. También consideramos y descartamos dos posibles hipótesis que podrían explicar esta mejora.

Resum

Els models en graf probabilístics, com bé els camps aleatoris de Markov i les xarxes bayesianes, ofereixen poderosos marcs de treball per la representació del coneixement i el raonament en models amb grans quantitats de variables. Tanmateix, els problemes d'inferència exacta en models de grafs son NP-hard en general, el qual ha provocat que es produeixi bastant d'interès en mètodes d'inferència aproximats.

El mini-bucket incremental es un marc de treball per a l'inferència aproximada que produeix com a resultat límits aproximats inferior i superior de la funció de partició exacta que funciona començant a partir d'un model al qual se l'hi han relaxat tots els constraints -es a dir, un model amb les regions mes petites possibles- i anar afegint a l'aproximació regions incrementalment més grans. Els mètodes d'inferència aproximada que existeixen actualment produeixen límits superiors ajustats de la funció de partició. Tanmateix, els límits inferiors acostumen a ser massa imprecisos o fins aviat trivials.

El objectiu d'aquest projecte es recercar estratègies de partició que millorin els límits inferiors obtinguts amb l'algorisme de mini-bucket, treballant dins del marc de treball del mini-bucket incremental.

La nostra idea de partida pel projecte es que creiem que hauria de ser beneficiós per la qualitat de l'aproximació raonar conjuntament amb les variables del model que tenen una alta correlació entre elles, i desenvolupem una estratègia per a la selecció de regions basada en aquesta idea. Posteriorment, implementem la nostra estratègia i explorem formes de millorar-la, i finalment mesurem els resultats obtinguts amb la nostra estratègia i els comparem a diversos mètodes de referència.

Els nostres resultats indiquen que la nostra estratègia obté límits inferiors més ajustats que els nostres dos mètodes de referència. També considerem i descartem dues possibles hipòtesis que podrien explicar aquesta millora.

Acknowledgements

I would like to thank the director of my project, Prof. Alexander Ihler, for giving me the opportunity for working with him and for hosting me in his lab, and for his advice and support and patience.

I would also like to thank the supervisor of my project, Prof. Lluís Belanche, for his advice and support.

Additionally, wish to express my sincere thanks to the Balsells fellowship program and Professor Roger Rangel for giving me the opportunity to carry out my project at UC Irvine.

Finally, I also want to thank my family for being so patient and supportive.

Contents

Chapter 1

Context

1.1 Problem formulation

Mini-bucket elimination is an algorithmic framework for approximate inference on graphical models that provides a lower and upper bound on the exact result. Mini-bucket elimination lowers the complexity of the problem to be solved by partitioning the factors that ought to be reasoned about together into mini-buckets, thus relaxing some of its constraints. The strategy used to partition the buckets heavily influences the accuracy of the bounds obtained. There exists a spectrum of partitioning strategies that attempt to obtain accurate bounds. For an upper bound, these approaches use a specific and greedy way to estimate the improvement that could be obtained through each of the available clusters. However, this greedy method is not applicable to lower bounds, and no equivalent method has yet been developed. Starting from the incremental mini-bucket algorithm developed by Forouzan and Ihler [2], this project focuses on researching partitioning strategies that improve the lower bounds obtained with mini-bucket elimination.

1.2 Background

1.2.1 Graphical models

A graphical model is a probabilistic model for which a graph expresses the conditional dependence structure among random variables [1] by explicitly representing the direct relationships inherent in the joint probability distribution [2]. Examples of graphical models include Bayesian networks, Markov random fields, constraint networks and influence diagrams. Consider a set of variables $X = \{x_1, ..., x_n\}$ and a distribution over them:

$$p(x) = \frac{1}{Z} \prod_{\alpha \in I} f_\alpha(x_\alpha)$$

$$Z = \sum_{x} f_\alpha(x_\alpha)$$

where x indicates a configuration of the variables and Z is the normalizing constant, called the partition function. We associate the probability distribution $p(x)$ with a graph $G = (V, E)$ where each variable x_i is associated with a node $i \in V$ and is connected by an edge of the graph to x_j if both variables x_i and x_j are arguments of the same function f_α. I is then a set of all the sets of variables that appear together (cliques) in G.

Common inference tasks on graphical models include finding the most likely, or MAP, configuration of $p(x)$, which is a combinatorial optimization problem, or computing the partition function Z, which is a combinatorial summation problem. Computing Z -which corresponds to the probability of evidence in Bayesian networks- or the marginal probabilities of $p(x)$ are central problems in many learning and inference tasks.

Unfortunately, inference on graphical models is NP-hard and often computationally intractable for real-world problems. Exact inference, such as bucket elimination, is exponential on the treewidth of the model, which motivates the development of a spectrum of approximations and bounds subject to computational limits that make them tractable.

1.2.2 Bucket elimination

Bucket elimination [3] is an exact inference algorithm that eliminates each of the variables sequentially. Given an elimination order over the variables of the distribution, bucket elimination collects all factors that include x_i as their earliest-eliminated argument, takes their product, and eliminates x_i from the product to produce a new factor over later variables, which is placed in the bucket of its "parent", the earliest uneliminated variable.

The space and time complexity of bucket elimination are exponential in the induced width of the graph given the elimination order. While good elimination orders can be identified using various heuristics, this exponential dependence often makes direct application of bucket elimination intractable for many real-world problems.

1.2.3 Junction trees

We can view the bucket elimination process as passing "messages" between the nodes and their parents along a tree structure [6]. Each newly generated factor in the bucket elimination step can be viewed as a "message" that is passed

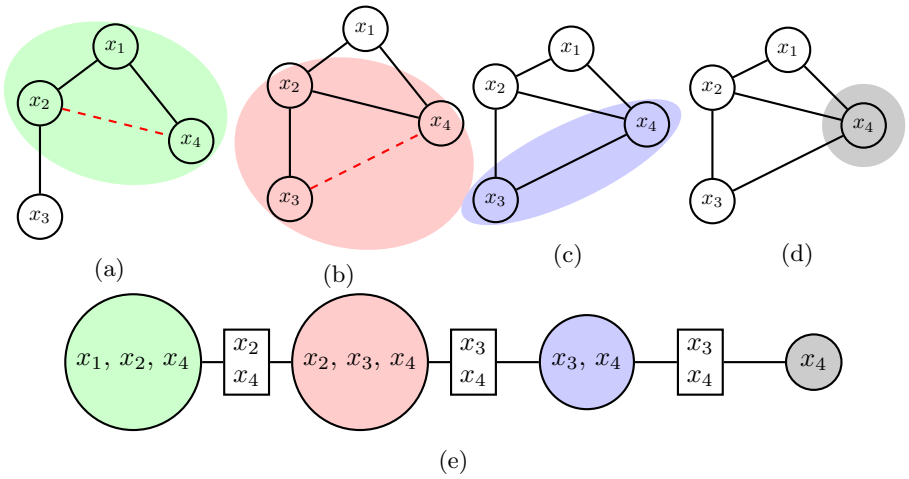


Figure 1.1: Construction of a junction tree on a small graph: (a) The first step is to triangulate (add edges between its parents) node 1. We can now create the first cluster of the junction tree, which is made up of node 1 and its parents, x_2 and x_4. (b) Next, we triangulate node 2. The second cluster is made up of node 2b and its parents, x_3 and x_4. Since x_2 is the first parent of node 1, we add an edge between the first and second clusters. (c) we continue triangulating and adding a new cluster with the node and its parents, and add an edge from the second to the third clusters. (d) The last cluster is only made up of node 4. (d) The resulting junction tree.

forward from node i (i.e. the bucket of x_i) to its parents. In the case that the graph $G = (V, E)$ is a tree, the forward and backward elimination processes can be directly viewed as passing messages between the nodes. However, for more general models, it is more notationally convenient to view the elimination as passing messages over a junction tree, a tree structure formed by subsets of nodes (called clusters) in the original graph.

A cluster graph is a graph formed by subsets of variables. A cluster graph is also a junction graph if it satisfies the running intersection property: for each variable $i \in V$, the sub-graph consisting of the clusters and edges of the cluster graph that include i is a connected tree.

Constructing junction trees

The bucket elimination process is analogous to constructing a junction tree. the junction tree simply records the variable scopes of the buckets $\{\mathbf{B}_i\}$ and their message trajectories (since j is the first parent of i iff the factor ψ_i^{new} created when eliminating x_i falls into the bucket $\mathbf{B}_j$ of variable x_j). This process is visualized for a small graphical model in Figure 1.1.

1.2.4 Mini-bucket elimination

Mini-bucket elimination [4] is an approximate version of bucket elimination where the factors in each bucket are grouped into partitions and where each partition, also called a mini-bucket, includes at most $ibound + 1$ variables. The bounding parameter *ibound* then serves as a way to control the complexity of elimination, as the elimination of each variable is performed on each mini-bucket separately, instead of on the more complex original bucket.

Mini-bucket elimination gives upper and lower bounds on the exact partition function, and its time and space complexity are exponential in the user-controlled *ibound*. Smaller *ibound* values result in lower computational cost, but are typically less accurate. The accuracy of the resulting bound also depends heavily on the strategy used for partitioning the buckets. From the variational perspective, this corresponds to the choice of regions in the approximations, where the regions define which sets of variables will be reasoned about jointly. Several partitioning strategies have been developed. This project will explore partitioning strategies that yield accurate lower bounds on the partition function.

It's useful to consider the effect of partitioning the factors into different mini-buckets and eliminating in each mini-bucket separately. This procedure effectively splits a variable into one or more replicates, one for each mini-bucket. Mini-bucket is equivalent to exact bucket elimination if all the copies of each variable are constrained to be equal. Otherwise, the additional degrees of freedom lead to a relaxed problem and can generate bounds on the exact partition function.

1.2.5 Weighted mini-bucket

An improvement to minibucket, weighted mini-bucket [5], generalizes the MBE bound with a "weighted" elimination, where the general procedure is the same, except that the sum/max operators are replaced with weighted sums where the weights are normalized to sum to one for each variable.

1.3 State of the art

1.3.1 Partitioning methods

Mini-bucket elimination and its weighted variant compute a partitioning over each bucket Bi to bound the complexity of inference and compute an upper bound on the partition function Z. However, different partitioning strategies will result in different upper bounds. Rollon and Dechter [7] proposed a framework to study different partitioning heuristics, and compared them with the original scope based heuristic proposed by Dechter and Rish [4]. Here we summarize

several approaches.

Scope-based Partitions

Proposed in Dechter and Rish [4], scope-based partitioning is a top-down approach that tries to minimize the number of mini-buckets in B_i by including as many functions as possible in each minibucket. To this end, it first orders the factors in B_i by decreasing number of arguments. Starting from the largest, each factor f is then merged with the first available minibucket where the size of the union of its variables with the variables of f isn't larger than $ibound+1$. If there are no minibuckets available that can include the factor, a new minibucket is created and the scheme continues until all factors are assigned to a mini-bucket.

Content-based Partitions

Rollon and Dechter [7], on the other hand, seeks to find a partitioning that is closest to the true bucket function. This requires solving the optimization problem of minimizing the distance between the true bucket and the product of the minibuckets. The distance is minimized in a greedy fashion, and Rollon and Dechter [7] studied the effectiveness of several different distance functions across multiple problem instances; however, no one distance was found to consistently outperform scope-based partitioning.

Relax-Compensate-Recover

Choi and Darwiche [8] indirectly addresses the problem of partition selection within their Relax, Compensate and Recover framework, in which certain equality constraints in the graph are first relaxed in order to reduce complexity of inference. New auxiliary factors are then introduced to compensate for the relaxation and enforce a weaker notion of equivalence. The recovery process then aims to identify those equivalence constraints whose relaxation were most damaging and recover them. Choi and Darwiche [8] proposed a number of recovery heuristics, including mutual information and residual recovery.

1.3.2 Variational bounds

The variational viewpoint of inference corresponds to optimizing an objective function over a collection of beliefs constrained to lie within the marginal polytope, or set of marginal probabilities that can be achieved by some joint distribution. Efficient approximations are then developed by relaxing these constraints to enforce only a subset of them. Like mini-bucket bounds, the quality of variational bounds depends significantly on the choice of regions. Often regions are chosen to match the original model factors, and then improved using methods like cluster pursuit.

Cluster pursuit

Sontag et al. [9] developed a bottom-up approach for MAP estimation in which regions (typically cycles or triplets) are added incrementally: First, the dual decomposition bound is optimized through message passing. Then, a pre-defined set of clusters, such as triplets or faces of a grid, are scored by computing a lower bound on their potential improvement of the bound. After adding the best-scoring cluster, the procedure repeats.

1.3.3 Incremental mini-bucket

Incremental mini-bucket [2] uses a hybrid approach that - like mini-bucket - uses the graph structure to guide region selection while also taking advantage of the iterative optimization and scoring techniques of cluster pursuit. Cluster pursuit algorithms use the function values in order to select which clusters should be added to the model. However, there are often prohibitively many clusters to consider, which forces cluster pursuit algorithms to restrict their search to a predefined set of clusters, such as triplets. Incremental mini-bucket uses the graph structure to guide region selection, restricting the search to merges of existing clusters, within one bucket at a time. This allows us to restrain the complexity of the search and add larger regions more effectively.

Current algorithms provide reasonably tight upper bounds, but the lower bounds are often too loose. This is because the lower bounds can be made trivial by zeroes in the factors. Consider an example when eliminating variable $X1$ from a bucket containing two factors, $\{f_{1,3}, f_{1,2}\}$, with $ibound = 1$. Suppose that $f_{1,2}$ returns 0 for some assignment to the variables in its scope. The lower bound provided by mini-bucket elimination is trivial

$$\sum_x f_{1,3}, \min_x f_{1,2}$$

since the zeroes in the result of $\min_x f_{1,2} = \lambda_2^1$ will get propagated to any factor that is reasoned about together with λ_2^1, and all the information they contain will be lost. In contrast, however loose the result of maximizing over a variable is, the results the product of that result with other factors will still contain some of the information of the other factors. This is why for this project we're focusing on researching ways to improve the lower bounds.

Chapter 2

Scope

2.1 Objective

The main objective of this project is to improve the results of the mini-bucket elimination algorithm due to Forouzan and Ihler [2] by developing a mini-bucket partitioning strategy that provides tighter lower bounds on the approximation it provides and that are efficient in memory usage.

The secondary objectives of the project arise from the iterative and incremental way that the project timeline is structured. For each potential partitioning strategy, the objectives of each iteration of the project are to make an incremental improvement to the strategy and to measure the magnitude of the improvement achieved in terms of its tightness and time and space complexity compared to the baseline.

2.2 Scope

This project builds on the framework of incremental mini-bucket. Working within this framework, we developed a region selection strategy starting from an idea that we thought could lead to tighter lower bounds on the approximation of the partition function, and we tested the performance of this strategy.

Before the project started, we had a code package implemented in python, PyGM, developed by Prof. Alexander Ihler, that implemented functionality for dealing with graphical models and for mini-bucket elimination. Our first step is to implement the incremental mini-bucket algorithm and related functionality, in order to be able to implement our strategy within this framework. The next step is to develop and implement our strategy. For this step, we first develop and implement a strategy that tries to make the highly correlated variables be reasoned about together. We then iteratively improve our strategy by perform-

ing experiments that measure the changes in accuracy of the algorithm when certain variables are changed, and incorporate the results into our algorithm.

Once we have developed the strategy, the next step is to design and perform experiments to test its performance. For this purpose, we compare the lower bounds obtained by our strategy to two baselines. Specifically, we compare using our heuristic for region choice to adding a random region, and we compare incremental mini-bucket using our strategy to scope-based incremental mini-bucket. We find that in both cases our method obtains tighter lower bounds on average.

Finally, we research possible reasons why our algorithm performs better than scope-based incremental mini-bucket. We considered whether the larger average size of the models generated by our algorithm may lead to tighter bounds, and we also consider whether selecting regions whose variables are more highly correlated on average may lead to tighter bounds. We find that we can discard both hypotheses.

Chapter 3

Methodology

This project was developed using an iterative development schedule made up of short one-week cycles, with weekly meetings dedicated to ensure an adequate progress of the project and to discuss the goals for the following cycle.

Before the start of the project, code with graphical models and mini-bucket elimination functionality was already developed, both in python and matlab. We chose to develop the project using python because of familiarity and speed of development and because of its tools, such as iPython, that are useful for performing experiments.

During the development stage, we used iPython notebooks to be able to closely examine the behavior of our algorithm and to make sharing the progress for meetings easier. In the iterations, we initially focused in developing and implementing the strategy from the initial idea, then we tested that it obtained useful results, and finally we focused on making improvements to the strategy. In order to make improvements, we designed and performed experiments in the testing step of each iteration to test the effects of changing some aspect of the algorithm, such as the amount of message passing. In the analysis and implementation steps, we incorporated our findings to the algorithm.

During the testing stage, we designed and performed experiments with the goal of measuring the accuracy of our algorithm and compare it to those of the baselines. These experiments were performed on artificial, randomly generated data. We tested the results against two baselines. Our first baseline was meant to determine whether our region choice heuristic is better than random choice. Our second baseline is a state-of the art algorithm for approximate inference, specifically, scope-based incremental mini-bucket. Because we found that our algorithm performed better on average than our state-of-the-art baseline, we additionally designed and performed tests aimed at finding the reason for the improvement.

Chapter 4

Planning

4.1 Duration

This project had a duration of six months. The starting date of this project was February 1st and its ending date was July 31st.

4.2 Stages

4.2.1 Setup

The first stage of the project was the setup. Its main objective was learning about the background of the problem that is addressed by the project and gaining familiarity with the codebase. In order to achieve this goal I studied the literature that makes up the context for the project and the state of the art of the work done on the problem.

4.2.2 Planning and feasibility

The Project Management Course taken at UPC covered the stage of analyzing the feasibility of the project and planning its objectives and stages thoroughly. This was done through the deliverables required for the course. There were six stages:

- Scope and context
- Planning
- Budget and sustainability

- Preliminary presentation
- Specialization module
- Presentation

4.2.3 Project iterations

After the setup stage, python code for incremental mini-bucket and our partition strategy was developed. Afterwards, the iterative process of improving the strategy started. Each iteration of this process involved the steps of analysis, implementation, test, and preparation for the next iteration. The first iteration started with the test step. In this iteration the performance of a scope-based partition strategy that has already been implemented was tested. This scope-based partition is a known strategy that works by merging the mini-buckets with the smaller scopes, and it was used as a baseline to compare to our strategy. In subsequent iterations, the strategy developed in the design stage will be implemented and iteratively refined.

Analysis

In the analysis step of each iteration, the strategy as implemented in the previous iteration is analyzed, and an improved strategy is developed. In this step, the objective is detecting problems with the current implementation, as well as analyzing the results of the testing steps in order to develop ways to incorporate the knowledge gained from them into our strategy. The analysis step also involves designing the code that will be developed in the following step.

Implementation

In the implementation step, the partitioning strategy developed from the analysis is implemented in python to be used with the existing incremental mini-bucket code.

Test

In the testing step, the results obtained with the new strategies are tested. The tests performed evaluate the tightness of the bounds obtained by the implemented strategies. Additionally, in this step, other experiments are designed and performed, in order to determine ways in which our strategy could be improved.

Stage	Estimated dedication(hours)	Actual dedication(hours)
Setup	105	230
Planning and feasibility	75	75
Project iterations	625	430
Testing and results		70
Final stage	70	70
Total	875	875

Table 4.1: Time breakdown

Preparation for the next iteration

In preparation for the next iteration, the problems with the current strategies will be identified in this step.

4.2.4 Testing and results

In this stage of the project, the final testing for the project was performed, and results were extracted. This involved two steps. The first step was to test the accuracy of the lower bounds obtained by our strategy, as compared to our two baselines. For this purpose, several experiments that compared the our strategy to a baseline were designed and performed. Our results from the first step indicated that our strategy performs better on average than both of our baselines. This result was surprising, since we expected our strategy to perform better than scope-based incremental mini-bucket only in certain cases. In the second step of the testing stage, some hypotheses for the reason for this result were tested. This required designing and performing experiments to test each of them.

4.2.5 Final stage

The final stage of the project involves the creation of the final report and preparation for the final presentation.

4.2.6 Time breakdown

The setup stage of the project lasted the first 9 weeks of the project. The planning and feasibility stage requires 75h of work, spread over 8 weeks, and it overlaps with the latter part of the setup stage. The rest of the available time before the last 4 weeks of the project is taken up by the project iterations. This means that TODO hours were dedicated to the project iterations, with each iteration lasting one week. After the project iterations, the testing step took up two weeks, and the final stage of the project took up the last 2 weeks of the project. Table 4.1 shows the time breakdown.

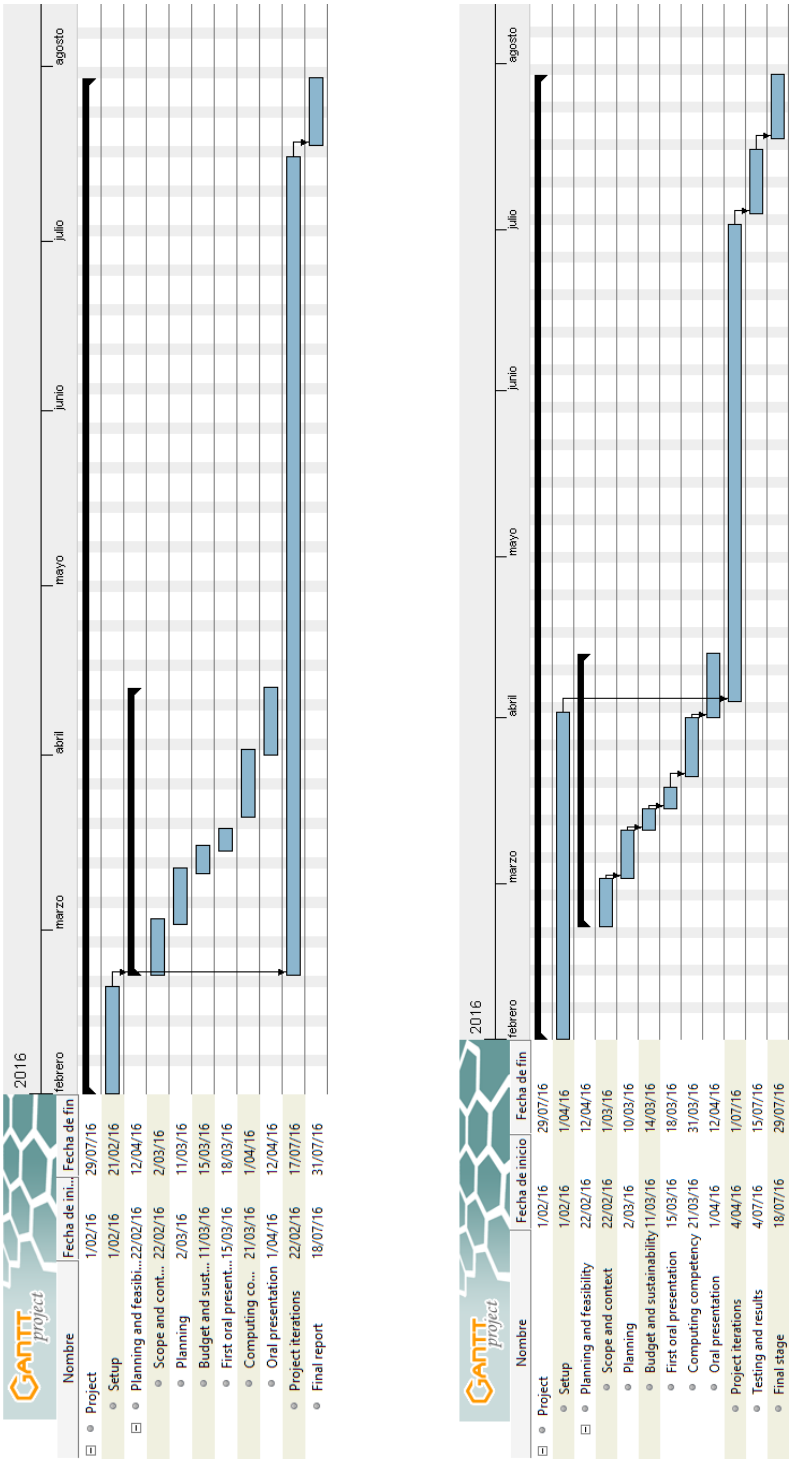
4.3 Gantt chart

Both the task durations and dependencies of the project estimated in the planning stage of the project and the final recorded ones are shown in the Gantt charts in Figure 4.1

4.4 Planning deviation

During the development of the project, there were some temporal deviations. They were fixed by focusing on developing one strategy, instead of several at the same time. The delay was caused by the time needed for the setup and knowledge acquisition being more than initially planned.

Additionally, a stage was added, taking up some of the time at the end of the project iterations stage, dedicated to performing experiments and gathering results. In Table 4.1 we can see the differences in duration estimated and actual for each step.



(a) (b)

Figure 4.1: (a) Gantt chart showing the estimated timing of the project stages. (b) Gantt chart showing the actual timing of the project stages.

Chapter 5

Budget

5.1 Resources

5.1.1 Hardware

1. Sony VAIO laptop

5.1.2 Software

1. Matlab 2015b
2. Python 3.4
3. Google drive
4. Dropbox
5. Windows 10

5.2 Human resources

This project has been developed by one person. Since this is a research project, the developer has two roles: project manager and researcher. The role of project manager entails planning the project, and the role of researcher entails doing the technical work of the project (designing, implementing, and testing the partitioning strategies for the mini-bucket algorithm). This being an academic project, the cost of human resources is fictitious; however, the cost has been estimated for the purpose of this budget. The costs are detailed in Table 5.1

Role	Hours	Cost per hour	Total cost
Project manager	75h	€50/h	€3750
Researcher	835h	€35/h	€29225
Total	910h		910

Table 5.1: Human resources budget

Product	Cost	Units	Service life	Amortization
Sony VAIO laptop	€800	1	5 years	€80
Total	€800			€80

Table 5.2: Hardware budget

5.3 Hardware

The development of this project requires a computer for every stage. Its cost is estimated in Table 5.2

5.4 Software

Some software products were needed to carry out this project. Most of the products needed are free to use, and the rest are free for academic projects like this one, but here their cost for commercial use is considered. The cost is detailed in 5.3

5.5 Unforeseen costs and contingency

Since the timeline is rigid, costs that only depend on the duration in months of the project were unlikely to change. The most likely unforeseen expense would have been hardware repair costs, which can be estimated at 300€. We estimated the risk of incurring these expenses to be at 10%. These expenses finally weren't incurred.

Product	Cost	Units	Service life	Amortization
Matlab 2016b student license	€100	1	3 years	€16.67
Python	€0	1	N/A	€0
Google Drive	€0	1	N/A	€0
Dropbox	€15/month	1	6 months	€90
Windows 10 Home	€135	1	3 years	€22.5
Sublime Text	€63	1	4 years	€7.88
Total	€388			€137.05

Table 5.3: Software budget

Concept	Estimated cost	Final cost
Human resources	€32975	€32975
Hardware	€80	€80
Software	€137.05	€137.05
Unforeseen expenses	€30	€0
Total	€33,222.05	€33,192.05

Table 5.4: Total budget: estimated and final

5.6 Budget deviation

This project didn't have a complex budget, its only requirements being a developer, a laptop, and some free-for-educational-purposes software. The biggest costs are those of human resources, and, since we were able to complete the project in the stated time, these costs haven't changed.

In our initial budget, we considered the possibility of incurring unforeseen costs during the course of the project. These unforeseen costs weren't actually incurred, so the final cost of the project is slightly smaller than the estimated cost.

We can see the comparison between estimated and final budgets in Table 5.4

Chapter 6

Sustainability report

This is a theory project that doesn't result in a product and, as such, has little environmental or social impact. However, no human activity is without external effects, and it's important to recognize and measure these effects in order to fully understand the contribution of the project.

6.1 Social sustainability

Working on this project has been an important experience for me in several ways. During the course of the project, I have gained important experience in applying the scientific method to finding solutions to a problem. Since my project is a science project, a big part of our methodology consisted in coming up with hypotheses and testing them, and using the test results to come up with new hypotheses. This method required some getting used to because it's different from most work I've done before in my schooling. Additionally, my work on this project highlighted certain problems I still haven't fully addressed, like that I need to be more proactive.

In its current form, and since it isn't geared toward any application in particular, this project doesn't have any social impact. The project may, however, have an impact through being a small contribution to the problem of inference in graphical models. As such, it will be beneficial to the community of graphical models researchers by providing an incremental improvement to existing approximate inference methods.

Additionally, graphical models are a powerful paradigm for knowledge representation and reasoning that have applications in a variety of fields, such as medical diagnostics or computer vision [2]. Advances in these fields may lead directly to increases in quality of life. Therefore, although a lot of work would be needed to incorporate its results into a practical application, the contribution of this project could have a positive societal impact.

On the other hand, the development of machine learning solutions causes some ethical concerns that need to be taken into account when considering the net effect of this development. A particularly pressing one is the possibility of a massive loss of jobs due to automation [12].

6.2 Economic sustainability

The budget for this project is described and justified in Chapter 5. Most of the budget is spent on human resources, with an item dedicated to hardware -the laptop used for development- and a few software licenses. The final budget for this project was of €33,192.05. This budget is hard to reduce, but a possible way to reduce some costs would be to try to use free or cheaper software instead of some of the programs that we needed paying licenses for.

6.3 Environmental sustainability

This project has some direct environmental impact as a result of being developed using a laptop with an estimated power rating of 33W. During the 875h of the project, this amounts to 26.25kWh. Since this project has been developed in California, we use the California emissions statistics to estimate the emissions footprint of this project. Therefore we estimate that electricity production causes emissions of 0.287kg of CO_2 per kWh[13], and that the carbon emissions for the entire project is of 7.53kg of CO_2. This impact is hard to reduce, especially since laptop computers have relatively low power consumption, and a high percentage of California's electricity is generated using renewable sources. Since the total time dedicated to development hasn't changed, the emissions estimate from the planning stage of the project is the same as the actual emissions generated by the project, save for possible differences between the actual sources of the consumed electricity and the percentages calculated in the statistics.

Graphical models are often used to tackle problems involving massive amounts of data, and algorithms are often extremely inefficient due to the inherent complexity of inference in graphical models. This means that using these methods requires significant computational resources and therefore has a high environmental impact.

This project's goal is to improve the efficiency of the algorithms used to solve our problem, and therefore could have an indirect positive impact in the environmental footprint of graphical models-based solutions. However, this project doesn't generate such a solution, and therefore doesn't have an impact in and of itself.

	PPP	Service life	Risks
Environmental	8	0	0
Economic	8	0	0
Social	10	0	0

Table 6.1: Sustainability matrix

6.4 Sustainability table

Table 6.1 presents the numerical sustainability score of the project, for the three aspects of sustainability considered. For both the service life and risks columns, we give all three aspects a score of 0, because, while we have identified potential impacts of the project, these aren't direct impacts and considerable work would have to be undertaken for them to be possible.

Chapter 7

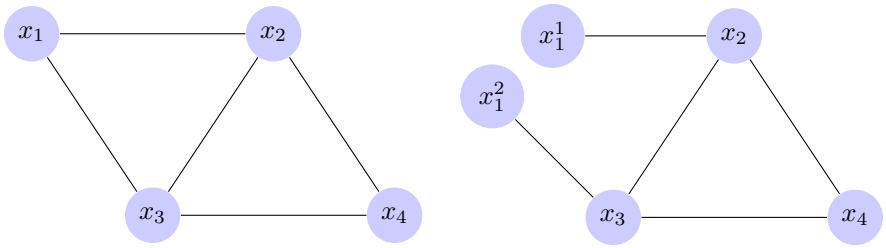
Partition strategy

We propose a strategy for region choice based on the idea that highly correlated variables should be reasoned about together. In order to improve the bounds, we need to take into account the values of the factors of the distribution, so that we can find the regions that lead to more improvement. Here, we observe that variables encode information about other variables that is lost when the variables aren't reasoned about together.

As an example, consider the graph in Figure 7.1a where both x_1 and x_2 and x_1 and x_3 are highly correlated. In this situation, x_2 and x_3 have the same value with high probability. However, in mini-bucket elimination, the bucket of the variable x_1 could be divided into two mini-buckets, each containing one of the factors $f(x_1, x_2)$ and $f(x_1, x_3)$, effectively creating two copies of the variable x_1, as shown in Figure 7.1b.

$$B_1 = \{f(x_1, x_2), f(x_1, x_3)\} \Rightarrow B_{1^1} = \{f(x_1^1, x_2)\}, B_{1^2} = \{f(x_1^2, x_3)\}$$

In this graph with relaxed constraints, having $x_2 \neq x_3$ is a consistent solution. This happens because the copies of x_1 can have different values.



(a) (b)

Figure 7.1: Example graph showing the effects of partitioning a bucket. (a) Graph where $x_1 \approx x_2$ and $x_1 \approx x_3$. (b) Graph where the bucket of x_1 is divided into two mini-buckets.

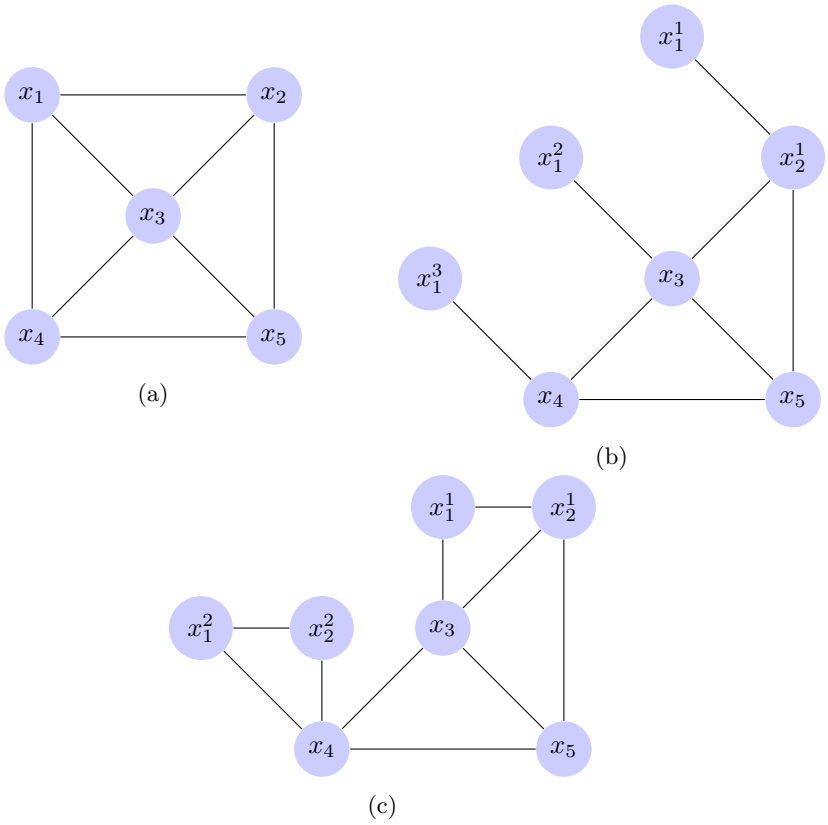


Figure 7.2: Example graph, showing the effects of partitioning a larger bucket and of adding a variable to the cliques. (a) A graph where the bucket of x_1 is formed by the functions f_{12}, f_{13}, and f_14. (b) The result of relaxing the constraints of the first bucket. (c) Result of adding x_2 to the mini-buckets and merging the buckets with the same variables.

Additionally, some pairs of variables contain more information about each other than others. We hypothesize that preserving the relationships between the most correlated variables will lead to tighter bounds.

Consider the graph in Figure 7.2a where x_1 is connected to x_2, x_3, and x_4. Suppose that $x_1 = x_2$. In incremental mini-bucket elimination, we start from a fully relaxed graph. A fully relaxed x_1 bucket looks like the graph in 7.2b.

If we want to slightly increase the complexity of the bucket, while taking into account our idea, we can try adding a highly correlated variable to all the cliques of the bucket. Doing this gives us the option of reasoning jointly about any two of three possible sets of variables -(x_1, x_2, x_3), (x_1, x_2, x_4) and (x_1, x_3, x_4)- all for the same memory cost. This cost is additionally lower than that of exactly eliminating x_1, which requires reasoning jointly over all of x_1, x_2, x_3, and x_4.

If $x_1 = x_2$, it makes sense to add x_2 to the cliques, as seen in graph 7.2c, since this actually leads to the same result as the exact graph, with lower complexity. Our hypothesis, then, is that, if x_1 doesn't equal x_2 but instead is only highly correlated to it, adding x_2 will lead to a good approximation.

Chapter 8

Implementation

This chapter describes the structure and implementation of our algorithm and discusses the experiments we performed in order to decide on the parameters of the algorithm and the conclusions we derived from them.

8.1 Algorithm

This section explains the structure of our code.

Our algorithm works within the framework of incremental mini-bucket described in Forouzan and Ihler [2].

The structure of the algorithm is the same; the differences come from the way that we're adding complexity to the model. Instead of finding the cliques we would benefit most from merging, we select a variable that we add to all the cliques.

The input of the algorithm is a factor graph, plus the *ibound*, which determines the maximum size of the cliques of the model and therefore its maximum complexity, and the numbers T, U, and V, which determine the amount of message passing that we will do before starting the merges, after each merge and in the end, respectively.

The algorithm begins by initializing a join graph using some elimination ordering for the variables, usually min-fill [10], and ibound = 1. For each bucket B_i, this results in each mini-bucket (or region) $q_i^k \in B_i$ containing a single factor θ_α.

We perform some rounds of message passing on the join graph. The number of rounds of message passing is determined by the input parameter T. In Section 8.2.2 we explore the effectiveness of various strategies for message passing. After the initial message passing, we go through the buckets following the elimination

order. For each of the buckets, the objective is to choose variables to add to the cliques of the bucket until the cliques reach the maximum size, which equals $ibound + 1$.

We work toward the goal by iteratively choosing the best variable, then making all the cliques of the bucket be made up of their old variables plus the new chosen variable. In order to choose the best variable v_j, we score all the variables of the bucket except the elimination bucket according to the strategy detailed in Chapter 7.

The next step is to update the graph. We do this using a variation of the method detailed in Forouzan and Ihler [2]. For each clique of the bucket, we create a new mini-bucket with scope var$(B_i) \cup v_j$ and add it to B_i. Eliminating v_i from this mini-bucket we obtain the forward message $\lambda_{r \Rightarrow \pi_r}$ from this region r to its parent region π_r. The earliest eliminated variable in the scope of $\lambda_{r \Rightarrow \pi_r}$ determines the bucket B_j containing mini-buckets that can be the parent of r.To find π_r, we first search for a mini-bucket q_j^kthat can contain r, that is, var($\lambda_{r \to \pi_r}$) $\subseteq$ var(q_j^k). If such a mini-bucket exists, we set π_r to q_j^k; otherwise, we create a new mini-bucket $q_j^{|Q_j|+1}$ with a scope that matches var$(\lambda_{r \to \pi_r})$ and add it to Q_j. If we don't find a mini-bucket that can be the parent, we repeat this procedure after eliminating x_j from $q_j^{|Q_j|+1}$ until we either find a mini-bucket already in the join tree that can serve as the parent, or var$(\lambda_{r \to \pi_r}) = \emptyset$ in which case the newly added mini-bucket is a root.

After this procedure is complete, we have a valid join tree. The next step is to remove any unnecessary mini-buckets that can be subsumed by the new regions and to update the join tree and the function values of the newly added regions to ensure that the bound is improved.

After each change to the graph, we do one or more rounds of message passing. The number of rounds is controlled by the parameter U. After all the buckets have reached their maximum complexity, we perform the final V rounds of message passing, in order to tighten the bound.

Algorithm 1 SelectMerge: Score the variables

Require: cluster tree wmb, bucket B_i,
  for every variable v_j such that $v_j \in \bigcup_{r_k \in B_j} \text{var}(r_k)$ and $v_j \neq v_i$ **do**
    $S(j) \leftarrow$ MutualInformation(v_i,v_j)
  end for
  $j^* = \text{argmax}_j \; S(j)$
    return v_{j^*}

8.2 Considerations

In this section we look into some of the details of our algorithm, and discuss several methods for each of them.

Algorithm 2 Incremental region selection for WMBE

Require: factor graph (G), bounding parameter *ibound*, initial iterations T, in-between iterations U, final iterations V
Initialize wmb to a join graph using e.g. a min-fill ordering o, uniform weights and uniform messages
for $iter = 1$ to T **do**
  $wmb \leftarrow$ update(wmb)
end for
for each bucket B_i following the elimination order **do**
  repeat
    AddVariable(wmb, B_i)
    for $iter = 1$ to U **do**
      $wmb \leftarrow$ update(wmb)
    end for
  until No more merges possible
end for
for $iter = 1$ to V **do**
  $wmb \leftarrow$ update(wmb)
end for

Algorithm 3 AddVariable:Variable scoring and region addition

Require: join graph wmb, bucket B_i
$v_j \leftarrow$ SelectMerge(B_i)
$R \leftarrow$ AddRegions(wmb, o, v_j)
$wmb \leftarrow$ MergeRegions(wmb, R)

Algorithm 4 AddRegions: Add the larger regions to the graph

Require: The join graph wmb, elimination order o, and the variable x_j to be added to the cliques

```
for each mini-bucket q_i in Q_i do
    Initialize new region q_r with var(q_r) = var(q_i) ∪ v_j and add it to Q_i
    repeat
        R ← R ∪ q_r
        new clique C ← var(q_r)\x_i
        if C = ∅ then
            done ← True
        else
            B_j ← bucket corresponding to the first un-eliminated variable
            in C based on elimination order o
            for each mini-bucket region q_j^k ∈ Q_j do
                if C ⊆ var(q_j^k) then
                    //forward message fits in existing mini-bucket:
                    done ← True
                end if
            end for
        end if
        if not done then
            //Create a new region to contain forward message:
            Initialize new region q_r with var(q_r) = C and add it to Q_j
        end if
    until done
end for
```

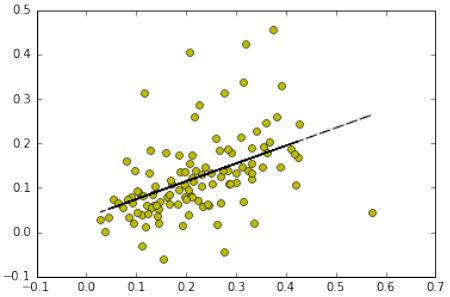


Figure 8.1: Improvement of merge over baseline as a function of the score of the variable added

In Section 8.2.1 we look into our way of measuring correlation between variables. In Section 8.2.2 we research the effect of the amount and the distribution of message passing on the accuracy of the bounds.

8.2.1 Correlation

Our method for tightening the bound on the partition function tries to find the most beneficial variables to add to the buckets. We do this by searching for the variable most correlated to the elimination variable. In order to estimate the relative improvements expected from choosing each variable, we need to use some measure of correlation in line 2 of Algorithm 1. We consider using the mutual information of the two variables,

$$MI(x_1, x_2) = p(x_i, x_j) \log \left(\frac{p(x_i, x_j)}{p(x_i)p(x_j)} \right)$$

where $p(x_i, x_j)$ is the estimate of the joint probability of the variables x_i and x_j, and $p(x_i)$ and $p(x_j)$ are the estimates of the probabilities of the variables, obtained by marginalizing on $p(x_i, x_j)$.

We want to determine whether mutual information is a useful metric. For this purpose, we look at the correlation between the improvement of the bound when adding a variable and the mutual information of the variable added with the elimination variable. Our hypothesis is that adding a variable that's more highly correlated with the elimination variable results in a larger improvement on the bound.

The result, as we can see in Figure 8.1 is that we observe a correlation between the mutual information of the variables and the improvement caused by adding them to the cliques of the bucket. Therefore, we conclude that mutual information is a useful metric that is able to guide the algorithm to select beneficial variables.

8.2.2 Message Passing

Incremental mini-bucket uses message passing on the join graph to guide the selection decision.

In the algorithm we have developed, we use message passing in three different moments that correspond to three loops in Algorithm 2: at the beginning of the algorithm in line 2, after each change in the graph in line 8, and at the end of the algorithm in line 13. The amount of message passing done at each point in the algorithm is determined by the user-controlled parameters T, U, and V, respectively.

We initially intended to use the method for message passing presented in Liu and Ihler [5]. However, we found that the method doesn't result in monotonic tightening of the bound when the graph contains cliques with weights close to 0. Because of this, we decided to use the method from Ping, Liu and Ihler [11], which ensures monotonically increasing bounds. Unfortunately, this method is much slower than non-monotonic message passing.

In this section, we explore some possible strategies for message passing. Specifically, we study the effect of the total amount of message passing on the bound, and we compare the effect of doing most of the message passing at the beginning of the algorithm to the effect of doing more message passing after each change to the graph.

Total amount of message passing

The amount of message passing performed on the graph may have an effect on the choices of the algorithm, making variables more or less attractive. Changes in the choices could be favorable or unfavorable. For example, in a graph where a factor indicates that its two variables are almost equal, these variables initially have very high mutual information. However, the effect that message passing has on this graph is equivalent to making a choice on the value that the two variables should have, that is, one of the possible assignments to the variables will have an increasingly high probability, and the rest will become increasingly unlikely. This would cause the mutual information of the variables to decrease. In this scenario, by doing message passing, we lose the information that the two variables have about each other, which could lead the algorithm to choose a different variable and have an adverse effect on the bound.

However, given a graph with a factor where only one of the possible assignments is likely, message passing would have the effect of spreading the information from the factor, and probably have a beneficial effect on the bound.

We compare the bounds achieved with a small amount of message passing to those obtained when doing more message passing before making any choices. We additionally look into what characteristics of the graphs may lead to message passing having a negative effect on the bound.

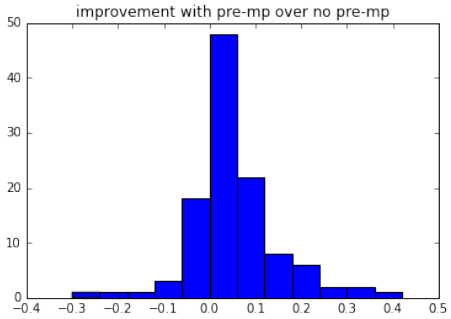


Figure 8.2: Improvement of 200 rounds of message passing before scoring over no message passing before scoring

In order to study the effect of the total amount of message passing, we look at the difference between doing no message passing and doing 200 rounds of message passing at the beginning of the algorithm.

For this experiment, we generate 3-by-3 grid graphs with random factors. For each of the graphs, we generate two instances of their join graph with completely relaxed constraints.

We run 200 rounds of message passing on the second instance. This may affect the relative attractiveness of the variables with respect to those in the first instance. Then, in both instances, we score the variables of the first bucket by measuring their mutual information with the elimination variable of the bucket. We then add the variable with the highest score to all the cliques of the bucket, using the procedure detailed in Chapter 8.

Finally, we do more message passing. In order to keep the two instances comparable, we do 200 more rounds of message passing on the first model. In order to measure the effect of the initial message passing, we now compare the lower bounds obtained by the two models.

As seen in Figure 8.2, the improvement achieved by doing message passing before any scoring is positive on average. The average improvement to the bound is of 0.0464225455864. Therefore, we conclude that doing message passing before starting the scoring is slightly beneficial ($p = 1e - 6$).

We also see that message passing has a negative effect in few cases. We look into what causes this negative improvement and we find that negative improvement usually happens when, in the version that doesn't do the initial message passing, mutual information selects the variable that leads to the highest bound, and in the version with initial message passing, the algorithm doesn't find the best variable.

However, when the best possible variable isn't found without doing mp, doing message passing is useful. In fact, doing message passing is more likely to make the algorithm find the variable that most improves the bound, when it didn't before, which happens with probability 0.436, than it is to lose the

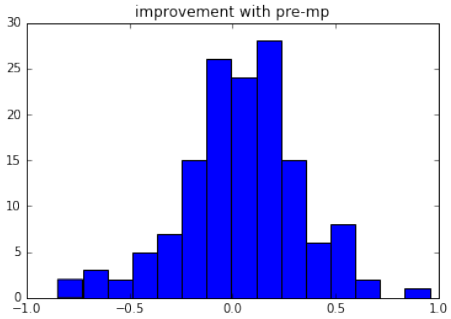


Figure 8.3: Improvement of $U = 15$ over $U = 1$ with same total amount of message passing

variable, which only happens with probability 0.3607.

This leads us to conclude that doing message passing before scoring the variables is beneficial, since it leads to tighter bounds on average and it has a good chance of improving the ability of the algorithm to find the best variable.

Distribution of message passing

Additionally, we looked into the distribution of the message passing in the algorithm. In algorithm 2, there are three different lines where message passing is performed. The first one is at the beginning of the algorithm, before any scoring is done, in line 2. The second line is after each change in the graph, in line 8. The last line the message passing at the end of the algorithm, after all the buckets have reached their maximum size, in order to tighten the final bound. This message passing happens in line 13.

In this section, we want to know whether the amount of message passing performed after each change in the graph -i.e. after each time that we select a variable and add it to each clique in the bucket- in line 8 has an effect on the accuracy of the bounds and whether this effect is beneficial. A larger amount of message passing in line 8 would mean that the information of the change made to the cliques of the bucket is spread to the rest of the graph.

In order to study our question, we design an experiment where we compare two instances of our algorithm where different amounts of message passing are performed in line 8, but the total amount of message passing is left constant, which means that the instance with a low value for parameter U has a higher value for parameter T.

For this experiment, we generate 3-by-3 grid graphs with random factors. For each of the graphs, we generate two instances of their join graph with completely relaxed constraints. We run Algorithm 2 on both instances. In the first instance, we set $T = 15$, $U = 15$, and $V = 15$. In the second instance, we set $U = 1$ and $V = 15$ and we set T such that the total rounds of message

passing in the entire algorithm are the same in both instances.

Figure 8.3 shows a histogram of the difference in bounds obtained by the two methods for each graph. Since we're testing lower bound performance, higher bounds are better. As we can see, the bounds obtained with the higher value of U are tighter on average. The average improvement is of 0.0421, which is significant with $p = 0.087$.

From this experiment we can conclude that doing message passing after each time there's a change in the graph is slightly beneficial on average to the accuracy of the final bonds, and that therefore it's better to spend the available resources for message passing in having a higher U than on a high T.

Chapter 9

Experiments and Results

In this chapter, we compare the results of using our algorithm with two different baselines. In Section 9.1, we compare our method for scoring variables to picking a random variable to add, and find that our method leads to better outcomes. In Section 9.2, we compare our algorithm to scope-based merge, and find that our method leads to better bounds on average. Finally, in Section 9.2.1 we look into some possible reasons why our algorithm may lead to better bounds than scope-based merge.

9.1 Comparison to random variable baseline

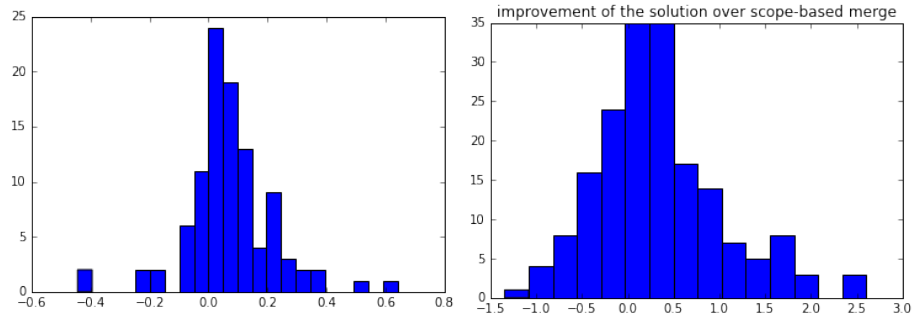
For our first baseline, we compare our method for adding variables based on correlation to a random version.

Because our method identifies variables with high correlation to the elimination variable, which, according to our hypothesis, is a desirable quality for variables that are added to all the cliques to have, we expect our method to lead to tighter bounds than the random version, which doesn't take into account any traits of the variables it selects.

In order to compare the performance of both methods, we generate two instances of the same graph and run algorithm 3 on the join graphs for both of them, setting their first bucket in the elimination order as B_i.

We test the improvement on randomly-generated 3-by-3 grid graphs.

For each of them, we generate two instances of their join graph with completely relaxed constraints. On the first instance, we take the first bucket and score its variables using MI. Then we add the best one to the mini-buckets. On the second instance, we randomly pick a variable and add it to the mini-buckets. We then do some message passing on both instances, and compare their bounds.



(a) (b)

Figure 9.1: (a) Improvement of using MI over choosing a random variable. (b) Improvement of the algorithm over scope-based merge

As seen in Figure 9.1a, using MI to choose the right variable leads to an improvement over making a random choice (p=1.54e-05). The average improvement to the bound is of 0.068.

This experiment allows us to conclude that our method performs better than random on average.

9.2 Comparison to scope-based merge

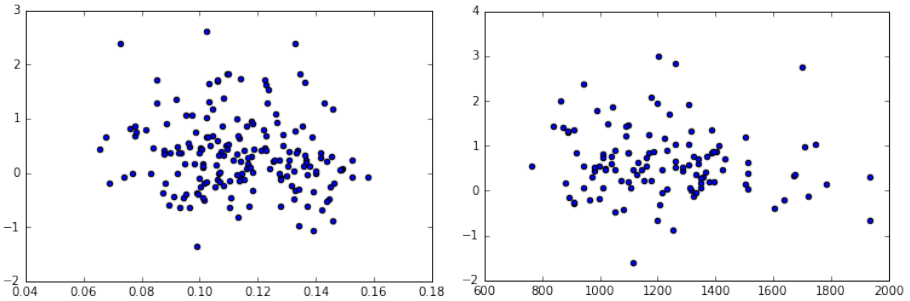
We test the improvement of our proposed method over a different baseline. This baseline is incremental mini-bucket with scope-based merge.

Our initial hypothesis is that neither method should be better than the other on average. We additionally expect that our method will tend to achieve a higher improvement over scope-based merge when the graphs have more highly correlated variables.

In order to test the improvement, we randomly generate 5-by-5 grid graphs. For each of them, we generate two instances of their join graph with completely relaxed constraints. We run Algorithm 2 on the first instance, and the incremental mini-bucket algorithm from Forouzan and Ihler [2] using a scope-based scoring function on the second instance.

As seen in Figure 9.1b, using MI to choose the right variable leads to an improvement over scope-based merge. The average improvement to the bound is of 0.339.

From this experiment we can conclude that our method performs better than scope-based merge. This contradicts our initial hypothesis, since we didn't expect either of the methods to perform better than the other. In Section 9.2.1, we look at two possible causes for this unexpected result.



(a) (b)

Figure 9.2: (a) Improvement of merging over baseline as a function of the average MI of the variables chosen. (b) Improvement of merging over baseline as a function of the total size of the model

9.2.1 Possible causes of improvement

In this section, we test two hypotheses that may help explain why our algorithm obtains tighter bounds than scope-based merge. We find that both hypotheses are unsupported by our data and therefore we don't know the real cause of the improvement.

Does a higher average MI lead to a higher improvement over scope-based merge?

Our first hypothesis is that by using correlation between variables to make choices, we are making smarter choices than scope-based merge, which doesn't take the content of the factors of the distribution into consideration. To test this, we measure whether in graphs where the variables have high average mutual information using our algorithm leads to a higher improvement over scope-based merge than in graphs with low average mutual information.

We run both our correlation-based algorithm and scope-based incremental mini-bucket on randomly generated 5-by-5 grid graphs and, as seen in Figure 9.2a, we find that there is no correlation between the average MI of the graphs and the advantage of our algorithm over scope-based merge, and therefore conclude that we can discard our hypothesis.

Do larger cliques lead to tighter bounds?

Our second hypothesis is based on the fact that our algorithm tends to generate uniformly large cliques that are close to the maximum size, while the scope-based merge tends to generate one large clique and some smaller cliques. Our hypothesis is that the larger size of the model created by our algorithm allows it to reach tighter bounds. In order to test this hypothesis, we measure whether

there is a correlation between the difference in size of the two models from the same graph and the improvement of the bound reached by our algorithm over the bound reached by scope-based merge.

We run both our algorithm and scope-based merge on randomly generated 5-by-5 grid graphs and, as seen in Figure 9.2a, we find that there is no correlation between the size difference and the advantage of our algorithm over scope-based merge, and therefore conclude that we can discard our hypothesis.

Chapter 10

Conclusions

Because graphical models are a powerful tool for knowledge representation and reasoning, they are widely used in a variety of domains. However, because of the inherent complexity of inference on graphical models, approximate inference algorithms are increasingly relied on.

In this project, we started from an approximate inference framework that merges aspects of mini-bucket elimination and variational bounding techniques into an incremental algorithm where increasingly complex regions are added to a model that starts with the smallest possible regions.

We focused on a known weakness of existing approximate inference algorithms, the tightness of lower bounds. We attempted to address this weakness by developing new strategies for region selection in incremental mini-bucket.

In Chapter 7 we presented a strategy for region selection based on the idea that highly correlated variables should be reasoned about together. In our strategy, for each bucket of the join graph, we iteratively find the variable that has the highest correlation to the elimination variable, and add it to the scope of all the cliques of the bucket. This ensures that the information that the two variables have about each other is taken into account in the approximation.

In Chapter 8 we discussed the implementation of our strategy within the framework of incremental mini-bucket. We described the changes made to adapt the incremental mini-bucket algorithm to our strategy, and we discussed some aspects of the implementation.

We discussed the method for measuring correlation and proved that higher mutual information correlates to higher improvement of lower bounds when adding a variable to the cliques of a bucket. We additionally discussed our message passing method and strategy. We decided to use the gradient descent algorithm from Ping, Liu, and Ihler [11] in order to ensure monotonic tightening of the bound.

We also studied the amount and distribution of message passing in the algorithm and found that some message passing at the start of the algorithm leads to tighter bounds than not doing any and that doing some message passing after each change in the bound leads to tighter bounds than doing all message passing at the beginning and end of the algorithm.

In Chapter 9 we discussed our results, and looked into some possible explanations for them. We first compared the bounds obtained with our algorithm to the bounds obtained with two different baselines.

First, we compared the bounds obtained by picking and adding a variable using our strategy to the bounds obtained by random choice, and found that our strategy leads to tighter lower bounds on average. Additionally, we compare our algorithm to scope-based incremental mini-bucket and find that our algorithm obtains tighter lower bounds on average.

Finally we research some possible reasons for our algorithm to obtain tighter bounds than scope-based incremental mini-bucket. We test whether our algorithm creating cliques with more highly correlated variables on average may lead to the improvement and whether the larger cliques that our algorithm tends to create may lead to a more expressive model and tighter bounds. We find that we can discard both hypotheses

Chapter 11

Future work

There are some aspects of the project that could be further developed. In this chapter, we list some of them.

In Section 9.2.1 we explored possible reasons why our algorithm might achieve tighter bounds than scope-based incremental mini-bucket. However, we found that our test results didn't bear out our hypotheses, and we consequently don't know the actual reason for the difference. Therefore, more work should be done to research the causes of the improvement.

Additionally, we ended up focusing on and developing only one of our ideas for algorithms. It would be an interesting follow-up to this project to develop our other ideas and compare their results to the ones we obtained here.

In this project, we developed and tested an algorithm for approximate inference, but we only developed a Python implementation designed for easy tweaking and testing. For this reason, we couldn't research the time cost of our algorithm with sufficient accuracy. A more complete assessment of the performance of our algorithm would require an evaluation taking into account its time costs.

It would also be interesting to see an implementation of our algorithm used by real users for real problems.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Wikipedia, "Graphical model", 2016. [Online]. Available: `https://en.wikipedia.org/wiki/Graphical_model`. [Accessed: 02 - Mar - 2016].

[2] A. Ihler and S. Forouzan, "Incremental Region Selection for Mini-bucket Elimination Bounds", in Uncertainty in Artificial Intelligence (UAI), Amsterdam, Netherlands, 2015, pp. 268-277.

[3] R. Dechter, "Bucket elimination: A unifying framework for reasoning", Artificial Intelligence, vol. 113, no. 1-2, pp. 41-85, 1999.

[4] R. Dechter and I. Rish, "A scheme for approximating probabilistic inference.", in Uncertainty in Artificial Intelligence (UAI), Providence, Rhode Island, USA, 1997, pp. 132–141.

[5] "Bounding the partition function using hölder's inequality.", in International Conference on Machine Learning (ICML), Bellevue, Washington, USA, 2011, pp. 849–856.

[6] Q. Liu, "Reasoning and Decisions in Probabilistic Graphical Models - A Unified Framework", Ph.D, University of California, Irvine, 2016.

[7] E. Rollon and R. Dechter, "Evaluating partition strategies for mini-bucket elimination.", in International Symposium on Artificial Intelligence and Mathematics (ISAIM), Fort Lauderdale, Florida, USA, 2010.

[8] A. Choi and A. Darwiche, "Relax, compensate and then recover.", in New Frontiers in Artificial Intelligence: JSAI 2008 Conference and Workshops, Asahikawa, Japan, June 11-13, 2008, Revised Selected Papers (Lecture Notes in Computer Science), 1st ed., Springer, 2010, pp. 167–180.

[9] D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola and Y. Weiss, "Tightening LP relaxations for map using message passing.", in Uncertainty in Artificial Intelligence (UAI), Helsinki, Finland, 2008.

[10] R. Dechter, *Constraint Processing*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2003.

[11] W. Ping, Q. Liu and A. Ihler, "Decomposition Bounds for Marginal MAP", in Neural Information Processing Systems (NIPS), Montreal, Canada, 2015.

[12] S. Russell and P. Norvig, *Artificial intelligence a modern approach.*, 3rd ed. New Jersey: Prentice Hall, 2003.

[13] "EIA - State Electricity Profiles", *Eia.gov*, 2016. [Online]. Available: http://www.eia.gov/electricity/state/california/. [Accessed: 26- Jul- 2016].