SYMMETRY AWARE INFERENCE AND DECISION MAKING IN PROBABILISTIC MODELS

ANKIT ANAND

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI

JULY 2019

SYMMETRY AWARE INFERENCE AND DECISION MAKING IN PROBABILISTIC MODELS

by

ANKIT ANAND

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of Doctor of Philosophy

to the

INDIAN INSTITUTE OF TECHNOLOGY DELHI

JULY 2019

Certificate

This is to certify that the thesis titled **Symmetry Aware Inference and Decision Making in Probabilistic Models** being submitted by **Mr. Ankit Anand** for the award of **Doctor of Philosophy** in Department of Computer Science and Engineering is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Computer Science and Engineering, Indian Institute of Technology Delhi**. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma unless otherwise stated explicitly. In particular, work done in Chapters 3, 4, 7, 8 and 9 were done jointly with undergraduate students. Work done in Chapter 5 was done jointly with a master's student. In each case, the part done by the collaborators appeared in their respective bachelor's or master's theses.

Parag Singla
Associate Professor
Department of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi- 110016

Mausam
Associate Professor
Department of Computer Science and Engg.
Indian Institute of Technology Delhi
New Delhi- 110016

Acknowledgements

*"***No dream is ever chased alone.***"*

Rahul S. Dravid, Indian Cricketer

As I stand near the end of my journey of formal education, I would like to look back and express thanks to some really great people whom I met on the way and without whom, this long path would have looked really dull.

First and foremost, I will like to express thanks to my parents who have supported me in my education and career through out my life. The liberty and the support they have given to me cannot be expressed in words. This thesis is dedicated to both of them for their effort at back end. I am also thankful to my brothers, bhabhi, chacha ji and chachi ji for always believing me and be there, whenever I needed them.

On the other side, there were two people who stood by me, weathering each storm and enjoying each breeze in my personal and professional life during these 5 years. I was fortunate and grateful to be advised by Parag Singla and Mausam. I enjoyed close personal and professional communication with both of my advisors during these past four years and a half. There has always been an inspiration in whatever way I look at the lives of both of them. The dedication shown by Parag in teaching, research, and importantly, to have right, calm and humble attitude even in most complicated situations has redefined the meaning of how to do any task in life for me. Mausam has always inspired me to go beyond my limits and much of my current state personally and professionally is owed to the honest and critical feedback given by him during this journey. I feel really grateful for the effort and belief they showed in me.

I am thankful to Prof. Alan Fern in hosting me for a research internship and providing me the flavour of international research culture without which this Ph.D has been incomplete. I am also thankful to Jesse Hostetler for the helpful research discussions during this summer of 2016.

I am thankful to Tata Consultancy Services for funding my Ph.D during past 4 years and numerous funding agencies like MSR India, IARCS, Xerox and various conference travel grants which made my travel possible to all the conferences in India or abroad.

I worked closely with many students during this thesis with many of whom I share great personal bonds till date. This direction of research was started alongside Aditya Grover. The numerous discussions in GCL, subway dinners and our Buenos Aires trip has been a great learning experience for me. Aditya has always been a friend cum collaborator I can personally turn to for any advice. The next person in line was Ritesh Noothigattu whose strict disciplinarian lifestyle and always happy face is still fresh in memory. The early morning GCL meetings was a great time for me. I am also thankful to Haroun Habeeb and Tapas Jain for the great technical discussions we had. In the last part of my Ph.D, I worked with Gagan Madan. I shared great friendship with Gagan during the past one year and his talent and humbleness has always motivated me. Gagan had always been available to discuss anything even when he had graduated. I also worked with Yatin Nandwani during the past one year. I really enjoyed our technical discussions and am really thankful to you and Shagun for the homemade food I got in the past one year.

The end of my Ph.D marks the end of my hostel life as well. During these years, away from home, I made many friendships which have supported me or provided me some awesome breaks. This is not limited to people from IIT Delhi but traverse to many people beyond this campus boundary. I am extremely thankful to Anurag Murty, Priyanka Singla, Haseen Zahera Rizvi and Ishani Mahajan who have always been a call away whenever I needed to discuss anything and everything. I was fortunate and grateful to have you people around me during these years which included many hills and troughs but your phone calls have always restored the confidence in me. Thanks for believing and trusting me and making myself believe in me.

I am also thankful to my Ph.D friends for the daily jokes, discussions, umpteen dinners which gave me a family atmosphere through out this journey. I feel specially thankful to Dinesh Khandelwal, Himanshu Jain, Prachi Jain, Yashoteja Prabhu, Happy Mittal, Neetu Jindal, Rajesh Kedia and Sakshi Tiwari. I am also thankful to Kunal Dahiya, Dilpreet Kaur, Anup Bhattacharya, Vishal Sharma, Arindam Bhattacharya, Suvam Patra, Syamantak Das, Saurabh Tripathy, Rajshekar K, Sandeep Chandran, Prathmesh Kallurkar, Nikhil K Gayathri A, Saurabh Goyal, Swarnadeep Saha, and Harshit with whom I have shared some memories at some point in this journey. Forgive me if I have missed anybody. I will cherish these memories through out my life.

I am thankful to Sai Rajeshwar for making my trip to Montreal possible and for hosting my stay during the summer school visit. I am also thankful to Ashish Kumar for the time we shared at IIT and guiding me during job search and listening to my job talk many times and giving constructive feedback.

I am thankful to my school, bachelor and masters friends many of whom have not let me pay a single penny during all these years, even in foreign countries saying, “Bhai tu student hai, tu pay nahi karega”. This kind of care really meant a lot to me. Heartfelt thanks to Mohit

Dhingra, Aakriti Gupta, Sai Kiran Korwar, Vivek K, Sanjeev, Hitesh Chawla, Sachin Miglani, Mohit Virmani, Ankit Thareja and Gaurav Pahwa (Sydney Memories) to name a few.

Ph.D has not been limited to academic journey. I developed many new hobbies and pursued those greatly during this time. Firstly, thankful to all people in "B-Twin Diaries", our cycling group where I made many trips in and around Delhi and shared special memories. The second group is Cricket group led by Happy Mittal which provided refreshing breaks during the last two years. I am also thankful to Himanshu Jain for pushing me to go for "Buran Ghati Trek" during the last year which was really special.

There are numerous people who will still be missing in this list but would have made this journey really special.

Last but not the least, I am thankful to the Almighty for creating this beautiful journey for me and making me meet many special people with whom I will share great bonds through out my life.

Ankit Anand

Abstract

Many contemporary artificial intelligence algorithms fail to scale to large problems, because the problem sizes typically increase exponentially with the number of features. Even though a problems size may be too large, there often exist repeated sub-structures, resulting in symmetries or other kinds of invariances within the problem. These symmetries and invariances, when identified accurately, can substantially save on downstream computation time. This thesis proposes the idea of symmetry aware AI, in which AI and ML algorithms compute these symmetries, and then use them for improving their efficiency and performance. Specifically, this thesis studies symmetry aware AI in the context of three different AI problems: (i) probabilistic inference in probabilistic graphical models, (ii) sequential decision making under uncertainty and (iii) structured output prediction in computer vision.

First, in probabilistic inference, we study different types of state symmetries to help speedup marginal inference. Specifically, we define novel notions of contextual symmetries [Anand et. al., IJCAI 16], variable-value and non-equicardinal symmetries [Anand et. al., AISTATS 18] and block-value symmetries [Madan et. al. UAI 2018], and propose algorithms to compute these. Further, we incorporate these symmetries for improving the mixing time of Markov Chain Monte Carlo (MCMC) methods.

Second, we define the ASAP (Abstraction of State-Action Pairs) framework, which extends and unifies past work on domain abstractions in sequential decision making under uncertainty. It holistically aggregates both states and state-action pairs, thereby, identifying significantly more symmetries than previous work. We also propose two novel algorithms: ASAP-UCT [Anand et. al., IJCAI 15] and OGA-UCT [Anand et. al., ICAPS 16], which use ASAP symmetries within a UCT framework, thus, combining strengths of online planning with domain abstractions.

Last but not the least, we explore the use of symmetries in state-of-the-art algorithms for real-world computer vision problems. We propose a novel coarse-to-fine symmetry aware template [Habeeb et. al., IJCAI 17] to exploit symmetries in structured output prediction tasks of stereo-vision and image segmentation. Our approximate notion of top-k label heuristic is robust across problems and provide significant time gains in near state-of-the-art algorithms for these tasks.

सार

प्रोबेबिलिस्टिक मॉडल्स में सिमिट्री अवेयरनेस अनुमान एंड डिसीजन मेकिंग

कई समकालीन आर्टिफिशियल इंटेलिजेंस (ए.आई.) (अर्थात कृत्रिम बुद्धि) एल्गोरिदम बड़ी समस्याओं के पैमाने पर विफल होते हैं, क्योंकि समस्या के आकार आमतौर पर विशेषताओं की संख्या के साथ तेजी से बढ़ते हैं। भले ही किसी समस्या का आकार बहुत बड़ा हो, लेकिन अक्सर उप-संरचनाएँ मौजूद रहती हैं, जिसके परिणामस्वरूप समस्या के भीतर समरूपता होती है। इन समरूपताओं की जब सही पहचान की जाती है, तो डाउनस्ट्रीम गणना समय पर काफी बचत हो सकती है। यह थीसिस समरूपता के बारे में जागरूक ए.आई. के विचार का प्रस्ताव करती है, जिसमें ए.आई. और एम.एल. एल्गोरिदम इन समरूपताओं की गणना करते हैं, और फिर उनकी दक्षता और प्रदर्शन में सुधार के लिए उनका उपयोग करते हैं। विशेष रूप से, यह थीसिस तीन अलग-अलग एआई समस्याओं के संदर्भ में एआई को समरूपता से अवगत कराती है: (i) संभाव्य ग्राफिकल मॉडल में संभाव्यता का अनुमान, (ii) अनिश्चितता के तहत क्रमिक निर्णय और iii) कंप्यूटर विज़न में संरचित आउटपुट भविष्यवाणी |

सबसे पहले, संभाव्य निष्कर्ष में, हम मार्जिनल अनुमान में मदद करने के लिए विभिन्न प्रकार के अवस्था समरूपता का अध्ययन करते हैं। विशेष रूप से, हम संदर्भ-गत सिमिट्रीज़ [आनंद एट अल।, IJCAI '16], की नई धारणाओं को परिभाषित करते हैं। वैरिएबल-वैल्यू और नॉन-इक्विकार्डिनल सिमिट्रीज़ [आनंद एट अल, AISTATS '18] और ब्लॉक-वैल्यू समरूपता [मदान एट अल UAI 2018], और इनकी गणना के लिए एल्गोरिदम का प्रस्ताव करते हैं। इसके अलावा, हम मार्कोव चेन मोंटे कार्लो (MCMC) विधियों के मिश्रण समय में सुधार के लिए इन समरूपताओं को शामिल करते हैं।

दूसरा, हम ASAP (स्टेट-एक्शन पेयर की एब्सट्रैक्ट) रूपरेखा को परिभाषित करते हैं, जो अनिश्चितता के तहत क्रमिक निर्णय लेने में डोमेन अमूर्त पर पिछले काम का विस्तार और एकीकरण करता है। यह दोनों स्टेट (अवस्था) और स्टेट-एक्शन जोड़े को समग्र रूप से एकत्रित करता है, जिससे पिछले कार्य की तुलना में काफी अधिक समरूपता की पहचान होती है। हम दो नए एल्गोरिदम का प्रस्ताव भी देते हैं: एएसएपी-यूसीटी [आनंद एट अल, IJCAI '15] और OGA-UCT [आनंद एट। अल।, ICAPS '16], जो एक UCT ढांचे के भीतर ASAP समरूपता का उपयोग करते हैं, इस प्रकार, डोमेन सार के साथ ऑनलाइन योजना की ताकत का संयोजन करते हैं।

अंतिम लेकिन कम से कम नहीं, हम वास्तविक दुनिया की कंप्यूटर दृष्टि समस्याओं के लिए अत्याधुनिक एल्गोरिदम में समरूपता के उपयोग का पता लगाते हैं। हम एक मोटे-से-ठीक समरूपता वाले जागरूक टेम्पलेट [हबीब एट अल, IJCAI '17] का प्रस्ताव करते हैं। स्टीरियो-दृष्टि और छवि विभाजन के संरचित आउटपुट भविष्यवाणी कार्यों में समरूपता का फायदा उठाने के लिए टॉप-के लेबल हेयुरिस्टिक की हमारी अनुमानित धारणा समस्याओं के प्रति मजबूत है और इन कार्यों के लिए अत्याधुनिक एल्गोरिदम के पास महत्वपूर्ण समय लाभ प्रदान करती है|

Contents

List of Figures

List of Tables

Part I

Prologue

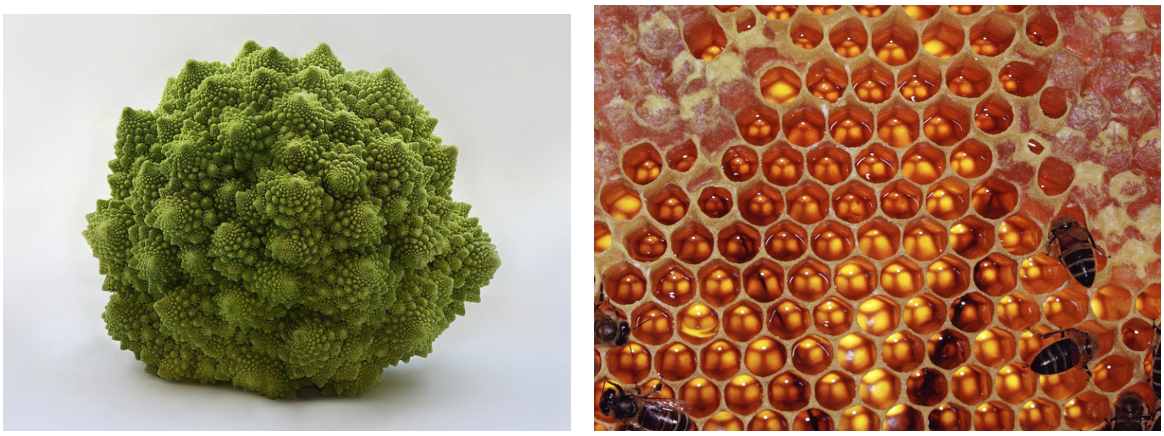
Chapter 1

Introduction

Symmetry, as wide or narrow as you may define its meaning, is one idea by which man through the ages has tried to comprehend and create order, beauty, and perfection.

Hermann Weyl, German Mathematician

The entire universe is composed of a wide range of objects, organisms, and environments each having varying degrees of complexity. One of the interesting rules followed by nature while creating complex objects (environments) is that most of these objects are carved by joining simpler objects repeatedly in addition to the specific mutation or variation unique for that object. Undoubtedly, the resulting objects are unique in themselves but essentially have a lot of similarity in structure underlying them. This could be seen in the design of plants like Romanesco Broccoli which has the same pattern repeated again and again or the design of honeycombs by bees where a hexagonal shape is repeated in a plane (Figure: 1.1). In fact, due to these similarities, many properties of these objects remain unchanged when they undergo certain transformations. Such properties are called *invariances* of that object under those transformations. Invariances have been well studied in the domains of mathematics and physics since the last century. Symmetry groups [Miller, 1973] and fractals [Barnsley, 2014] have been the key mathematical tools to formally represent these invariances. The far-reaching impact and importance of these invariances can be gauged further from the beautiful and seminal work of early 1900s by Emmy Noether, *Noether's theorem* [Byers, 1998; Noether, 1971]. The key idea of Noether's theorem is that for every invariance in nature, there is a corresponding law of conservation. For example, two objects collide in space in a unique manner (if they have identical respective attributes), irrespective of their coordinates, due to the law of conservation of momentum. Similar ideas hold for other invariances like the invariance in time is attributed to the law of conservation of energy while rotational invariance is attributed to the law of conservation of angular momentum.



(a) (b)

Figure 1.1: Invariances in Nature **(a)** Structure repetition in Romanesco Broccoli **(b)** Hexagonal Symmetry in Honeycomb

Most of the contemporary Machine Learning (ML) and Artificial Intelligence (AI) research owes its origin to ideas in physics and mathematics. In a similar way, this thesis is an attempt to leverage the invariances present in nature to improve the efficiency of AI algorithms. The process begins with identifying and discovering these invariances automatically from the data or problem definition. This is followed by using the identified invariances in appropriate ways to improve the efficiency, scalability or quality in AI systems.

Why are invariances needed in AI? With the recent advancements in automation and AI, many of the simple tasks performed by humans are being overtaken by autonomous agents. Though some of our day-to-day tasks have been automated, the grand goal of AI where AI systems are able to perform these tasks with the same degree of efficiency, clarity, and accuracy as humans do, is still quite far. One of the key roadblocks in addressing this goal is *the curse of dimensionality*. Scaling the existing approaches to real world instances has always been a huge challenge and most of the methods fail to perform efficiently as the problem size increases. This thesis argues that although problems and environments are big and complex, often they have some structure or invariances, and these underlying invariances have a great potential to improve the efficiency and scalability of multiple AI and ML algorithms. Specifically, we call these invariances as *symmetries* present in the problem. There are multiple ways of exploiting symmetries in AI problems: sometimes they are used to reduce the problem size as a preprocessing step, while at other times, they become an integral part of the algorithm. Importantly, exploiting these symmetries can provide orders of magnitude gain in time in many time-critical tasks e.g., decision making in autonomous agents, in environments where real-time performance is required.

Before delving into the specific technical details of symmetries in AI and ML algorithms,

Figure 1.2: Street map of Manhattan in New York depicts perfect grid of streets and avenues. City map reveals many symmetric states and paths which can be potentially used for efficient planning. Source: http://www.joemygod.com/2011/03/21/200-years-of-manhattan-street-grid/

we initially discuss the symmetries exhibited in various AI and ML problems.

- **Invariances in Game Playing:** One popular way to evaluate progress in AI has been to measure its success in the field of game playing [Campbell *et al.*, 2002; Silver *et al.*, 2016]. Some of the best examples of exploitation of invariances come from this field. Most of the board games have many board configurations which are symmetric and equivalent to each other with respect to the final outcome of the game. As an example, one can trivially observe that many board configurations are equivalent in the game of chess (for e.g., a mirror image).

- **Symmetries in Path Planning:** One important capability of any AI agent is to navigate efficiently and accurately in the world. A key module for navigation is path planning which is still inefficient for large problems. However, many of these navigation maps have a number of structural invariances which are potentially useful for efficient planning. For example, the famous street grid of Manhattan in New York and the city map of the Union Territory of Chandigarh in India are depicted in Figures 1.2 and 1.3 respectively. Both the maps are examples of a perfect grid and traversing from point A to point B has many equivalent paths and correspondingly equivalent states in terms of distance

Figure 1.3: Street map of city of Chandigarh, India. The city is perfect grid like Manhattan, New York which can be utilized for efficient path planning and navigation. Source: http://chandigarh.gov.in/knowchd_map.htm

traversed. These equivalences could be of great value in efficient planning. Though navigation maps of many cities or terrains are not perfect grids, many other approximate symmetries still exist and have the potential to provide huge gains during path planning.

- **Pixel Labelling in Computer Vision:** Consider the task of pixel labelling for an image, e.g., semantic image segmentation [Kohli *et al.*, 2013] or stereovision [Mozerov and van de Weijer, 2015]. One may envisage this task as a classification task over each pixel (or structured prediction problem). This can be computationally very expensive. However, many of the pixels in the real world scenarios behave identical or approximately identical to each other and are likely to obtain the same label. One can merge these identical pixels in a single pixel to obtain a smaller model, thereby saving computation. Our work [Habeeb *et al.*, 2017], merges such pixels to obtain impressive speedups in the tasks of stereovision and semantic image segmentation.

Classification of past work on symmetry: Past work has shown that symmetries and invariances can be used to improve the efficiency and scalability in AI problems by exploiting repeated sub-structures. Symmetries have been studied in many AI problems like game playing [Schiffel, 2010; Koriche *et al.*, 2017], structured prediction [Nath and Domingos, 2010],

probabilistic inference [Getoor and Taskar, 2007] and decision making [Givan *et al.*, 2003; Ravindran, 2004]. Particularly, in the probabilistic inference community, this idea of exploiting symmetries for efficient inference is called *Lifted Inference* [Kimmig *et al.*, 2015]. There are multiple ways to classify past works on how symmetries have been used in different AI algorithms.

- **Algorithm Oblivious or Algorithm Dependent Symmetries:** Past work which uses symmetries for improving efficiency can broadly be classified in two categories: 1) those works which study symmetries as a generic notion, independent of the downstream inference algorithm to be used and 2) those works which study special kind of symmetries tied to a specific algorithm. Most of the initial works [Givan *et al.*, 2003; Ravindran, 2004] in sequential decision making proposed notions of symmetry of states and actions which are oblivious to the planning or reinforcement learning algorithm to be used. On the other hand, works in lifted probabilistic inference focused on symmetry exploiting (lifted) variants of popular inference algorithms like variable elimination [Poole, 2003], belief propagation [Singla and Domingos, 2008; Kersting *et al.*, 2009] and knowledge compilation [Van den Broeck *et al.*, 2011]. There have also been many recent works [Niepert, 2012] which studied algorithm-oblivious notions of symmetries in Probabilistic Graphical Models (PGMs).

- **Exact or Approximate Symmetries:** Another distinguishing factor in past works is whether to exploit exact symmetries or approximate symmetries. Though exact symmetry notions are based on sound mathematical principles like bisimulation [Givan *et al.*, 2003] and/or graph automorphisms [Ravindran, 2004], it is not useful for many real world scenarios. There are many initial works which focused on exploiting exact symmetries. The solutions provided by algorithms exploiting exact symmetries suffer zero loss in quality with respect to the solutions obtained by base algorithms. However, because exact symmetries are difficult to find in real world, many subsequent works took inspiration from exact symmetries and showed application of approximate symmetries in a variety of tasks. Our work spans both exact and approximate symmetries. We defined exact notion of state-action pair symmetries in Markov Decision Processes (MDPs), and defined novel notions of exact state symmetries [Anand *et al.*, 2016a; Anand *et al.*, 2017; Madan *et al.*, 2018] in PGMs, based on graph automorphism. On the other hand, we also devised relaxation for state-action pair symmetries for its application in Monte Carlo Tree Search (MCTS) algorithms [Anand *et al.*, 2015a; Anand *et al.*, 2016b]. We also developed novel heuristics to merge approximately symmetric pixels in PGM based computer vision models [Habeeb *et al.*, 2017].

- **Unit of Symmetry Computation:** One important decision while using symmetries for any problem is to determine the basic unit of equivalence which could be useful for that particular problem. While the equivalence unit is dependent on the domain, it can vary for each algorithm within the domain as well. For e.g., in planning, states [Givan *et al.*, 2003; Ravindran, 2004] and state-action pairs [Anand *et al.*, 2015b; Anand *et al.*, 2015a] are the possible units of equivalence while one can study variable equivalence or state equivalences in PGMs. There are many works which exploit variable symmetries [Singla and Domingos, 2008; Kersting *et al.*, 2009] while there are also some recent works [Bui *et al.*, 2012; Niepert, 2012] which exploit state equivalences using graph automorphisms.

- **Application Domain:** A natural way to classify past work is based on the application domain. There have been works on exploitation of symmetries in decision making [Givan *et al.*, 2003; Ravindran, 2004], probabilistic inference [Kimmig *et al.*, 2015], constraint satisfaction [Cohen *et al.*, 2006], etc. Many of these works have foundations in similar ideas (like bisimulations [Givan *et al.*, 2003] and graph homomorphisms [Ravindran, 2004]) yet they have been studied independently. In this thesis, we focus on two important tasks in AI: decision making and inference in probabilistic models. Further, we also study exploitation of symmetries for PGM-based models in computer vision.

This thesis focuses particularly on probabilistic models for decision making and inference though many of the ideas are general and may be applicable in other AI problems as well. Our choice of probabilistic models is motivated by the fact that these models capture most of the semantics of the real world by modeling uncertainties present in the world as probabilities. Also, these models are inherently complex and have been models of great interest in the AI community to speed up computation. They have been used in various applications in computer vision [Szeliski *et al.*, 2008b] and natural language processing [Blei *et al.*, 2003].

Although, we have classified most of the past work on factors stated above, we next propose the key characteristics one should consider while incorporating symmetries in any artificial intelligence or machine learning problem. We encapsulate all the above work under the general class of what we call *symmetry aware algorithms*, which we describe in the next section.

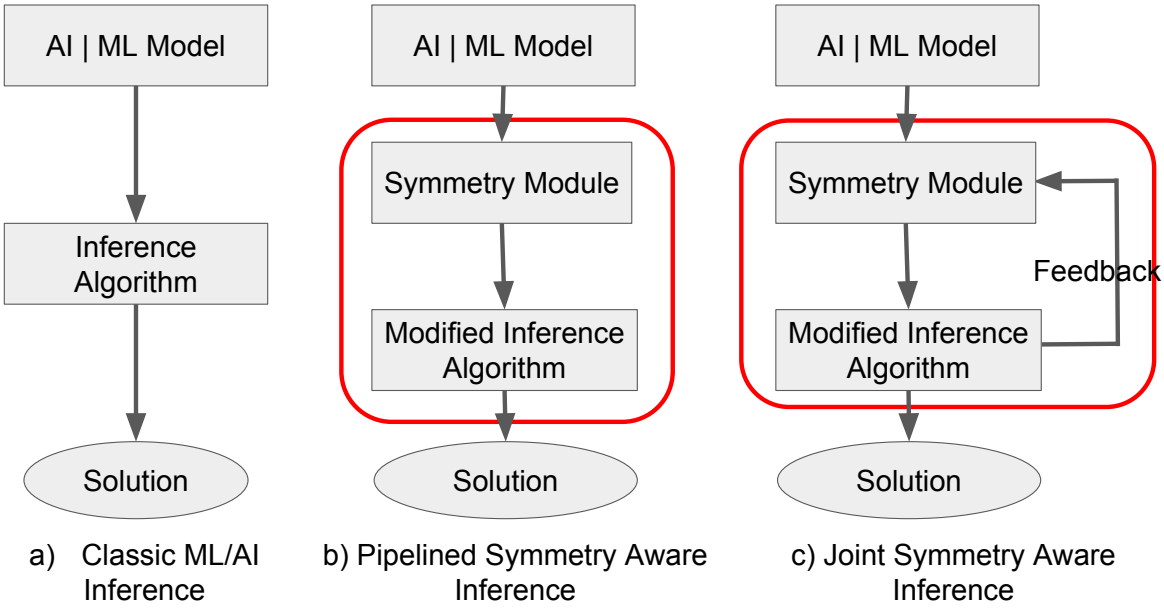


Figure 1.4: **(a)** Classic ML — AI inference model **(b)** Pipelined Symmetry Aware Inference Model: Symmetry Module is pipelined between ML model and Modified Inference procedure **(c)** Joint Symmetry Aware Inference Model: Symmetry Module receives feedback from the inference module to compute better and better symmetries which are fed back to inference module again and again

1.1 Symmetry Aware Algorithms

In general, we call all those algorithms which utilize symmetries to improve their efficiency as *symmetry aware algorithms*. However, we classify symmetry aware algorithms in two categories: *pipelined* and *joint* symmetry aware algorithms depending on how their symmetry module interacts with the inference procedure. Figure 1.4 (a) shows the classic ML/AI paradigm where a learned machine learning model is given as an input to an inference procedure which makes predictions. Figure 1.4 (b) and (c) show the two types of symmetry aware algorithms: pipelined and joint. In pipelined symmetry aware algorithms, the symmetry module computes symmetries as a pre-processing step. Computed symmetries are then given as input to a possibly modified inference procedure which uses these symmetries. In joint symmetry aware algorithms, symmetry computation module closely interacts with the inference procedure. There is an additional feedback from the modified inference procedure to symmetry computation module. The feedback guides the symmetry module regarding further improvements in generated symmetries, which in turn are used by modified inference procedure. This is particularly useful in systems based on approximate symmetries. We discuss both pipelined and joint symmetry aware systems in this thesis. Next, we identify certain key characteristics which are useful and effective in designing modern symmetry aware AI systems. Our works incorporate one or more of these characteristics while designing symmetry aware algorithms for various problems.

Characteristics of Symmetry Aware Algorithms: We identify three important characteristics which are important in the design of symmetry aware algorithms for contemporary ML and AI problems.

- **On-the-fly Symmetry Computation:** Firstly, the symmetry aware algorithms should be able to find the symmetries (present in the domain) on-the-fly, within the base algorithm. Many contemporary AI algorithms have *anytime* behavior where a solution is available at all times and the solution quality is improved over time. If there is an initial symmetry computation phase as a preprocessing step, it destroys the anytime characteristic of the algorithm where no solution is available during initial symmetry computation phase. This is important since symmetry computation routine can be computationally expensive.

- **Incremental Symmetry Improvement:** Since the algorithm is computing the symmetries on-the-fly, it computes symmetries without looking at the whole problem structure, and hence, the initial symmetries may be wrong (as they are based on partial information). The desirable characteristic to alleviate this problem is that the algorithm should be able to incrementally improve symmetries. It should reuse the wrongly computed symmetries as well as inference algorithm's behavior to compute more accurate symmetries present in the problem over time.

- **Adaptive Symmetry Computation:** Most of the time it is observed that symmetry finding routines are computationally expensive and are uniformly applied over the whole problem space. Such routines become a bottleneck in the overall efficiency of symmetry aware algorithms. This problem can be alleviated by *adaptive symmetry computation* where symmetry computation routine spends more time only in promising parts of problem space instead of uniform symmetry computation over the whole problem space.

Having described the basic framework of symmetry aware AI systems, we describe how we apply this symmetry aware template to 3 different problems in AI : i) Marginal inference in Probabilistic Graphical Models (PGMs) ii) Sequential decision making (AI planning and reinforcement learning) and iii) Structured output prediction in computer vision.

1.2 State Symmetries in Probabilistic Graphical Models

Most of the real world AI problems involve uncertainty, and probability distributions have been used to model this uncertainty. A crucial capability of any AI system is to efficiently perform reasoning on these probability distributions. Particularly, given a set of random variables and a probability distribution defined over them, the task of probabilistic inference is to reason different queries on this joint probability distribution. The queries may vary from finding

the probability of a small subset of variables called Marginal Inference [Koller and Friedman, 2009], to finding the state having maximum probability called Maximum-A-Posteriori (MAP) Inference [Koller and Friedman, 2009]. The problem in itself is computationally hard since it involves reasoning over an exponential number of states. Probabilistic Graphical Models (Bayesian and Markov Networks) [Koller and Friedman, 2009] were proposed for efficient inference by utilizing the independences present in the distribution. They specify independences present within the variables in the form of a graphical structure and became a model of choice for many applications in computer vision [Szeliski *et al.*, 2008b] and natural language processing [Blei *et al.*, 2003] during the late 1990s to early 2000s.

A popular approach for efficient probabilistic inference in PGMs is to exploit the symmetries present in the problem. The idea has been fairly popular in the last decade and is called *lifted inference* [Kimmig *et al.*, 2015]. There have been different ways to exploit symmetries for faster inference depending on whether the task is marginal or MAP inference. One of the approach is to identify the variables (states) which behave similar to each other for inference and then reason on a group of these variables together instead of individual ground variables (states). Since the inference algorithm is run on a smaller model, it is significantly faster. Then, the solution is mapped back to the original model. Symmetry exploiting variants have been proposed for both exact inference algorithms like variable elimination [Poole, 2003], weighted model counting [Gogate and Domingos, 2011], knowledge compilation [Van den Broeck *et al.*, 2011] as well as approximate inference algorithms like belief propagation [Singla and Domingos, 2008; Kersting *et al.*, 2009] and Markov Chain Monte Carlo methods (MCMC) [Van den Broeck and Niepert, 2015; Venugopal and Gogate, 2014b]. Past works have exploited symmetries for Marginal Inference [Singla and Domingos, 2008; Kersting *et al.*, 2009; Gogate and Domingos, 2011], MAP inference [Noessner *et al.*, 2013; Mladenov *et al.*, 2014; Mittal *et al.*, 2014; Sarkhel *et al.*, 2015] and more recently for the task of Marginal-MAP inference [Sharma *et al.*, 2018].

Approximate inference methods play a central role in scaling PGM-based models to real-world datasets. The primary argument being that the exact inference methods either do not scale well to the real-world problems or fail to give reasonable solutions within a desired time. We focus on applying symmetries in a popular approximate inference technique: MCMC methods [Koller and Friedman, 2009]. Past work [Niepert, 2012] has utilized states which have the same probability called, equi-probable states, for improving the mixing time in MCMC methods. They partition the whole state space in clusters of states where all states within a cluster have the same probability. This partition is compactly defined in terms of permutation groups where the states within a cluster are obtained from each other by permuting variables of states. Since these permutations are defined on variables, we call these permutations as *variable permutations* and the resulting symmetries as *variable symmetries*. Subsequently, they also utilize

these symmetries based on variable permutations in MCMC algorithms. We call the resulting MCMC algorithm which uses symmetries based on variable permutations as *Variable-MCMC*.

Variable symmetry, defined as permutation of variables, misses out on a large number of equi-probable states, which cannot be obtained from each other by permuting variables. These uncaptured state equivalences, if utilized by MCMC, have the potential to deliver significant improvements in mixing time. We ask the question: What is the largest set of state equivalences which can be useful for efficient inference? We answer this, by formally characterizing this set based on a novel idea of *weight-signature of a state*. The existing idea of symmetries based on variable permutations capture only a subset of these equivalences. We expand this set of state-equivalence by introducing multiple novel notions of symmetries. Also, we develop methods to efficiently compute and represent these state equivalences. Finally, we develop end-to-end MCMC inference methods to use each of these symmetries, showing significant empirical gains on benchmark domains. Next, we give a brief overview of each of these novel notions of symmetries:

- **Contextual Symmetries:** Firstly, past work studying symmetries in PGMs have defined symmetries unconditionally i.e., symmetries exist on the complete space without any condition. This misses out on large number of symmetries which arises only under a particular context (condition). A context is defined as a subset of variables and one of their assignments. Our work on *Contextual Symmetries* [Anand *et al.*, 2016b] addresses this lacunae by defining a novel notion of symmetry which arises under a particular context. The work draws inspiration from the work on contextual independence [Boutilier *et al.*, 1996] where certain variables are independent under a particular context. We formally define contextual symmetries in this work. Finally, we extend Variable-MCMC to use contextual symmetries by developing a novel *CON-MCMC algorithm*, showing significant end-to-end empirical gains over Variable-MCMC and Vanilla-MCMC (base MCMC procedure).

- **Variable-Value (VV) Symmetries:** Variable Symmetries and Contextual Symmetries capture equivalence between states by defining a permutation in variables. This misses a significant number of state equivalences where one state cannot be obtained by permuting variables of another state. For example, in a 2-bit state representation, it will never be able capture equivalence among states $\{1, 1\}$ and $\{0, 0\}$. We augment the set of state equivalences, by defining permutations over *variable-value* pairs instead of variables. The set of symmetries captured by variable-value permutation group are called *Variable-Value (VV)* symmetries [Anand *et al.*, 2017]. VV-Symmetry group not only subsumes symmetries captured by variable symmetry permutation group but also captures many more state equivalences. In addition, Variable-MCMC can be extended in a straight forward manner

to Variable-Value MCMC (VV-MCMC) which uses VV symmetries.

- **Non-EquiCardinal (NEC) Symmetries for Multi-Valued Domains:** Most of the previous work defined notions of state symmetries only on Boolean-valued domains. Some of these ideas can be extended to multi-valued domains with some modifications. However, none of the existing work is able to define the states equivalences among states, where the permutation needs to exchange values among variables of different domain cardinalities. Our work [Anand *et al.*, 2017] addresses this issue by defining a novel notion of *Non-Equicardinal (NEC) Symmetries*, which uses VV permutations to exchange values among variables having different domain cardinalities. We further illustrate the use of NEC-Symmetries in MCMC by developing a novel Metropolis Hastings [Koller and Friedman, 2009] extension.

- **Block-Value (BV) Symmetries:** VV Symmetries capture a significant number of state equivalences by defining permutations on VV pairs. But there are certain state equivalences which are missed by VV symmetries as well. Lastly, we ask the question: Is there a representation of state equivalences which is more powerful than VV? We answer this by defining a novel notion of Block-Value (BV) symmetries [Madan *et al.*, 2018] which captures many more state equivalences than caught by VV symmetries. BV symmetries define permutations in terms of Block-Value (BV) pairs where a block is a subset of variables. We design a novel Aggregate-Orbital-MCMC which is a first of kind MCMC method to utilize symmetries from multiple automorphism groups in Block-Value symmetries.

Space of State Symmetries: Having introduced multiple notions of state symmetries and shown their corresponding advantages, we characterize and understand how these state symmetries relate to each other by defining the complete space of state symmetries in PGMs. We show that Block-Value symmetries can capture any state equivalence (based on weight-signature) for arbitrary block size, however, the representation size for BV symmetries grows exponentially with increase in block size.

Our notions of proposed symmetries are exact symmetries and are algorithm oblivious though the effectiveness is shown on MCMC algorithms. It is an example of pipelined symmetry aware systems. State symmetries in these models are pre-computed, and, then, utilized in modified MCMC methods. The effectiveness of the system is attributed to the fact that symmetries are computed and represented very efficiently in terms of permutations groups using graph isomorphism solvers. These solvers incur negligible computational overhead although symmetries are computed at the preprocessing stage and are found uniformly on the whole state space. Next, we describe symmetry aware systems in the problem of sequential decision making.

1.3 Symmetries in Sequential Decision Making

Analogous to probabilistic inference, automated decision making is another important problem studied in AI Though the problem has many fundamental differences with probabilistic inference, it shares some of the key characteristics as well. Both the problems have an associated graphical structure and many classical algorithms for both the problems perform computation that depend only on local neighbours.

Automated decision making is not limited to a single decision (modeled as Bandits [Sutton and Barto, 1998]) but consists of making a series of decisions to optimize a long-term objective. This task of sequential decision making under uncertainty is commonly modeled as a Markov Decision Process (MDP) [Puterman, 1994]. The problem is well studied as an AI planning [Mausam and Kolobov, 2012] or a reinforcement learning problem [Sutton and Barto, 1998] depending on whether the model dynamics and rewards are known (AI planning) or unknown (reinforcement learning). We restrict our work to planning which assumes that the transition and reward model is given as an input to the problem.

Given the transition and reward model, the classical methods like value iteration, policy iteration, and linear programming [Bellman, 1957; Howard, 1960] have polynomial complexity in the size of state and action spaces. These methods scale exponentially with an increase in number of state and action features, and hence, suffer from the curse of dimensionality. One of the popular approaches for saving computation in such scenarios is to reduce the model size by merging symmetric states and actions together. Previous works ([Givan *et al.*, 2003; Ravindran, 2004; Li *et al.*, 2006]) compute these symmetries as a preprocessing step and then run the classical algorithms on the reduced model. Exact symmetries have been defined in terms of bisimulation relations [Givan *et al.*, 2003] or homomorphisms [Ravindran and Barto, 2004]. Both these definitions are restricted to merging of states or merging of actions within a state. Our work [Anand *et al.*, 2015a; Anand *et al.*, 2015b] argues that many of the state-action pairs may also have similar behaviour even though the corresponding parent states of these actions are not symmetric. This leads to much more reduction in model size as compared to the previous approaches. We call this new notion of symmetry as *"Abstraction of State-Action Pairs" (ASAP)*. We prove that ASAP symmetry subsumes previous notions of symmetries based on bisimulations and homomorphisms. The proposed notion of ASAP symmetries is general and oblivious to the algorithm. Further, the proposed definitions compute exact symmetries on states and state-action pairs which can be easily to multiple approximate notions.

Although ASAP symmetries can be used to reduce the model size, there has been a limited interest in model reduction approaches during the last decade. Instead, new approaches that sample only useful parts of state space have emerged. These are called Monte-Carlo Tree Search (MCTS) algorithms [Browne *et al.*, 2012; Kocsis and Szepesvári, 2006]. Their popu-

larity can be attributed to the fact that the state-of-the-art planner for International Probabilistic Planning Competition (IPPC)-2011 [Sanner and Yoon, 2011] and IPPC-2014 [Grzes *et al.*, 2014] was based on an MCTS algorithm. Also, MCTS was a key component in the Alpha-Go system [Silver *et al.*, 2016] which defeated the world champion Lee Sudol in the game of Go. MCTS algorithms sample a tree in the neighborhood of starting state, based on careful balancing of exploration-exploitation trade-off. It then takes a decision on the root node based only on the sampled tree. These algorithms need not look at the complete state space for taking a decision and have anytime behavior.

Our work, along with other works [Anand *et al.*, 2015a; Anand *et al.*, 2016b; Jiang *et al.*, 2014; Hostetler *et al.*, 2014; Hostetler *et al.*, 2015], observes that the idea of model reduction is orthogonal to these sampling-based approaches, and applying symmetries in MCTS could further augment the gains of MCTS-based algorithms. Our work ASAP-UCT [Anand *et al.*, 2015a] is one of the earliest works which illustrates the use of symmetries in an MCTS framework. Particularly, we incorporate ASAP symmetries in UCT algorithm [Kocsis and Szepesvári, 2006], a popular variant of MCTS. We call the new algorithm: ASAP-UCT. ASAP-UCT is a batch algorithm which iterates between two phases: a symmetry computation phase and a tree building phase. Symmetry computation phase computes symmetry only over the currently sampled tree while the tree building phase utilizes the computed symmetries to further guide the tree building process. The key idea is that a single sample at the leaf of the tree will be able to update values of all symmetric states along the path to the root. Our experiments illustrate up to 25% improvements as compared to the previous notions of symmetries as well as vanilla UCT on a number of planning benchmark domains.

We further observe that ASAP-UCT does not fully realize the benefit of symmetries in MCTS. The resisting issue for ASAP-UCT is batch computation of symmetries. Although symmetries are computed as the tree is built, this symmetry computation and updating is only periodical. Also, symmetries are computed only on a sampled tree, hence, they may be erroneous and approximate. Erroneous symmetries are used till the next symmetry computation phase and there is limited incremental symmetry improvement. Additionally, computationally expensive symmetry finding routine is run uniformly over the whole tree in each phase while UCT focuses only on promising parts of the tree by exploration-exploitation trade-off. Based on these insights and considering our design characteristics of symmetry aware systems, we develop OGA-UCT: On-the-Go Abstractions in UCT [Anand *et al.*, 2016b] which replaces this batch symmetry computation by finding symmetries on-the-fly as the tree is built. This novel algorithm computes symmetries on-the-fly, has adaptive symmetry computation and incrementally improves symmetries. Hence, it fulfills all the desirable characteristics of symmetry aware algorithms. We experimentally observe that OGA-UCT is robust across a variety of planning domains and obtains up to 28% quality improvements.

1.4 Structured Output Prediction in Computer Vision

Lastly, we directly explore the use of symmetries in state-of-the-art algorithms for a given application. Computer Vision, Natural Language Processing and Speech Processing etc. have been the primary hunting grounds for machine learning researchers. We focus on the task of computer vision in this thesis since it has some natural symmetries and invariances in the grid structure of image. Also, most of the computer vision tasks are computationally expensive, making a natural candidate for leveraging the benefit of symmetries in saving computation. We apply symmetries in PGM-based models in computer vision. PGMs have been widely used for modelling pixel labelling tasks in computer vision like semantic image segmentation [Kohli *et al.*, 2013], stereovision [Mozerov and van de Weijer, 2015] etc. The key challenge in using PGMs for a grid structured graphical model is that inference on such a graph is highly computationally expensive and intractable [Koller and Friedman, 2009] due to high tree width of these graphs. Many works [Szeliski *et al.*, 2008b] in the past have resorted to approximate inference techniques in such situations. In this work, we study the application of symmetries for improving the efficiency further.

The problem of pixel labelling is described as: given an image, label each pixel in the image from the given set of labels. Since the output variables have a spatial structure, this task is also known as *structured output prediction*. The higher order potentials of the grid structured PGM capture the neighborhood dependencies and the task reduces to finding the labelling which has maximum probability or minimum energy, popularly called as *Maximum A Posteriori (MAP) Inference*.

We argue that this task of MAP inference on a grid structured graph has immense potential for exploiting symmetries. Firstly, the structure of PGM is a grid which has symmetries of pixels along the diagonal. Secondly, the higher order potentials imposing the smoothness constraints on the image are often constant throughout the image. The key challenge lies in handling unary potentials on the pixels. Each of these unary potentials is a function of image intensities of that pixel. Since the pixel intensities vary for each pixel (for real-world images), these potentials have distinct values for each pixel variable. This prohibits us to obtain a smaller reduced graphical model by exact merging of pixels with each other. In response, we propose a novel heuristic for approximately merging unary potentials.

The proposed *top-K label heuristic* merges all those pixels whose top-k ordered labels in unary potential tables match with each other. The obtained grouping is further split by applying some iterations of the popular symmetry computation algorithm in PGMs, Color Passing Algorithm [Kersting *et al.*, 2009]. Intuitively, these iterations of color passing ensure that merged pixels have a similar neighbourhood in space. We call each such grouping as a *lifted pixel*. These groupings impose a constraint on inference procedure such that all the ground pixels of

a grouping are forced to take the same label. This constraint reduces the size of model to the number of lifted pixels, thereby, resulting in significant computational savings.

Unfortunately, it may also impact the solution quality. We alleviate this problem by varying the degree of approximation of symmetries. We obtain a series of coarse-to-fine lifted pixels, which captures quality vs time trade-off. By switching from coarser-to-finer (C2F) groupings, we ensure that the solution quality is not impacted, and at the same time, we obtain significant computational savings. This approach gives us a template for lifted MAP inference. We apply this C2F template to near state-of-the-art PGM based models in Stereovision and Image Segmentation. Our experiments illustrate up to 60% quality gains in time constrained settings with an order of magnitude improvement in time for obtaining the best solution.

This work is an example of use of symmetries directly in near state-of-the-art algorithms that may have many domain specific insights and specifications. Since exact symmetries may not be found in real world, we resort to approximate symmetries based on top-k label heuristic. The symmetries defined are algorithm oblivious though these use many insights from computer vision domain. Further, our C2F template is a joint symmetry aware system. It begins from an initial set of pixel symmetries but also incrementally improve symmetries by refining the pixel partition till a ground network is obtained.

1.5 Contributions and Outline

This thesis illustrates that symmetries and invariances have great potential in saving computations in multiple AI scenarios. Overall, it makes the following contributions:

1. We discuss the concept of symmetry aware inference which can improve the efficiency and performance of contemporary AI and ML algorithms in Chapter 1. We enumerate the desirable characteristics of modern symmetry aware inference algorithms. Further, we study these characteristics in the context of symmetry aware algorithms for sequential decision making and probabilistic inference.

2. In Part-II, we begin by discussing the framework of state symmetries in PGMs in Chapter 2. We introduce multiple novel notions of state symmetries in PGMs for probabilistic inference: Contextual Symmetries in Chapter 3, Variable-Value Symmetries and Non-Equicardinal Symmetries in Chapter 4, and Block-Value Symmetries in Chapter 5. We are also the first to use these symmetries in multi-valued graphical models. We further illustrate the effectiveness of these symmetries in Markov Chain Monte Carlo (MCMC) methods by developing novel variants that exploit symmetries, and show significant empirical gains in domains where these symmetries are present. We develop hierarchy of state symmetries in Chapter 6.

3. In sequential decision making, we introduce a novel notion of ASAP abstractions which merges the state-action pairs in addition to states in Chapter 7. We also illustrate how to incorporate symmetries in Monte Carlo Tree Search framework via two methods: ASAP-UCT in Chapter 7 and OGA-UCT in Chapter 8, and obtain roughly 25% improvements in quality.

4. Lastly, we develop techniques for exploiting symmetries in MAP inference for computer vision algorithms in Chapter 9. We developed a coarse-to-fine lifted MAP template for exploiting symmetries using a novel k-label heuristic. We apply this template to near state-of-the-art algorithms in Stereovision and Image Segmentation and obtain up to 60% quality gains in time constrained settings.

This thesis is also arranged in three parts and discusses symmetry aware algorithms for state symmetries in PGMs, sequential decision making and structured output prediction in computer vision. Each part begins by building relevant background and related work to that part. Part II discusses the symmetry aware inference algorithms for the problem of efficient probabilistic inference. Part III applies the idea of exploiting symmetries to the sequential decision making under uncertainty. Part IV focuses on using symmetries directly in computer vision applications, specifically, for Stereovision and Image Segmentation. Lastly, we conclude our work and discuss key directions for future work in epilogue.

Work done in Chapter 3 and Chapter 7 was done jointly with Aditya Grover. Work done in Chapter 4 and Chapter 8 was done jointly with Ritesh Noothigattu. Work done in Chapter 9 was done jointly with Haroun Habeeb. Work done in Chapter 5 was done jointly with Gagan Madan. In each case, the part done by the collaborators appeared in their respective bachelor's or master's theses.

Part II

State Symmetries in Probabilistic Graphical Models

Chapter 2

Foundations and Setup

"The most important questions of life are indeed, for the most part, really only problems of probability."

Pierre-Simon Laplace

Probability theory has been the key mathematical tool to model uncertainties present in the universe. In fact, many real world events are described by defining probability distributions over their occurrence. Reasoning over these probability distributions is one of the central tasks in ML and AI. Probabilistic Graphical Models (PGMs) [Koller and Friedman, 2009] like Bayesian and Markov networks were proposed for reasoning by utilizing model independencies present in the distribution. The probability distributions are specified by a graphical structure and the independencies are captured by special connectivity notions within these graphs. One of the popular forms of reasoning commonly performed in PGMs is *Marginal Inference*. The task of marginal inference is to determine the probability of a particular variable (or a small subset of variables) given some observations (called *evidence*). As the number of variables increase, reasoning over these distributions become more complex. An important area of research is to perform efficient reasoning or inference over these distributions. Many classical algorithms [Koller and Friedman, 2009] like variable elimination, junction tree, message passing etc. have been proposed for inference on PGMs. The complexity of these algorithms is exponential in a property of the graph, called the tree width [Koller and Friedman, 2009]. This exponential complexity is a major concern and a bottleneck when applying PGMs to problems having high tree width.

A popular approach for efficient inference in PGMs, called *lifted inference* is to exploit symmetries present in the underlying domain. It is especially useful for statistical relational learning models such as Markov logic networks [Richardson and Domingos, 2006], which exhibit repeated sub-structures – many objects are indistinguishable from each other and their associated relations have identical probability distributions. *Lifted inference* algorithms (see [Kimmig *et*

al., 2015] for a survey) exploit this phenomenon by grouping symmetric states (variables) into meta-states (meta-variables) and performing inference in a reduced (lifted) graphical model.

Early approaches to lifted inference devised first order extensions of propositional inference algorithms. These include approaches for lifting exact inference algorithms such as variable elimination [Poole, 2003; Braz *et al.*, 2005], weighted model counting [Gogate and Domingos, 2011], knowledge compilation [Van den Broeck *et al.*, 2011], as well as lifting approximate algorithms such as belief propagation [Singla and Domingos, 2008; Kersting *et al.*, 2009; Singla *et al.*, 2014], Gibbs sampling [Venugopal and Gogate, 2012] and importance sampling [Gogate *et al.*, 2012]. In all of these, the lifting technique is tied to the specific algorithm being considered. More recently, another line of work [Jha *et al.*, 2010; Bui *et al.*, 2013; Niepert and Van den Broeck, 2014; Sarkhel *et al.*, 2014; Kopp *et al.*, 2015] has started looking at the notion of lifting independent of the inference technique. In several cases, these symmetries are compactly represented using permutation groups. The computed symmetries have been used downstream for lifting existing algorithms such as variational inference [Bui *et al.*, 2013], (integer) linear programming [Noessner *et al.*, 2013; Mladenov *et al.*, 2014], and Markov-chain Monte Carlo (MCMC) [Niepert, 2012; Van den Broeck and Niepert, 2015]. We focus on symmetries defined using permutation groups and their use in MCMC algorithms.

Specifically, Niepert [2012] illustrated that states having the same probability (*equi-probable states*) can be utilized to improve the mixing time of MCMC methods. They computed and represented these states by defining permutations over variables. For example, consider the graphical model defined in figure 2.1, states $\{X_1 = 0, X_2 = 1\}$ and $\{X_1 = 1, X_2 = 0\}$ have same unnormalized probability given by the value b and this equivalence can be captured by the permutation which permutes the values of X_1 and X_2. These permutations called *variable permutations*, form a group and can represent exponential number of state equivalences in a polynomial representation. Although the idea is powerful, it cannot find equivalence between many states which have the same probability. For example, consider a simple graphical model over 4 variables in figure 2.2. It can be easily seen that many states have the same probability in this simple graphical model, for example, states $s_1 = \{X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 0\}$ have unnormalized probability of $a \times b$ which is exactly equal to unnormalized probability of state $s_2 = \{X_1 = 0, X_2 = 1, X_3 = 1, X_4 = 1\}$. In fact, every state has an unnormalized probability given as product of two terms $v_1 \times v_2$ and one can obtain an equi-probable state by reversing the order in which v_1 and v_2 are picked from two potential table. This discussion is true for all graphical models having similar form irrespective of numeric values $\{a, b, c, d\}$. More importantly, none of these equi-probable states can be represented by defining a permutation over variables and hence, cannot be computed and used by previous work.

In this work, we focus on capturing all these equi-probable states in a single equivalence class for each probability value. Particularly, we focus on those type of equi-probable states

which have the same probability irrespective of numerical values of weights or potential table. These type of equi-probable states arise due to sharing of parameters in the graphical model and similarity in structure and hence, we refer to them as *state symmetries* of graphical models. For example, for the graphical model in figure 2.2, parameter-value is shared between features $(\neg X_1 \wedge \neg X_2)$ and $(X_3 \wedge X_4)$ represented by value a. The similar parameter sharing can be seen for other values. To represent these equi-probable states, we begin by defining a novel representation of graphical model, called *weight-tying representation*. Weight-tying representation separates the numeric parameter values and parameter-sharing in the graphical model at the representation. We formalize this idea by defining a novel notion of *weight-signature* of a state which is a symbolic equivalent of the probability of that state. Under this framework, we define the partition of state space based on their equivalence in weight-signature. All states having the same weight-signature are assigned to the same equivalence class.

This chapter defines the framework of what all state equivalence we intend to represent in subsequent chapters. It begins by defining the notation and background work on symmetries in graphical models using permutation groups. It, then, introduces weight-tying representation of a graphical model and introduce the symbolic equivalent of probability of a state, called the weight-signature. The subsequent chapters define novel notion of symmetries to capture significantly more states (having same weight-signature) as equivalent than done in previous work.

Particularly, we defined a novel notion of symmetry, called *contextual symmetry* that capture state equivalences arising only under a particular context. Contextual symmetries also result in weight-signature preserving state partition and subsume the state equivalences captured by variable symmetries. Subsequently, we defined permutations on the set of variable-value pairs and called the resulting symmetries as *Variable-Value (VV) symmetries*. VV symmetries can represent significantly more state equivalences than identified by variable permutations. Though, VV symmetries can represent a significant number of state equivalences, it still misses out some states that have same weight-signature. This was addressed in our work of *Block-Value (BV) symmetries* where permutations are defined on block-value pairs. A Block is a subset of variables. BV permutations can represent all the possible state-equivalences based on weight-signature at the cost of computational effort. We also study hierarchy of state symmetries and relationships between them in Chapter 6.

X_1	X_2	Φ
0	0	a
0	1	b
1	0	b
1	1	c

Figure 2.1: A 2-variable graphical model with its potential table. States $\{0,1\}$ and $\{1,0\}$ have same probability and can be captured by a simple permutation $X_1 \leftrightarrow X_2$.

X_1	X_2	Φ
0	0	a
0	1	b
1	0	c
1	1	d

X_3	X_4	Φ
0	0	b
0	1	d
1	0	c
1	1	a

Figure 2.2: A 4-variable graphical model with its potential tables. Pair of states $\{0,0,0,0\}$ and $\{0,1,1,1\}$ have the same probability but variable permutations are not able to capture these as symmetric states

2.1 Related Work

Lifted inference algorithms typically cluster symmetric states (variables) together and use these clusters to reduce computation, for example, by avoiding repeated computation for all members of a cluster via a single representative. There are multiple ways to classify the literature on lifted inference in PGMs as described in Chapter 1. This could include approximate or exact symmetries, state or variable symmetries, algorithm oblivious or algorithm dependent symmetries and symmetries in marginal or MAP inference. Our work in this part deals with exact state symmetries which are oblivious to the algorithm. These symmetries are, then, utilized in MCMC algorithms which are typically used for marginal inference.

Closely related to our work, the earliest notion of state symmetries was described for clausal theories using permutation groups by Crawford *et al.* [1996]. Bui *et al.* [2013] used this idea by defining automorphic group of an exponential family or a graphical model. Instead of working at the algorithm level, they studied these generic notion of symmetries in their variational formulations. Parallely, Niepert *et al.* [2012] used a similar idea of state symmetry as permutation groups to improve the mixing time of MCMC. They defined permutations over the set of variables. This idea was further extended to incorporate approximate state symmetries in MCMC by a Metropolis Hastings extension by Broeck and Niepert [2015].

Our work falls in this category of expressing state equivalences (of equi-probable states) as permutation groups. Specifically, we start from the framework of state symmetries proposed by Niepert [2012] and extend it further to understand the complete space of equi-probable states in PGMs. Though the notion of symmetries as permutation of variables is a powerful idea as it captures exponential state equivalences in polynomial representation, it misses out on many equi-probable states present in the problem. These missed states if potentially utilized could further speedup the downstream inference algorithms.

We ask the question: What is the largest set of equi-probable states which can be represented independently of numerical values? In response to this, we develop the idea of weight-signature of a state and define weight-signature preserving state partition. It characterizes the largest set of states that have same probability based on how weights are shared and the structure of graphical model. Further, we propose multiple novel notions of state symmetries which capture more and more state equivalences than done by Niepert [2012]. Further, we develop a complete hierarchy of state symmetries and define the notion of block-value symmetries which capture all the state equivalences based on weight-signature.

One of the closely related concepts to our work is the idea of exchangeability and partial exchangeability. The works of [Niepert and Van den Broeck, 2014; Niepert and Domingos, 2014] have focused on developing the theory of partial exchangeability in graphical models and connected the ideas of exchangeability with tractable inference in graphical models. Our

work, on the other hand, proposes novel notions of symmetry in graphical model related to these ideas and, also, proposes algorithms to compute and use these symmetries in MCMC algorithms.

In addition, there is related work which has focused on symmetric clique potentials [Gupta *et al.*, 2010] where symmetries are captured within a single clique potential. However, these works are limited to the cases where the counts of 0s and 1s match in symmetric states. These symmetries have been exploited in computer vision for higher order potentials [Tarlow *et al.*, 2010], in Natural Language Processing (NLP) for models used for collective inference [Gupta *et al.*, 2010] and lifted inference, in general [Milch *et al.*, 2008].

2.2 Preliminaries and Background

Let $\mathcal{X} = \{X_1, X_2, \cdots, X_n\}$ denote a set of random variables with each variable X_i having a domain D_i. For Boolean valued domains, $D_i = \{0, 1\}$. We further call a variable with its assignment as a literal. We will denote the probability distribution associated with the set $\mathcal{X}$ as $\mathcal{P}$. A state $s = \{(X_i, v_i)\}_{i=1}^{n}$ is a complete assignment to variables in $\mathcal{X}$, with values $v_i \in D_i$. We will use the symbol $\mathcal{S}$ to denote the entire state space. We denote $s(X_i)$ to the value of variable X_i in state s.

A typical graphical model $\mathcal{G}$ over the set of variables $\mathcal{X}$ is defined as the set of pairs $\{f_j, w_j\}_{j=1}^{m}$ where f_j is a feature function and w_j is the corresponding weight [Koller and Friedman, 2009]. In general, a feature can be any real valued function of a state s. For simplicity, we assume binary valued features i.e $f(s) \in \{0, 1\}$ and unless specified, a feature is clausal[1] where a clause is a disjunction of finite set of literals. For multi-valued graphical models, a feature is a disjunction of variables equated to their values i.e $\{X_i = v_j\}$ where $v_j \in D_i$. We denote the subset of variables participating in a feature f_j as $Vars(f_j)$. The probability of a state s under $\mathcal{G}$ is defined as follows:

$$P(s) = \frac{1}{Z} e^{\sum_{j=1}^{m} w_j f_j(s)} \tag{2.1}$$

where

$$Z = \sum_{s \in S} e^{\sum_{j=1}^{m} w_j * f_j(s)}$$

is the normalization constant or the partition function. The calculation of Z involves sum over exponential number of states and hence, calculation of exact probability is computationally

[1] Each model can be pre-converted to an equivalent model in which all features are clausal.

expensive. A variable $X_i \in Vars(f_j)$ is said to be consistent in s with f_j if $(X_i = s(X_i)) \in \mathrm{f}_j$.

Graphical models can also be equivalently represented as a set of functions called potential functions or potential tables Φ. Each potential function is defined over a sub-set of variables and has domain as all possible assignments of that sub-set of variables which is mapped to a non-negative real number. Any given potential function can be equivalently converted to a set of features with each row denoting a feature over conjunction of variables along with their values in that row. The weight of feature is given by logarithm of potential function value in that row. For example, in Figure 2.3 , the potential function can be represented in feature representation by the features $\{f_1 = A \wedge \neg B, f_2 = \neg A \wedge B\}$ where both f_1 and f_2 will have weight $\log(w)$. We will interchangeably use feature representation and potential function representation for ease of explanation in this thesis.

2.2.1 Symmetries of a Graphical Model

Symmetries of any structured object has traditionally been defined in terms of permutations. A permutation θ of $\mathcal{X}$ is a bijection of the set $\mathcal{X}$ onto itself.$\theta(X_i)$ denotes the variable that X_i is mapped to by permutation θ. We will refer to θ as a *variable-permutation*.

Definition 2.2.1. *The application of a permutation θ on state s, denoted by $\theta(X_i)$ results in another state $s' = \theta(s)$, s.t. if X_i takes the value v_i in state s, then, $\theta(X_i)$ takes the value v_i in s'*

For example, consider the variable-permutation θ_1 given by $\theta_1(A) = B$ and $\theta_1(B) = A$, then, for $s = (1, 0)$, $\theta_1(s)$ results in a new state $(0, 1)$.

Definition 2.2.2. *Similar to $\theta(s)$, we define application of θ on a feature f_j as $f'_j = \theta(f_j)$ which is given by replacing each variable X_i in clausal form of f_j by $\theta(X_i)$.*

Further, we define application of θ on a graphical model $\mathcal{G} = \{f_j, w_j\}_{j=1}^m$ as $\theta(\mathcal{G})$, a graphical model constructed by applying θ on each f_j i.e. $\theta(\mathcal{G}) = \{\theta(f_j), w_j\}_{j=1}^m$. A set of permutations Θ is called a permutation group if it is closed under composition, contains the identity permutation, and each $\theta \in \Theta$ has its inverse in the set.

Drawing parallels from graph automorphism where the permutation over vertices maps the graph back to itself, we define the notion of automorphism (referred to as symmetry, henceforth) of a graphical model as follows [Niepert, 2012].

Definition 2.2.3. *Given a graphical model $\mathcal{G}$, a permutation θ of $\mathcal{X}$ is a* **variable symmetry** *of $\mathcal{G}$ if application of θ on $\mathcal{G}$ results back in $\mathcal{G}$ itself, i.e., it returns the same set $\{f_j, w_j\}_{j=1}^m$ as in $\mathcal{G}$. We also call such permutations as variable permutations.*

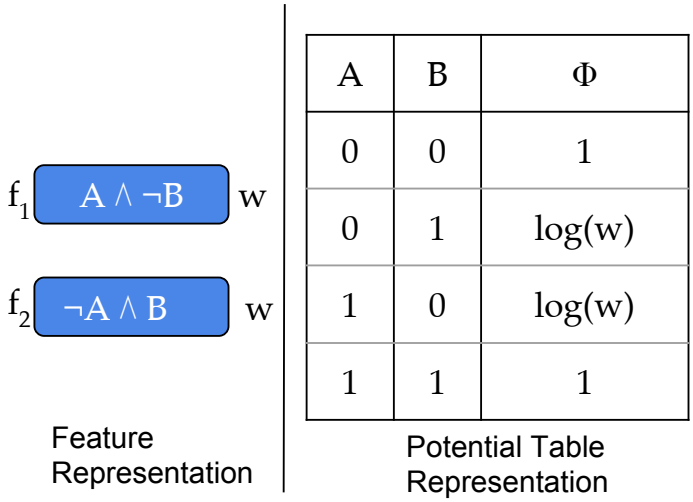


Figure 2.3: Graphical Model $\mathcal{G}_1$ shown in form of potential table and weighted features

For example, θ_1 defined above is a variable symmetry of $\mathcal{G}_1$ as it results in the same set of formula with equal weight.

Theorem 2.2.1. *Given a graphical model $\mathcal{G}$, the set of all variable symmetries form a group Θ. This group is called as* **automorphism group** *of the graphical model $\mathcal{G}$.*

Another important concept is the notion of an orbit of a state resulting from the application of a permutation group.

Definition 2.2.4. *The* **orbit** *(Γ) of a state s under the permutation group Θ, denoted by $\Gamma_\Theta(s)$, is the set of states resulting from application of permutations $\theta \in \Theta$ on s, i.e., $\Gamma_\Theta(s) = \{s' | \exists \theta \in \Theta, \theta(s) = s'\}$.*

Note that orbits form an equivalence partition of the entire state space. In this work, we are interested in orbits obtained by application of an automorphism group, because all states in such an orbit have the same joint probability. Let $\mathcal{P}$ denote the joint probability of a state s under $\mathcal{G}$. The following theorem captures the relation between variable automorphism group and probabilities.

Theorem 2.2.2. *[Niepert, 2012] Given a graphical model $\mathcal{G}$, let Θ be a variable automorphism group of $\mathcal{G}$. Then for all states $s \in \mathcal{S}$ and permutations $\theta \in \Theta$, the application of θ on s preserves the probability i.e. $\forall \theta \in \Theta$, $s \in \mathcal{S}$, $\mathcal{P}(s) = \mathcal{P}(\theta(s))$.*

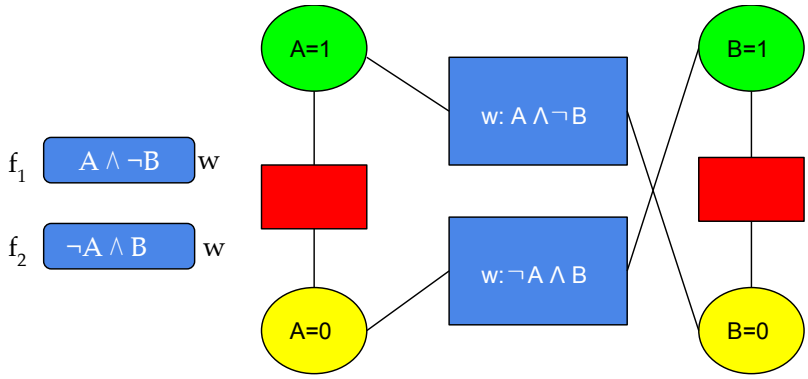


Figure 2.4: Colored graph for Graphical Model $\mathcal{G}_1$ defined in Figure 2.3

2.2.2 Graph Isomorphism for Computing Symmetries

The procedure for computing an automorphism group [Niepert, 2012] first constructs a colored graph $G_V = T(\mathcal{G})$ from the graphical model $\mathcal{G}$. In this graph there are $|D_i|$ nodes for each variable, one for each of its value, and a node for each feature in $\mathcal{G}$. We add an additional node for each variable which is connected to all its value. This represents a feature having ∞ weight representing mutual exclusivity between all values of a variable. There are edges between a literal node and a feature if that literal appears in that feature in the graphical model. For example, the graphical model $\mathcal{G}_1$ along with its associated colored graph is shown in Figure 2.4.

For Boolean graphical model, each node is assigned a color such that all 1 value nodes get the same color, all 0 value nodes get the same color (but different from 1 node color), and all feature nodes get a unique color based on their weight (including the ∞ weight feature). That is, two feature nodes have the same color if their weights in $\mathcal{G}$ are the same. The similar ideas can be extended to non-boolean graphical models.

A graph isomorphism solver (e.g., Saucy [Darga *et al.*, 2008]) over $G_V(\mathcal{G})$ outputs the automorphism group of this graph through a set of permutations. These permutations can be easily converted to variable permutations of $\mathcal{G}$. Any output permutation always maps a variable's 0 and 1 nodes to another variable's 0 and 1 nodes, respectively because 0 value nodes are given the same color and 1 value nodes are assigned the same color and these two nodes are connected by a feature having infinity weight (corresponding color). These permutations collectively represent an automorphism group of $\mathcal{G}$.

Theorem 2.2.3. *[Niepert, 2012] The automorphism group of colored graph $G_V(\mathcal{G})$ constructed from the graphical model $\mathcal{G}$ gives the automorphism group of graphical model $\mathcal{G}$*

2.2.3 Orbital Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods [Koller and Friedman, 2009] are one of most popular methods for approximate inference. In these methods, a Markov chain $\mathcal{M}$ is set up over the state space and samples are generated. Running the chain for a sufficiently long time, starts generating samples from the true distribution. Gibbs sampling is one of the simplest MCMC methods.

Orbital MCMC [Niepert, 2012] adapts MCMC to exploit the given variable symmetries of the graphical model $\mathcal{G}$.

Given a Markov Chain $\mathcal{M}$, a variable automorphic group Θ and starting from state s_t, Orbital MCMC generates the next sample s_{t+1} in two steps:

- It first generates an intermediate state s'_t by sampling from the transition distribution of $\mathcal{M}$ starting from s_t
- It then samples state s_{t+1} uniformly from $\Gamma_\Theta(s'_t)$, the orbit of s'_t

The Orbital MCMC chain so constructed converges to the same stationary distribution as original chain $\mathcal{M}$ and is proven to mix faster, because of the orbital moves.

Theorem 2.2.4. *[Niepert, 2012] Let the Markov Chain $\mathcal{M}$ has the transition probability function $\mathcal{T}$. Further, let $\mathcal{M}$ be regular and has unique stationary distribution π. The orbital markov chain $\mathcal{M}'$ constructed from variable automorphism group Θ is also regular and also converges to the unique stationary distribution π.*

2.3 Understanding probability preserving partitions

The potential gain in Orbital MCMC is related to the number of equivalences among equi-probable states. Larger number of equivalences leads to more gain in MCMC. This motivates us to find the largest set of state equivalences of equi-probable states. We understand this notion by defining a partition of the state-space of a graphical model based on probability and, then, defining the granularity of these partitions.

Definition 2.3.1. *Given a graphical model $\mathcal{G}$, we define $\Delta = \{\Delta_1, \Delta_2 \cdots \Delta_k\}$ as a partition of state-space $\mathcal{S}$ into k equivalence classes s.t. $\forall j \in [1, k]; \Delta_j \subseteq \mathcal{S}$, $\Delta_i \cap \Delta_j = \emptyset$ if $i \neq j$ and $\bigcup_{j=1}^{k} \Delta_j = \mathcal{S}$. We call equivalence classes $\Delta_1, \Delta_2 \cdots \Delta_k$ as the partition-elements of partition Δ.*

The state space could be partitioned in multiple ways. For example, the simplest partition is to have a separate equivalence class for each state. On the other extreme, accumulating all states

in a single class is another partition. We formally capture this spectrum of different partitions by defining the notion of coarser and finer partitions.

Definition 2.3.2. *Given a graphical model $\mathcal{G}$, a state space partition $\Delta' = \{\Delta'_1, \Delta'_2, \cdots, \Delta'_{k'}\}$ is said to be* **coarser** *than another state space partition $\Delta = \{\Delta_1, \Delta_2, \cdots, \Delta_k\}$ if $\forall \Delta_i \in \Delta, \exists j$ such that $\Delta_i \subseteq \Delta'_j$. Equivalently, Δ is then said to be a* **finer** *partition than Δ'.*

Further, for the purpose of our work, we are interested in a special types of partition, called probability preserving partition.

Definition 2.3.3. *Given a graphical model $\mathcal{G}$, a partition Δ is* **probability preserving partition** *if all states belonging to the same equivalence class have equal probability i.e. $\forall j \in [1, k] \forall (s, s') \in \Delta j$, we have $\mathcal{P}(s) = \mathcal{P}(s')$*

As described earlier, variable automorphism group creates a partition of state space among equivalence classes which is probability preserving. But variable automorphism group misses out on many states which have same probability because these states cannot be expressed as permutation of variables.

Ideally, we would like to compute the **coarsest probability preserving state partition** for a given PGM $\mathcal{G}$, i.e. get the probability preserving partition with the minimum number of orbits, which, among other applications, would eventually lead to faster mixing in MCMC style inference algorithms. But finding the coarsest probability preserving partition is computationally hard problem. Infact, we can show that even to find the number of orbits in coarsest partition is a NP-Hard problem. We begin by giving a decision version of a subproblem, which we would use to show this is NP-Hard.

Definition 2.3.4. PGM-Num_Orbit: *Given a graphical model $\mathcal{G}$, is the number of orbits in the* **coarsest probability preserving** *partitioning of the state space $\leq k$?*

We show that the problem **PGM-Num_Orbit** is NP-Hard by reducing **3-SAT** to **PGM-Num_Orbit**. For the ease of readability, the proof is given in the appendix.

Theorem 2.3.1. PGM-Num_Orbit *is NP-Hard.*

In fact, finding the coarsest probability preserving partition is not the focus of our work. We intend to find equivalence among those states which have same probability based on how parameters are shared and the structure of model. This differs from the equivalence among states based on numeric value of probability. For example, consider the 2-variable toy graphical model $\mathcal{G}_2$ shown in Figure 2.5 with 3 features and their corresponding weights. The unnormalized probability value for all the states is also shown in the figure on the right. While states $(0, 1)$ and $(1, 1)$ have the same unnormalized probability, this equivalence in probability is dependent

on numeric values of parameters in the model ($0.2 + 0.2 = 0.4$). Changing the numerical value of parameters breaks this state equivalence. Instead of focusing on probability equivalence, we focus on symbolic equivalence of probability which is independent of parameter values but only dependent on how parameters are shared.

Interestingly, only symbolic equivalences can only be captured by graph autormorphism based methods. Graph automorphism methods assign a unique color to each weight value while computing symmetries of a graph. Hence, it computes only those equivalences which are independent of numerical value of parameters but only capture which numerical values are same. The next section formally characterize these equivalences by defining a weight-tying representation of graphical model which separates the parameter values from the information, how parameters are shared. Then, it defines the weight-signature of state which captures symbolic equivalence of probability and then, it defines partition of state-space based on the idea of weight-signature.

2.4 Weight-signature preserving state partitions

In this section, we begin by defining a new representation of a graphical model which explicitly separate the weight values and weight tying. We begin by defining a partition of the feature set, and then defining *weight tying representation* of a graphical model.

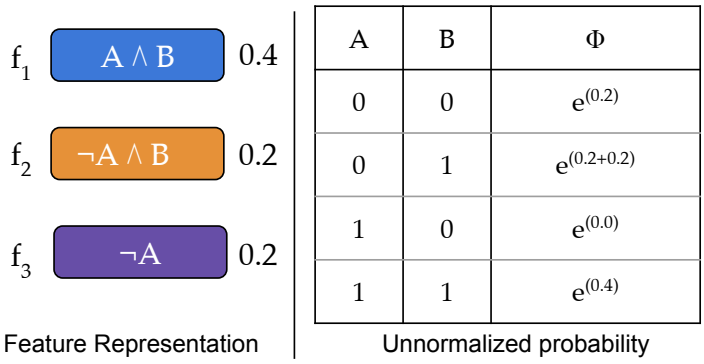


Figure 2.5: States (0,0) and (0,1) have the same probability but automorphism based algorithms would not capture this state equivalence

2.4.1 Weight Tying Representation

A typical graphical model $\mathcal{G} = \{f_j, w_j\}_{j=1}^m$ is represented as a set of weighted features. Though, in general, each of these features can be associated with a different real valued weight. However, many real-world scenarios have a small number of weights where these weights are shared across multiple features in the graphical model. For example, the weight between neighbouring pixels in an image is shared through out the image in Potts model [Koller and Friedman, 2009]. We capture this sharing of weights by defining a partition of features based on weight values.

Definition 2.4.1. *Given a graphical model $\mathcal{G} = \{f_j, w_j\}_{j=1}^m$, let $F = \{f_j\}_{j=1}^m$ denote the set of all features,* ***a feature-partition****, $\Delta^f = \{\Delta_1^f, \Delta_2^f \cdots \Delta_k^f\}$ is defined analogous to state partition s.t $\Delta_j^f \subseteq F$, $\bigcup_{j=1}^k \Delta_j^f = F$ and $\Delta_i^f \cap \Delta_j^f = \emptyset$ if $i \neq j$. .*

Further, we can define a weight-preserving feature partition as follows:

Definition 2.4.2. *Given a graphical model $\mathcal{G} = \{f_j, w_j\}_{j=1}^m$, let Δ^f be a feature partition. Δ^f is* **weight-preserving feature partition** *if weights of all features belonging to the same partition-element are equal i.e $\forall f_a, f_b \in F$, $\forall \Delta_j^f \in \Delta^f$, if $f_a, f_b \in \Delta_j^f$, then $w_a = w_b$.*

We are interested in coarsest weight-preserving feature partition. For example, the weight-preserving feature partition for $\mathcal{G}_2$ defined in Figure 2.5 is given by $\{(f_1), (f_2, f_3)\}$ where f_2 and f_3 belong to one cluster while f_1 belongs to a separate cluster. This allows us to define a new representation of a graphical model based on weight tying in features.

Definition 2.4.3. *Given a graphical model $\mathcal{G}$, its* **weight-tying representation,** *$\hat{\mathcal{G}}$ is given by a set of pairs $\{w_j, \Delta_j^f\}_{j=1}^k$ where Δ_j^f is a sub-set of features all having the same weight w_j and $\{\Delta_j^f\}_{j=1}^k$ forms the coarsest weight-preserving partition of feature-set F.*

This weight-tying representation further allows us to define a family of graphical models, whose partition of feature set is same irrespective of weight values.

Definition 2.4.4. *A* **family of graphical models having identical weight-tying***, $\mathcal{F}$ consists of all the graphical models whose feature-set F is same and they have identical coarsest weight-preserving feature partition Δ^f. This means that the weight-tying representation of all the members in the family differ only by the weight-values (w_j).*

It should be noted that while defining the family, we have not taken into consideration the actual weight values but have only considered the tying of features based on weights. Further, all automorphism based algorithms (including those of variable symmetries) while computing symmetries only consider weight tying and not the weight values. Therefore, all automorphism based algorithms will compute exactly same set of symmetries for all graphical models in the family.

Next, we define the equivalence of states in a symbolic notion of unnormalized probability which we call as em weight signature of state.

2.4.2 Weight-Signature of a State

Intuitively, a weight-signature of a state is a symbolic notion of probability of a state in the absence of weight-values. Since, weight-signature of a state only uses weight-tying given by feature partition, the weight-signature of a state will be same for all the members in the family of graphical models having identical weight-tying.

Definition 2.4.5. *Given a family of graphical models $\mathcal{F}$ having identical weight-tying, given by feature-partition Δ^f. We define* **the weight-signature of a state** *s, $W(s)$ as a set of 2-tuples of $\{(\Delta^f_l, C_l)\}_{l=1}^k$ where there is an exactly one element for each partition-element of feature-partition and count-value C_l describes the number of features associated with Δ^f_l which holds true in state s i.e $C_l = \sum_{f_j \in \mathcal{G}} [f_j(s) = 1 \wedge f_j \in \Delta^f_l]$*

For example, consider the graphical model $\mathcal{G}_2$ described in Figure 2.5. The feature-partition of $\mathcal{G}_2$ is $\{(f_1), (f_2, f_3)\}$. Let Δ^f_1 be partition element associated with (f_1) cluster while Δ^f_2 be partition-element associated with $\{(f_2, f_3)\}$ cluster. Then, the weight-signature of state $(0, 0)$ is $\{(\Delta^f_1, 0), (\Delta^f_2, 1)\}$ while weight-signature of state $(1, 1)$ is $\{(\Delta^f_2, 1), (\Delta^f_2, 0)\}$ and so on.

Since, the unnormalized probability of a state s is given by $e^{\sum w_j f_j(s)}$ where $f_j(s) \in \{0, 1\}$, the unnormalized probability is dependent on how many times each weight (feature) holds true in that state. This is exactly the same information which is captured by weight-signature of a state. The following theorem captures the relation between probability values and weight-signature.

Theorem 2.4.1. *Given a family of graphical models $\mathcal{F}$ having identical weight-tying, given by feature-partition Δ^f. If weight-signature of s and s' are equal, then $\mathcal{P}(s) = \mathcal{P}(s')$ holds true for all graphical models in the family*

Proof. We will prove that this statement holds for any arbitrary member of the family $\mathcal{G}$ and hence, it holds for all members of the family. Let the weight-signature of states s and s' be $W(s) = \{\Delta^f_l, C_l\}$. Let us denote the weight of Δ^f_l equivalence class in any particular graphical model $\mathcal{G}$ by w_l. Then, the unnormalized probability of any s given by $e^{\sum_j w_j f_j(s)}$ can be rewritten as $e^{\sum_l C_l * w_l * [\Delta^f_l(s)]}$ where $[\Delta^f_l] = 1$ if $\exists f_j \in \Delta^f_l$ such that $f_j = 1$. Exactly, same expression for unnormalized probability will be written for s' as it also has same weight-signature, hence, $\mathcal{P}(s) = \mathcal{P}(s')$. □

Interestingly, one can define a partition of state-space based on equivalence of weight-signature i.e if two states have the same weight-signature, then, they belong to same equivalent class. Since similar weight-signature leads to same probability of states, the resulting partition is probability preserving. We can also define a hierarchy of weight-signature based state partitions from coarse-to-fine as described earlier. The finest partition contains only singleton states in an equivalent class while the coarsest partition has only a single equivalent class for each weight-signature.

Definition 2.4.6. *Given a family of graphical models $\mathcal{F}$ having identical weight-tying, given by coarsest weight-preserving feature-partition Δ^f. Let $\Delta = \{\Delta_1, \Delta_2 \cdots \Delta_k\}$ be a state-partition.* Δ **is weight-signature preserving state partition** *if any two states belonging to the same equivalent class have the same weight-signature i.e $\forall s, s' \in S$, $\forall \Delta_l \in \Delta$, if $s \in \Delta_l$ and $s' \in \Delta_l$, then, $W(s) = W(s')$*

In this framework, we intend to find the coarsest weight-signature preserving state partition. Interestingly, all the state partitions resulting from automorphism based algorithms results in weight-signature preserving state partitions. We now prove that the partition resulting from the variable automorphic group preserves weight-signature. To prove this, we need to prove the following lemma.

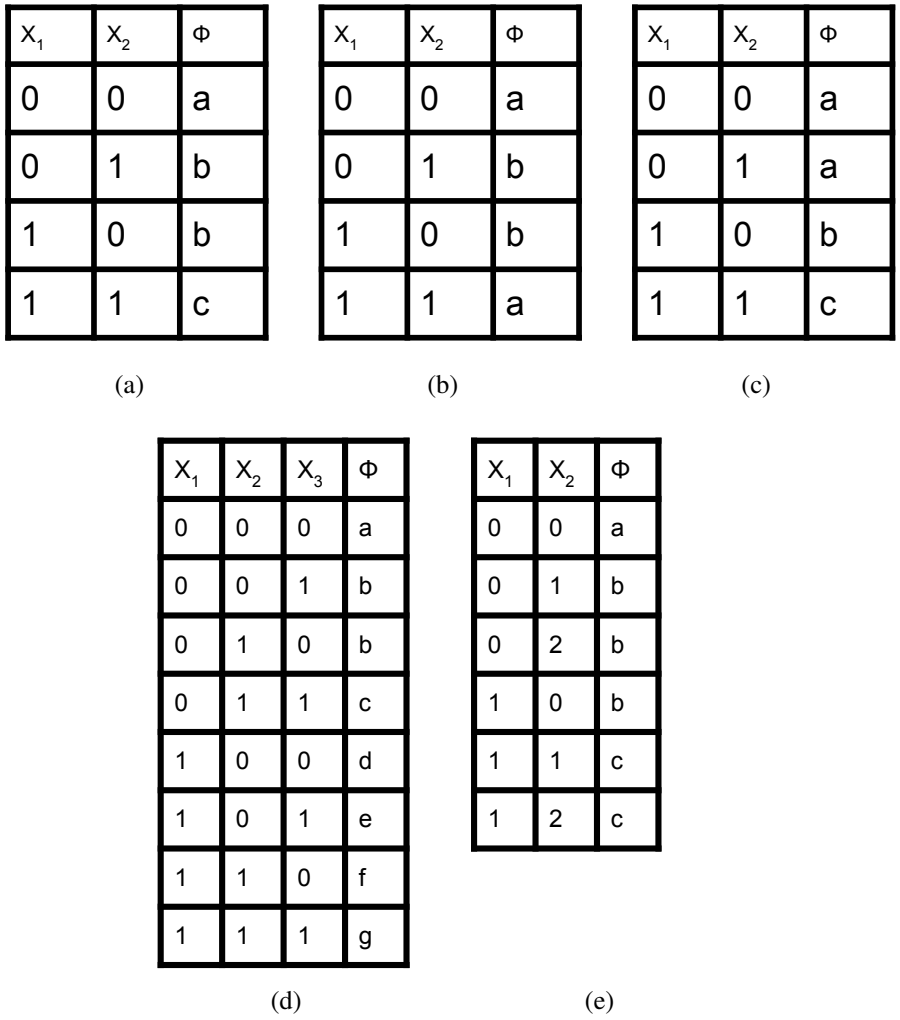
Lemma 2.4.1. *Given a graphical model $\mathcal{G}$, If θ is a variable symmetry of $\mathcal{G}$ and $f'_j = \theta(f_j)$ and $s' = \theta(s)$, then, f_j holds true in s if and only f'_j holds true in s'.*

Proof. Without loss of generality, we assume f is described in clausal form (disjunction of literals). We prove this lemma in two parts: In first part, we prove that if f_j holds true in s, then, f'_j holds true in s'. Since, f_j holds true in state s, then, it means $\exists X_i \in Var(f_j)$ s.t X_i in s is consistent with f_j. Let $\theta(X_i) = X'_i$, then, X_i will be consistent w.r.t f'_j since $s'(X'_i) = s(X_i)$ and f'_j will have X'_i wherever X_i is present. Hence, f'_j will hold true in state s'.

On the other side, if f_j is false for state s, it means $\forall X_i \in Var(f_j)$, the value of X_i in s is not consistent with f_j. For f'_j to be false in state s', each $X'_i \in Var(f'_j)$ should not be consistent with f'_j. This is true since each $X'_i = \theta(X_i)$ where $X_i \in Var(f_j)$ and X_i in s is not consistent with f_j. Therefore, similar to the argument in first part, X'_i is not consistent with f'_j □

Having proved the given lemma, we now prove that variable automorphic group results in weight-signature preserving partition.

Theorem 2.4.2. *Given a family of graphical models $\mathcal{F}$ having identical weight-tying, given by coarsest weight-preserving feature-partition Δ^f. The variable automorphic group Θ of any arbitrary graphical model $\mathcal{G}$ belonging to the family $\mathcal{F}$ results in a weight-signature preserving partition i.e $\forall \theta \in \Theta$, s.t $s' = \theta(s)$, we also have $W(s) = W(s')$*



X_1	X_2	Φ
0	0	a
0	1	b
1	0	b
1	1	c

(a)

X_1	X_2	Φ
0	0	a
0	1	b
1	0	b
1	1	a

(b)

X_1	X_2	Φ
0	0	a
0	1	a
1	0	b
1	1	c

(c)

X_1	X_2	X_3	Φ
0	0	0	a
0	0	1	b
0	1	0	b
0	1	1	c
1	0	0	d
1	0	1	e
1	1	0	f
1	1	1	g

(d)

X_1	X_2	Φ
0	0	a
0	1	b
0	2	b
1	0	b
1	1	c
1	2	c

(e)

Figure 2.6: Examples of Symmetries: a) Variable Symmetries: (01)-(10) b) VV Symmetries: (00)-(11) and (01)-(10) c) BV symmetries: (00)-(01) d) Contextual Symmetries: (001)-(010) with context as $(X_1 = 0)$ e) NEC symmetries: (01)-(02)-(10)

Proof. Let θ be a variable symmetry of $\mathcal{G}$. This means that $\theta(\mathcal{G}) = \mathcal{G}$. Consider a feature $f_j \in F$ having weight w_j. Application of $\theta(f_j)$ results in a new feature f'_j. If θ is a variable symmetry, then $f'(j) \in F$ (as θ results in the same graphical model). Also, f'_j has same weight as f_j in graphical model $\mathcal{G}$. This means, f'_j and f_j belong to same partition element ($\Delta^f_l \in \Delta^f$ as Δ^f is coarsest weight preserving partition). Lemma 2.4.1 proves that for every feature true in state s, there is a corresponding feature which is true in s'. Since, both these features belong to same partition, the count of the number of features belonging to a particular partition which are true in both the states s and s' are equal. Also, this property holds true for all the partition-elements, hence, weight-signature of state s and s' are same. □

The variable automorphic group captures weight-signature preserving state partition but

clearly it does not capture the coarsest weight-signature preserving state partition. The next few chapters define novel notions of permutations based on automorphism which captures more and more state symmetries and results in coarser partitions than those obtained by variable automorphic groups. Specifically, we defined novel notion of contextual symmetries, Variable-Value (VV) symmetries, Non-Equi Cardinal (NEC) Symmetries and Block-Value (BV) symmetries each of which captures significantly more symmetries than variable symmetries. Figure 5.1 gives simple toy examples on 2 or 3 variables showing each of these symmetries. (a) shows the classical notion of variable symmetries where symmetric states are captured as permutation of variables. (b) shows the case of variable-value symmetries where no variable permutation is able to capture symmetry between $\{$A=0, B=0$\}$ and $\{$A=1, B=1$\}$ and hence, permutation has to be defined over variable-value pairs. The third example demonstrates the case where symmetry is expressed between block-value pairs where block is a subset of variables. (d) shows contextual symmetries where variable symmetries only arise under a particular context ($X_1 = 0$) while (e) shows symmetries among variables of different cardinalities. It should be noted that variable symmetries and contextual symmetries are only able to capture symmetries between states where counts of 0s, 1s, 2s (and so on) match while VV, BV and NEC symmetries are able to capture more general symmetries where counts may not match.

We also propose novel MCMC variants to use these symmetries. Lastly, we also characterize the complete space of state symmetries in graphical models, the relationships between proposed symmetries and finally, show that the block-value symmetries gives the representation to capture the coarsest weight-signature preserving state partition while paying additional computation costs.

Chapter 3

Contextual Symmetries

A key shortcoming of existing lifted inference algorithms is that they only identify and exploit sets of variables (states) that are symmetric *unconditionally*. In this chapter, our goal is to extend the notion of symmetries to *contextual* symmetries, sets of states that are symmetric under a given context (variable-value assignment). Our proposal is inspired by the extension of conditional independence to context-sensitive independence [Boutilier *et al.*, 1996], and analogously extends unconditional symmetries to contextual. As our first contribution in this chapter, we develop a formal framework to define contextual symmetries. We also present an algorithm to compute contextual symmetries by reducing the problem to graph isomorphism.

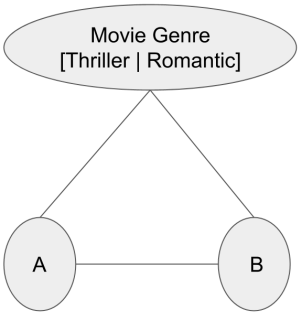
Figure 3.1(a) illustrates an example of contextual symmetries. A couple A and B may like to go to a romantic movie. They are somewhat less (equally) likely to go alone compared to when they go together. However if the movie is a thriller, A may be less interested in going by herself, but B may not change his behavior. Hence, A and B are symmetric to each other if the movie is romantic, but not symmetric if the movie is a thriller. We will call the A and B contextually symmetric conditioned on the movie being romantic.

Our work extends the line of work on Orbital MCMC [Niepert, 2012] (described in chapter 2) which was a state-of-the-art approach to exploit unconditional symmetries in a generic MCMC framework. Orbital MCMC achieved reduced mixing times compared to Gibbs sampling in domains where such symmetries exist. We design CON-MCMC, an algorithm that uses contextual symmetries within the MCMC framework. Our experiments demonstrate that on various interesting domains (relational and propositional), where contextual symmetries may be present, CON-MCMC can yield substantial gains compared to Orbital MCMC and Gibbs Sampling. We also release a reference implementation of CON-MCMC sampler for wider use.[1]

[1] `https://github.com/dair-iitd/con-mcmc`

Genre (G)	A	B	ϕ
Thriller	0	0	2
Thriller	0	1	4
Thriller	1	0	3
Thriller	1	1	6
Romantic	0	0	2
Romantic	0	1	4
Romantic	1	0	4
Romantic	1	1	6

(a)



Genre (G)	A	B	ϕ
Thriller	0	0	2
Thriller	0	1	4
Thriller	1	0	4
Thriller	1	1	6
Romantic	0	0	2
Romantic	0	1	4
Romantic	1	0	4
Romantic	1	1	6

(b)

Figure 3.1: Illustration of (a) Contextual Symmetry (with Genre="Romantic") (b) Variable Symmetry in the Movie Network.

3.1 Contextual Symmetries

Our work proposes the novel notion of contextual symmetries – symmetries that only hold under a given context. We now extend the definitions of the variable symmetries (described in Chapter 2) to their contextual counterparts. First we define a context.

Definition 3.1.1. *A* **context** C *is a partial assignment, i.e., a set of pairs* (X_i, v_i)*, where* $X_i \in \mathcal{X}$ *and* $v_i \in D_i$*, and no* X_i *is repeated in the set.*

For example, in Figure 3.1, we can define a context (Genre, "Romantic"). We refer to a context as a *single variable context* if there is only one element in the context set. We say that a variable X_i appears in a context if there is a pair $(X_i, v_i) \in C$. Given a context C, we will use $\mathcal{X}_C$ to denote the subset of variables of $\mathcal{X}$ which appear in C. We will use $\bar{\mathcal{X}}_C$ to denote the complement of this set. Given a state s, we will use $\zeta_{X_i}(s)$ to denote the value of X_i in state s. We say that a state s is *consistent* with the context C iff $\forall (X_i, v_i) \in C$ we have $v_i = \zeta_{X_i}(s)$.

In order to define contextual automorphism, we will need to define the notion of a reduced model.

Definition 3.1.2. *Given a graphical model* $\mathcal{G} = \{f_k, w_k\}_{k=1}^m$*, and a context* C*, the* **reduced model** $\mathcal{G}^r_C$ *is defined as the new graphical model obtained by substituting* $X_i = v_i$ *in each formula* f_k *for all* $(X_i, v_i) \in C$ *and keeping the original weights* w_k*.*

Note that $\mathcal{G}^r_C$ is defined over the set $\bar{\mathcal{X}}_C$. As an example, if the model is represented by the formulas $\{(P \vee Q, w_1), (R \vee Q \vee S, w_2)\}$, the reduced model under the single variable context $\{(R, 0)\}$ will be $\{(P \vee Q, w_1)$ and $(Q \vee S, w_2)\}$. In the factored form representation, reduction

by a context corresponds to fixing the values of the context variables in the potential table. E.g., in Figure 3.1 given the context "Romantic", we reduce the factor to the bottom four rows of the potential table where Genre has value "Romantic". We are now ready to define a contextual symmetry of a graphical model.

Definition 3.1.3. *A* **contextual symmetry of a graphical model** $\mathcal{G}$ *under context* C *is represented as a permutation* θ *of variables in* $\mathcal{X}$ *s.t. a)* $\theta(X_i) = X_i, \forall X_i \in \mathcal{X}_C$ *i.e. variables in the context are mapped to themselves, and b)* $\exists$ *a variable symmetry* θ^r *of the reduced model* $\mathcal{G}^r_C$ *such that* $\theta(X_i) = \theta^r(X_i)\ \forall X_i \in \bar{\mathcal{X}}_C$*, i.e. mapping of the remaining variables defines a variable symmetry of the reduced graphical model under context* C*.*

For example, in Figure 3.1, let a permutation θ^* be: $\theta^*(G) = G, \theta^*(A) = B, \theta^*(B) = A$. θ^* is a contextual symmetry under the context (Genre, "Romantic"), but not under the context (Genre, "Thriller").

Definition 3.1.4. *A* **contextual automorphism group of a graphical model** $\mathcal{G}$ *under context* C *is defined as a permutation group* (Θ_C) *over* $\mathcal{G}$*, such that each* $\theta \in \Theta_C$ *is a contextual symmetry of* $\mathcal{G}$ *under context* C*.*

Definition 3.1.5. *The* **contextual orbit of a state** s *under the contextual automorphism group* Θ_C *(given the context* C*) is the set of those states which are consistent with* C *and can be reached by applying* $\theta \in \Theta_C$ *to* s*, i.e.,* $\Gamma_{\Theta_C}(s) = \{s' \in D^n \mid \exists \theta \in \Theta_C \text{ s.t. } \theta(s) = s' \bigwedge \forall (X_i, v_i) \in C, \zeta_{X_i}(s') = v_i\}$.

Note that s must be consistent with C for it to have a non-empty contextual orbit. Analogous to variable symmetries, contextual symmetries preserves weight-signature and are also probability preserving. Similar to variable symmetries, we prove the following lemma which relates the validity of feature in a state to the validity of transformed feature in state s'

Lemma 3.1.1. *Given a graphical model* $\mathcal{G}$*, If* θ *is a contextual symmetry with context* C *and* $f'_j = \theta(f_j)$ *and* $s' = \theta(s)$ *where context is true in* s *and* s'*, then,* f_j *holds true in* s *if and only* f'_j *holds true in* s'*.*

Proof. If θ is a contextual symmetry, then, θ^r on non-context variables is a variable symmetry. Hence, if for a feature f_j, if $Vars(f_j) \cap \mathcal{X}_C = \varnothing$, then, those features do not go any transformation in reduced graphical model . Hence, lemma trivially holds by extending variable symmetry to contextual symmetry. If $Vars(f_j) \cap \mathcal{X}_C \neq \varnothing$, then, those features got eliminated (f_j is clausal and context-variable has its value consistent with that feature) or got reduced (not consistent). If features got eliminated, then, the lemma holds. If the features got reduced, then, on the reduced model, since, θ^r is a variable symmetry, the lemma holds again by extension of variable symmetry to contextual symmetry. □

Theorem 3.1.1. *A contextual symmetry θ of $\mathcal{G}$ under context $C = \{(X_i, v_i)\}$ results in a weight-signature preserving state partition if s is consistent with C. Also, contextual symmetry preserves probability i.e. $P(s) = P(\theta(s))$.*

Proof. This argument is similar to the theorem 2.4.2. Since θ is a contextual symmetry, then, application of θ results in the same graphical model. Hence, f_j and transformed feature $\theta(f_j)$ belong to original features of graphical model and have the same weight. Hence, they belong to the same partition-element of feature-set partition. Using lemma 3.1.1, for any feature f_j which holds true in state s, there is a corresponding feature $\theta(f_j)$ which holds true in state s' which is in same partition-element. Hence, the count of number of features which are true in state s in any partition-element is exactly equal to number of features which are true in state s' in any partition-element. Therefore, weight-signature of states are preserved under contextual symmetry. And as per theorem 2.4.2, if two states have same weight-signature, then, they have same probability.

□

3.1.1 Relationship with Related Concepts

The set of contextual symmetries subsumes that of variable symmetries – any variable symmetry is a contextual symmetry under a null context $\emptyset$. The two notions are even more related, as the following two lemmas show. Let $\mathcal{X}_\theta^I$ be the set of variables that map onto itself in a permutation θ, i.e. $\forall X \in \mathcal{X}_\theta^I : \theta(X) = X$.

Theorem 3.1.2. *A variable symmetry θ is a contextual symmetry under a context C if $\mathcal{X}_C \subseteq \mathcal{X}_\theta^I$. Hence, contextual symmetries subsume variable symmetries trivially for $\mathcal{X}_C = \emptyset$.*

Proof. The proof is trivial following the definition of contextual symmetry. If $\mathcal{X}_C \subseteq \mathcal{X}_\theta^I$, then, all the context variables map to themselves. The second thing we need to prove is that there exists a variable symmetry over the reduced graphical model. Consider the reduced graphical model and the reduced permutation θ^r over the subset $\mathcal{X} - \mathcal{X}_C$ of remaining variables. We argue that if θ defines a transformation of feature f_j into another feature f'_j, then, under θ^r, either f_j does not exist (if context variable is true in that feature in clausal form) or the reduced f_j is transformed into $f_j'^r$ (since θ^r and θ maps all other variables in exactly same way. And so, all the features are mapped identically under θ and θ^r resulting in the same set of reduced weighted features if $theta$ results in the same set of weighted features in original graphical model. □

We now distinguish the notions of context and contextual symmetries from two other related concepts. First, a context is different from evidence. External information in the form of evidence modifies the underlying distribution represented by the graphical model. In contrast,

a context has no effect on the underlying distribution. Second, it might be tempting to confuse contextually symmetric states with contextually independent states [Boutilier *et al.*, 1996]. In the example of Figure 3.1(a) given Genre="Thriller", A and B are contextually independent, i.e., probability of A does not change depending on B. For this context A and B are non-symmetric. For Genre="Romantic" A and B are symmetric but not independent. Finally, in Section 3.5, we discuss the relationship between contextual symmetries and the recent notion of conditional decomposability [Niepert and Van den Broeck, 2014].

3.1.2 Computing Contextual Symmetries

Computing contextual symmetries for $\mathcal{G}$ under a context C is equivalent to computing variable symmetries on the reduced model $\mathcal{G}^r_C = \{f^r_k, w^r_k\}_{k=1}^{m^r}$. To compute variable symmetries we adapt the procedure from Niepert [2012]. Following Niepert, we describe the construction when each f^r_k is a clause, though it can be extended to the more general case.

Niepert's procedure creates a colored graph, with two nodes corresponding to every variable (one each for the positive and negative state), and one node for every formula f^r_k. Edges exist between the positive and negative states of every variable, and also between the formula nodes and the variable nodes (either positive or negative) appearing in the formula. Finally, colors are assigned to nodes based on the following criteria: (a) every positive variable node is assigned a common color, (b) every negative variable node is assigned a different common color, and (c) every unique formula weight w^r_k is assigned a new color. The formula nodes f^r_k inherit the color associated with their weight w^r_k.

This color graph is then passed through a graph isomorphism solver (e.g., Saucy), which computes the automorphism group for $\mathcal{G}^r_C$. This is equivalent to computing contextual automorphism group for $\mathcal{G}$ under C:

Theorem 3.1.3. *The automorphism group for the color graph of the reduced graphical model $\mathcal{G}^r_C$ along with an identity mapping of the context variables gives a contextual automorphism group of $\mathcal{G}$ under C.*

Proof. To prove this, we need to prove for every permutation on reduced graphical model $\mathcal{G}^r_C$ along with identity mapping gives a new permutation which when applied to a graphical model results in the original graphical model. The proof holds by extension of similar theorem for variable symmetry. By theorem 2.2.2, the permutation on G^r_C gives a variable symmetry on $\mathcal{X} - \mathcal{X}_C$ and appending identity mapping for $\mathcal{X}_C$ gives the contextual symmetry as per definition of contextual symmetry. □

Note that in case we have any evidence E available, the reduced model over which we induce a colored graph corresponds to $\mathcal{G}^r_{C \cup E}$. This is in contrast with original Niepert's proce-

dure, where evidence nodes are not removed from the color graph and instead act as additional formulas for the original graphical model with infinity weights. This elimination of evidence nodes helps discover many more symmetries in the corresponding color graph while still preserving correctness. For example, if the model is represented by formulas $\{(P \vee R, w_1), (Q, w_1)\}$, and evidence is $(\neg R)$, P and Q become symmetric only if R is eliminated from the color graph, and not in Niepert's procedure.

3.2 Contextual MCMC

We now extend the Orbital MCMC algorithm from chapter 2 so that it can exploit contextual symmetries; our algorithm is named CON-MCMC, and is parameterized by $\alpha \in [0, 1)$. Orbital MCMC reduces mixing times over original MCMC, because it can easily transition between high probability states falling in the same orbit, which may otherwise be separated by low probability regions. Unfortunately, as Figure 3.1 demonstrates, a domain may have little variable symmetry, but still important contextual symmetry. CON-MCMC(α) exploits these for inference.

We are given a set of context variables $V \subset \mathcal{X}$ (more on this later). Let $\mathcal{C}_V$ denote the set of all possible contexts involving all the variables in V. Overloading the notation, we will use $\mathcal{C}_V(s)$ to denote the (unique) context in $\mathcal{C}_V$ consistent with state s. We compute contextual symmetries Θ_C under each context $C \in \mathcal{C}_V$ using the algorithm from Section 9.2. We are also given an original regular Markov chain M that converges to the desired probability distribution $\pi(s)$. CON-MCMC(α) runs a *Contextual Markov Chain* $M_{con}(\alpha)$ that samples a state $s^{(t+1)}$ from $s^{(t)}$ as follows:

1. *Gibbs-orig move:* We sample an intermediate state $s'^{(t+1)}$ from the current state $s^{(t)}$ as:
   (a) with probability α (Gibbs): flip a random context variable in $s^{(t)}$ using Gibbs transition probability.
   (b) with probability 1-α (original): make the move from $s^{(t)}$ based on the transition probability in M.

2. *Con-orbital move:* Let $C = \mathcal{C}_V(s'^{(t+1)})$ be the context consistent with $s'^{(t+1)}$. Let $\Gamma_{\Theta_C}(s'^{(t+1)})$ denote the contextual orbit of $s'^{(t+1)}$ under the context C. Sample a state $s^{(t+1)}$ uniformly at random from $\Gamma_{\Theta_C}(s'^{(t+1)})$.

When $\alpha = 0$ our algorithm reduces to a direct extension of Orbital MCMC, where in second step, we sample uniformly from a contextual orbit instead of the original orbits. In the more interesting case of $\alpha > 0$, we enable the Markov chain to move more freely between different contexts using a Gibbs flip over the context variables. This Gibbs transition helps us carry

Algorithm 1 Contextual Markov Chain

Contextual MCMC(Markov Chain M, ContextFlipProb α, Context Variables V, NumSamples n)
  $\mathcal{C}_V \leftarrow$ CreateAllContexts(V)
  Sample Initial state s_0
  for $t := 0 \rightarrow n-1$ **do**
    Sample r uniformly from $(0, 1)$
    if $r \leq \alpha$ **then**
      $X_c \leftarrow$ SampleUniformly(V)
      $s'^{(t+1)} \leftarrow$ GibbsTransition($s^{(t)}, X_c$)
    else
      $s'^{(t+1)} \leftarrow$ MarkovChainTransition($M, s^{(t)}$)
    $C \leftarrow$ GetContext($(s'^{(t+1)}$)
    $\Theta_C \leftarrow$ PrecomputedGroup(C)
    $s^{(t+1)} \leftarrow$ SampleUniformly($\Gamma_{\Theta_C}(s'^{(t+1)})$)
  return List of states $s^{(t)} : t := 0 \rightarrow n-1$

over the effect of symmetries exploited under one context (via the orbital moves in step 2) to others. This can be especially useful when symmetries are unevenly distributed across multiple contexts (as also confirmed by our experiments).

In order to sample a state uniformly at random from a contextual orbit, we use the product replacement algorithm [Pak, 2000] as described and used by Niepert [2012]. Recall that since we are working with contextual permutations, the context variables are mapped to themselves and we are guaranteed to not change the context. Next, we show that CON-MCMC(α) converges to the desired stationary distribution $\pi(s)$. We need the following lemma.

Lemma 3.2.1. *Let M_1 and M_2 be two Markov chains defined over a finite state space S with transition probability functions P_1 and P_2, respectively, such that $\pi(s)$ is a stationary distribution for both P_1, P_2, i.e., $\pi(s) = \sum_{r \in S} \pi(r) * P_i(r \rightarrow s)$, $i \in \{1, 2\}$. Further, let M_2 be regular. Then, the Markov chain M' with the transition function $P'(r \rightarrow s) = \alpha * P_1(s \rightarrow s) + (1-\alpha) * P_2(r \rightarrow s)$ is also regular and has a unique stationary distribution π for $\alpha \in [0, 1)$.*

Proof. The chain P' is regular since, there is non-zero probability of returning back to same state. This is because P' is composed of two probabilities and $\alpha \neq 1$. As P_2 is regular, so there is non-zero probability of returning back to same state, so, second term of summation is non-zero for returning back to same state. So, P' is regular and converges to a unique stationary distribution. Next, we prove that it converges to π. To prove this, we need to prove that

$$\pi(s) = \sum_{r \in S} \pi(r) P'(r \rightarrow s) \tag{3.1}$$

Taking R.H.S,

$$\sum_{r\in S}\pi(r)P'(r\to s)=\sum_{r\in S}\pi(r)[\alpha*P_1(r\to s)+(1-\alpha)*P_2(s\to r)] \tag{3.2}$$

$$=\alpha\sum_{r\in S}\pi(r)P_1(r\to s)+(1-\alpha)\sum_{r\in S}\pi(r)P_2(s\to r) \tag{3.3}$$

$$=\alpha\times\pi(s)+(1-\alpha)\times\pi(s)=\pi(s) \tag{3.4}$$

The above calculation uses simple algebra and the facts that M_1 and M_2 converge to the same stationary distribution π. □

Let $M^{go}(\alpha)$ refer to the family of Markov chains constructed using only step 1 of our algorithm i.e. no symmetry moves. M is regular with the stationary distribution $\pi(s)$. Further, each individual Gibbs flip over a variable satisfies stationarity with respect to the underlying distribution $\pi(s)$ [Koller and Friedman, 2009]. Hence, using Lemma 3.2.1, $M^{go}(\alpha)$ is regular with $\pi(s)$ as a stationary distribution.

Theorem 3.2.1. *The family of contextual Markov chains $M^{con}(\alpha)$ constructed using* CON-MCMC*(α) converges to the stationary distribution of original Markov chain M for any choice of context variables V and $\alpha\in[0,1)$.*

Proof. Let $\pi(s)$ be the stationary distribution of M. Since $M^{go}(\alpha)$ is regular it is easy to see that $M^{con}(\alpha)$ is also regular (there is always a non-zero probability of coming back to the same state in an orbital move). Therefore, $M^{con}(\alpha)$ converges to a unique stationary distribution. Then, we only need to show that $\pi(s)$ is the stationary distribution of $M^{con}(\alpha)$. Let $S=\{0,1\}^n$ denote the set all of all the states. For $r,s\in S$, let $P^{go}[\alpha](s\to r)$ and $P^{con}[\alpha](s\to r)$ represent the transition probability functions of $M^{go}[\alpha]$ and $M^{con}[\alpha]$, respectively. In order to show that $M^{con}[\alpha]$ also converges to $\pi(s)$, we need to show that:

$$\pi(s)=\sum_{r\in S}\pi(r)P^{con}[\alpha](r\to s) \tag{3.5}$$

The RHS of the above equation can be written as:

$$=\sum_{r\in S}\pi(r)\sum_{s'\in\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)}P^{go}[\alpha](r\to s')\frac{1}{|\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)|}$$

$$=\sum_{r\in S}\sum_{s'\in\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)}\pi(r)P^{go}[\alpha](r\to s')\frac{1}{|\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)|}$$

$$
\begin{aligned}
&= \sum_{s' \in \Gamma_{\Theta_{\mathcal{C}_V(s)}}} \left[\sum_{r \in S} \pi(r) P^{go}[\alpha](r \to s') \right] \frac{1}{|\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)|} \\
&= \sum_{s' \in \Gamma_{\Theta_{\mathcal{C}_V(s)}}} \pi(s') \frac{1}{|\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)|} \\
&= \sum_{s' \in \Gamma_{\Theta_{\mathcal{C}_V(s)}}} \pi(s) \frac{1}{|\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)|} \\
&= \pi(s)
\end{aligned}
$$

Here, recall that $\Theta_{\mathcal{C}_V(s)}$ denotes the contextual automorphism group for the (unique) context $\mathcal{C}_V(s)$ consistent with the state s, and $\Gamma_{\Theta_{\mathcal{C}_V(s)}}(s)$ denotes the corresponding orbit. Step 1 above follows from the definition of contextual orbital move. Step 4 follows from the stationarity of $M^{go}[\alpha]$. Step 5 follows from the fact that all the states in the same contextual orbit have the same probability (Theorem 3.1.1). □

3.3 Experimental Evaluation

Our experiments evaluate the use of contextual symmetries for faster inference in graphical models. We compare our approach against Orbital MCMC, which is the only available algorithm that exploits symmetries in a general MCMC framework. We also compare with vanilla Gibbs sampling, which does not exploit any symmetries. We implement CON-MCMC(α) as an extension of the original Orbital MCMC implementation[2] available in the GAP language [GAP, 2015]. The existing implementation uses Saucy [Darga *et al.*, 2008] for graph isomorphism and Gibbs sampler as the base Markov chain. We experiment on two versions each of two different domains, with context variables pre-specified. We next describe our domains. These domains are synthetically designed to show the effectiveness of our approach if contextual symmetries are present but no or only limited variable symmetry is present.

3.3.1 Domains and Methodology

Sports Network: This Markov network models a group of students who may enter a future sport league, which could be for one of two sports, badminton or tennis (modeled as the variable $Sport$). Each student belongs to one of the dorms on campus. The league accepts both singles as well as doubles entries. For each student X, the domain has a variable for playing singles, S_X. For each pairs of students X, Y coming from the same dorm, we have a variable indicating that they will play doubles together, D_{XY}. Multiple students (in the same dorm) train together in

[2] `https://code.google.com/archive/p/lifted-mcmc/`

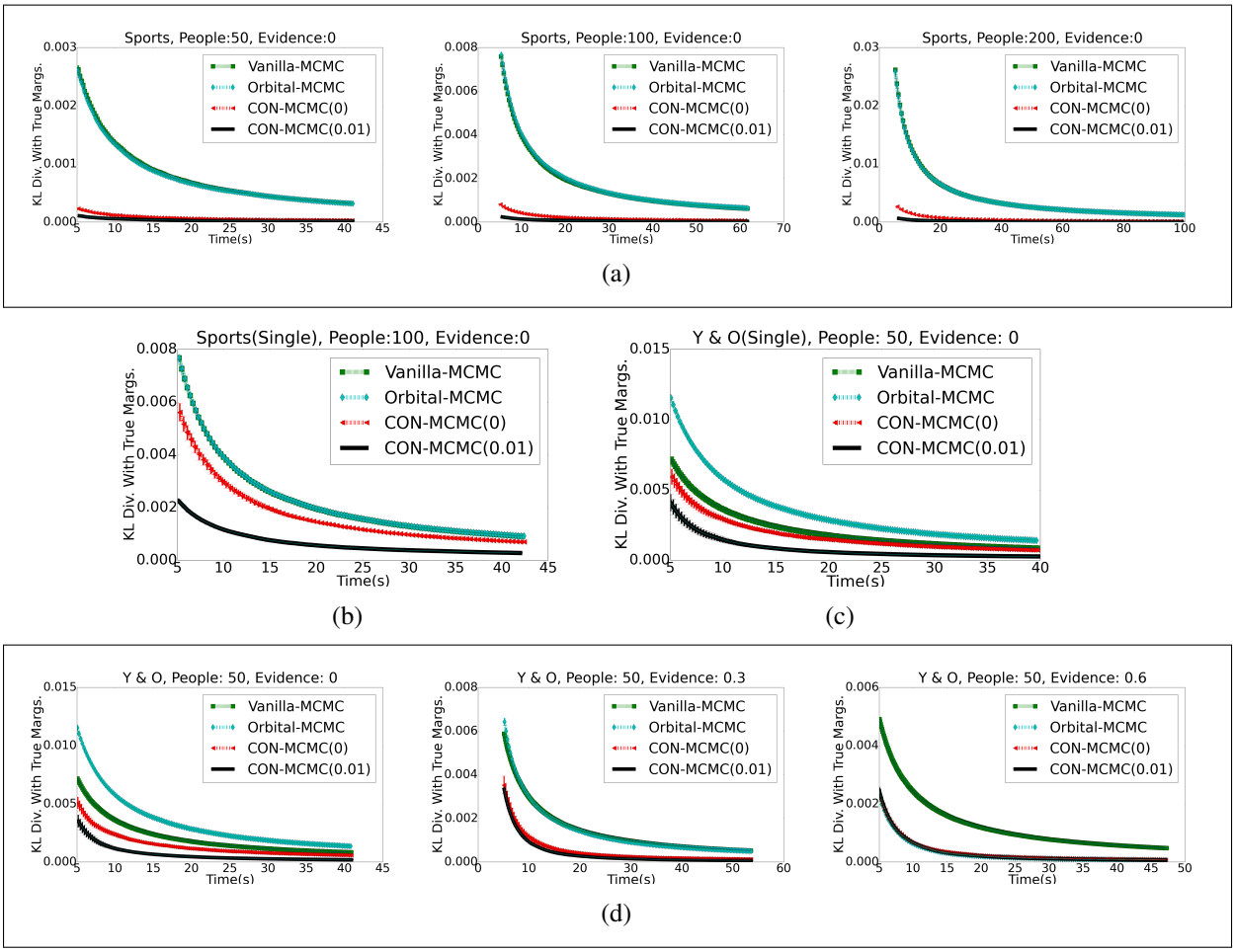


Figure 3.2: **(a)** CON-MCMC effectiveness increases tremendously with increasing domain sizes. Note that y-axes are on different scales. **(d)** New variable symmetries are created with increasing evidence, leading to improved performance of Orbital MCMC. **(b, d)** Curves for Sports Network (Single) and Y & O (Single) respectively – CON-MCMC(0.01) performs the best and vastly outperforms CON-MCMC(0).

training groups, which are different for the two sports. A student's participation in the league for a given sport is (jointly) influenced by the participation of other students in her training group for that sport. Moreover, if two students decide to play singles, it increases the probability that they may also team up to play doubles independent of their training groups. In this domain, different subsets of students in a dorm (based on their training groups) are symmetrical to each other depending upon $Sport$, which becomes a natural choice for the context. In our experiments, we use training groups of 5 students and dorms with 25 students each.

Young and Old: This domain is modeled as an MLN and is an extension of the Friends and Smokers (FS) [Singla and Domingos, 2008] network. $Y\&O$ has a propositional variable $IsYoung$ determining whether we are dealing with a population of youngsters or older folks. For every person X in the domain, we have predicates $Smokes(X)$, $Cancer(X)$ and $EatsOut(X)$. We also have the predicate $Friends(X, Y)$ for every pair of persons. We have rules stating that young persons are more likely to smoke and older people are less likely to smoke. Similarly, we have rules stating that young people are more likely to eat out and old people are less likely to eat out. When the population is young, everyone has the same weight for the smoking rule and slightly different weight (sampled from a Gaussian) for eating out. When the population is old, everyone has a slightly different weight (again sampled from a Gaussian) for smoking and the same weight for eating out. As in the original FS, we have rules stating smoking causes cancer and friends have similar smoking habits. We also have rules stating that cancer and friends variables have low prior probabilities. In this domain, smoking, cancer and friends variables are symmetric to each other when population is young, whereas all eating out variables are symmetric when the population is old. Clearly, $IsYoung$ is a natural choice for context in this domain.

An important property of both these domains is that different contextual symmetries exist for *both* assignments of the respective context variables. To test the robustness of CON-MCMC we further modify these domains so that contextual symmetries exist only on *one* of the two assignments of context variable. In $Y\&O$ (Single), we give (slightly) different weights to $EatsOut(X)$ variables when $IsYoung$ is false, i.e., symmetries exist only when $IsYoung$ is true. In Sports Network (Single), S_X variables involved in a training group are symmetric only for tennis; for badminton, each S_X in a group behaves (slightly) differently. We refer to these two variations as the *Single* side versions of the original domains.

For these four domains, we plot run time *vs.* the KL-divergence between approximate marginal probabilities computed by each algorithm and the true marginals.[3] For both Orbital MCMC and CON-MCMC, the time to compute symmetries is included in the run time. For each problem we run 20 iterations of each algorithm and take the mean of the marginals to

[3]computed by running a Gibbs sampler for sufficiently long time.

reduce variance of the measurements. We also plot the 95% confidence intervals. We show CON-MCMC results for $\alpha = 0$ and 0.01, which was chosen based on performance on smaller problem sizes. We perform various control experiments by varying the size of domains, amount of available evidence, marginal posterior probability of the context variable and the value of α parameter. All the experiments are run on a quad-core Intel i-7 processor.

3.3.2 Results

Figures 3.2 and 3.3 show the representative graphs across multiple domains and varying experimental conditions. We find that CON-MCMC(0.01) almost always performs the best or at par with the best of other three algorithms. CON-MCMC(0) usually performs better than Gibbs and Orbital MCMC, but its performance can be closer to Gibbs or CON-MCMC(α) depending upon the experimental setting. Orbital MCMC does not usually offer much advantage over Gibbs, primarily because these domains do not have many variable symmetries. For Sports Network, there are no variable symmetries at all; Orbital MCMC avoids the overhead of the symmetry move and performs at par with Gibbs. For $Y\&O$, Orbital MCMC finds a few symmetries, which do not particularly help in reducing mixing time. However, it still incurs the overhead of symmetry moves, leading to a significantly worse performance compared to Gibbs.

Variation with Domain Size: Figure 3.2(a) compares the algorithms as we increase the domain size for the Sports network from 50 to 200 students. The overall trends remain similar, i.e., CON-MCMC algorithms outperform Gibbs and Orbital MCMC by huge margins. A closer look reveals that the y-axes are at different scales for the three curves – the relative edge of CON-MCMC algorithms increases substantially with larger domain sizes.

Variation with Amount of Evidence: Figure 3.2(c) compares the performance of the algorithms as we vary the amount of (random) evidence available from 0% to 60% in the $Y\&O$ domain on predicates other than $Friends(X, Y)$ using a domain size of 50. As earlier, CON-MCMC algorithms outperform others. We observe that the relative gain of CON-MCMC algorithms with Orbital MCMC decreases with increasing evidence (for 30% evidence Orbital MCMC overlaps with Gibbs, for 60% evidence, Orbital MCMC overlaps with CON-MCMC). We believe that this is due the fact that more evidence tends to disconnect the network introducing additional symmetries which can be exploited by Orbital MCMC. Nevertheless, CON-MCMC algorithms perform at least as well as Orbital MCMC for all values of evidence that we tested on.

Variation across Versions of a Domain: Figure 3.2(b) and 3.2(d) show the plots for the *Single* side versions of Sports network and $Y\&O$, respectively. We observe a significant difference in the performance of the two CON-MCMC algorithms. The reason is subtle. Since symmetries

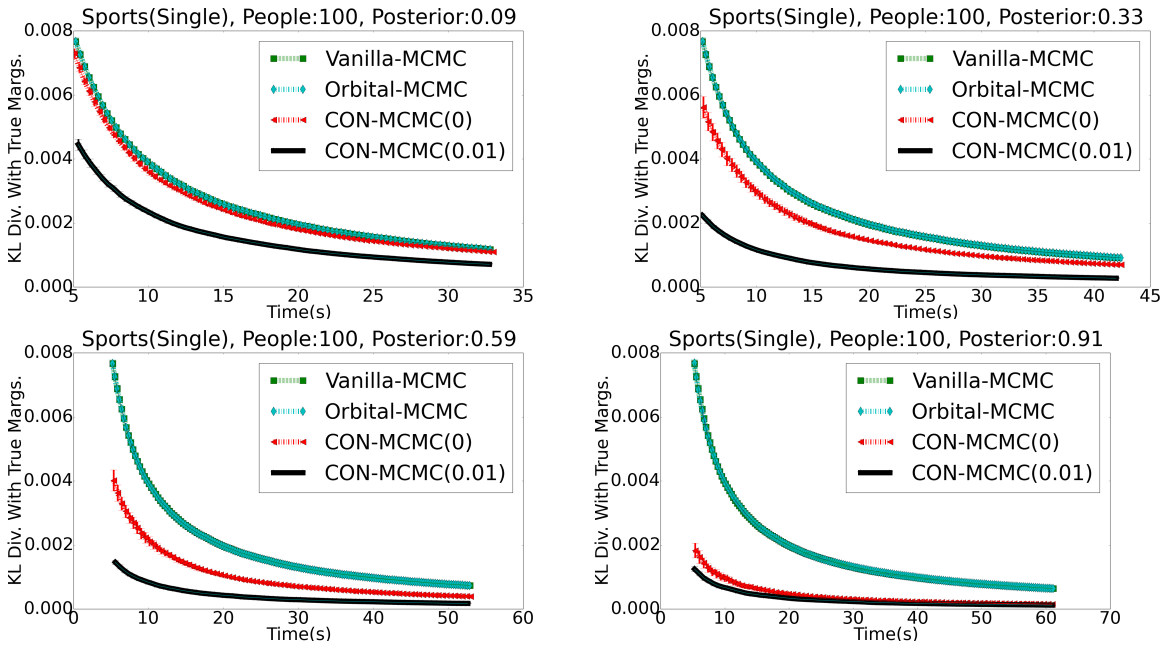


Figure 3.3: CON-MCMC effectiveness increases in Single Side Symmetry cases as we increase the marginal of context variable to the side having symmetry from 0.09 to 0.91. CON-MCMC(0.01) provides significant gains even at very low posterior values. CON-MCMC(0) performance improves with increase in the marginal.

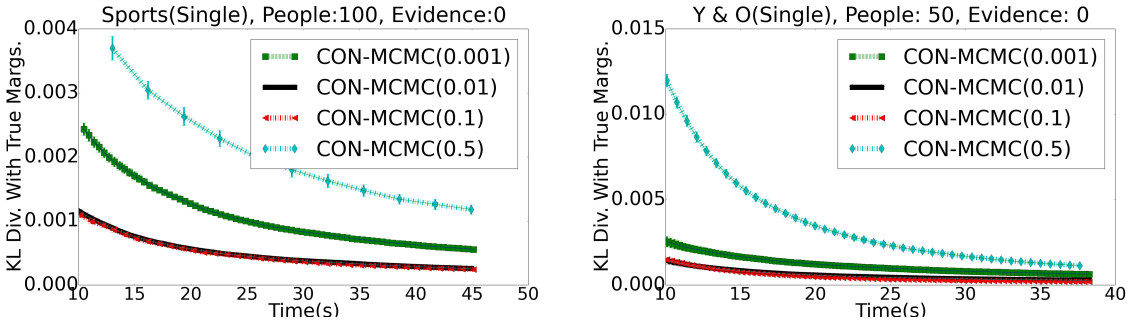


Figure 3.4: α=0.01 and α=0.1 work best across both domains. Very high as well as very low values of α lead to poor performance.

exist only on one side, that side mixes quickly for CON-MCMC(0); however, the other side does not mix as well, because of lack of symmetries. CON-MCMC(α) for $\alpha > 0$ mitigates this by upsampling the flip of the context variable. This enables the rapid mixing on symmetry side to regularly influence the non-symmetry side (via Gibbs move), which leads to a faster mixing on that side too. Nevertheless, CON-MCMC(0) is still able to outperform both Gibbs sampling as well as Orbital MCMC by exploiting the single sided symmetry. The posterior of symmetry side is 0.33 in Sports network (Figure 3.2(b)) and 0.37 in $Y\&O$ (Figure 3.2(d)).

We also observe in the first graph of Figure 3.2(c), that CON-MCMC(0) performs somewhat worse than CON-MCMC(0.01). We believe that the reason for performance in this two-sided symmetry domain is similar to the single-sided case. In $Y\&O$, when $IsYoung$ =true, substantial symmetries may exist due to smoking, cancer and friends variables. However, on the other side, the symmetries are far less (only for eating out variables). This implies that CON-MCMC(0) will have much faster mixing on one side, but not on the other. On the other hand, CON-MCMC(0.01) will upsample context variable flips and allow the stronger symmetry side to influence the other. In general, CON-MCMC(α) performance is highly robust to varieties of symmetric and asymmetric domains.

Variation with Posterior of Context Variable: We investigate performance on Single-sided domains further by varying the posterior marginal probability of the context variable. Figure 3.3 shows the results for Sports network (Single) with marginal probability of $Sport =$ tennis varying from 0.09 to 0.91. Note that $Sport =$ tennis side is the side where symmetries exist.

The graphs show an interesting trend. Even for very low marginals, CON-MCMC(0.01) is able to benefit from one sided symmetries. Since the marginal is low we expect any MCMC algorithm to spend most of its time on the non-symmetry side. However, CON-MCMC(0.01) will still go back and forth several times between two sides; each flip to symmetry side and back will help in potentially reaching a different region of the state space leading to better mixing on the non-symmetry side.

Not surprisingly, CON-MCMC(0) does not perform as well for low marginals – it does not get to switch contexts as often, and ends up mixing slowly on the important, non symmetry side. As marginal of the context variable increases, the relative performance of CON-MCMC(0) improves substantially. As marginal becomes high (0.91), both CON-MCMC samplers end up sampling mostly on the symmetry side, and can reap benefits of symmetries similarly. We also conduct these experiments for the $Y\&O$ domain and observe a very similar behavior.

Variation with α Parameter: Figure 3.4 shows the performance of CON-MCMC(α) for different values of α in the range 0.001 to 0.5 for both Sports network (single) and $Y\&O$ (single) domains. Our algorithm is fairly robust for values of α between 0.01 and 0.1. Its performance starts to degrade for very low as well as very high values of α. For very low values of α, algo-

rithm's behavior approaches that of CON-MCMC(0). For very high values of α, the algorithm spends too much time flipping the context variable and not enough time exploring the state space, resulting in poor performance.

Overall, we conclude that CON-MCMC(0.01) is robust to various experimental settings and obtains the best results significantly outperforming Orbital MCMC and Gibbs. This underscores the importance of our contextual symmetry framework for probabilistic inference.

3.4 Discussion and Future Work

While our work extends the capability of lifted inference to a wider range of settings, it also raises important questions. In many cases, the set V of context variables is known from domain knowledge or domain description especially in relational models. An open question is how to automatically compute a good set V, since trying all possible sets can be prohibitive. We have designed a heuristic approach that greedily chooses the most useful context variable every iteration and adds it to the context set. It uses a few initial rounds of the color passing algorithm [Kersting *et al.*, 2009] to approximate the amount of additional symmetry obtained by making a variable part of the context. More experiments are needed to assess the effectiveness of our approach.

Another important observation is that the set of contextual symmetries may not monotonically increase with increasing context size. This may happen if additional context variables break existing symmetries, since context variables are forced to undergo identity mapping. Then, how do we design algorithms so that their effectiveness monotonically increases with larger contexts in all cases? This is an important direction for future work.

Another question concerns the robustness of performance of symmetry-based inference algorithms. Over the course of our experiments, we tested our algorithms on several domain variations. While in most cases CON-MCMC(0.01) and CON-MCMC(0) performed much better than Gibbs, in rare cases, the performance was worse too. Further investigations revealed two main sources of lower performance.

The first and more prominent cause is the trade-off between mixing speed and sampling time. Because all symmetry-based algorithms run an expensive product replacement algorithm [Pak, 2000] to sample from an orbit, next samples for CON-MCMC (and Orbital MCMC) are generated much slower than Gibbs. In domains where symmetries are prevalent, this slower sampling is mitigated by rapid mixing, but in other domains, it could result in a worse performance. An intelligent wrapper that guesses whether to exploit symmetries or not in a given domain will be crucial for developing a robust inference algorithm. The second reason for lower performance is subtle. CON-MCMC(α) is able to exploit contextual symmetries (even

single-sided) in a wide variety of settings, but in one situation it can lose to other algorithms. This happens when the context variable has a huge Markov blanket, so much so that one Gibbs move that flips the context variable becomes overbearingly costly. Since CON-MCMC(α) up-samples flips of context variables, this can cause significant loss to overall performance, even though the mixing is much faster with respect to number of samples.

Another observation relates to the effect of evidence in a domain. Evidence can both help and hurt symmetries in an inference problem. In some cases, evidence can break existing symmetries and reduce the relative gain of symmetry-based algorithms. In other cases, evidence can break edges and create new symmetries and help them. While in our experiments, we did not find CON-MCMC(0.01) to be ever worse than Gibbs due to additional evidence, such pathological cases can be constructed.

It would be interesting to see how algorithms other than MCMC can benefit from our contextual symmetry framework. In the future, we would also like to explore approximate contextual symmetries that could make our contribution applicable to several other domains, where exact contextual symmetries cannot be found. We would also like to theoretically analyze the mixing time of CON-MCMC.

3.5 Related Work

Some papers have discussed methods for computing symmetries under a given evidence [Van den Broeck and Darwiche, 2013; Venugopal and Gogate, 2014b; Kopp *et al.*, 2015]. As discussed in Section 3.1.2, the algorithm for computing contextual symmetries is closely related to computing evidence-based symmetries. The main difference is in the way we use these symmetries for downstream inference.

While our general notion of contextual symmetries is novel it has connections to a few recent works. The RockIt system [Noessner *et al.*, 2013] identifies contextual symmetries in a very special case in which the domain theory has a set of disjunctive clauses of a specific kind $g_i \vee c$ where each g_i is a single literal (or its negation). For this setting, c is a natural context and symmetries among g_is can be exploited. RockIt does not provide any general notion beyond this special case. It constructs a reduced ILP for MAP inference instead of marginal inference, as in our case.

There is recent work on exploring connections between the concept of exchangeability of random variables and tractability of probabilistic inference [Niepert and Van den Broeck, 2014]. Our contextual symmetries can be seen as a generalization of their conditional decomposability to conditional *partial* decomposability where the sufficient statistics are precisely the contextual orbits. Whereas Niepert and Van den Broeck [2014] primarily focus on developing the theory

for conditional decomposability, we propose and additionally connect this with the symmetries present in the structure of a graphical model. Further, unlike them we develop an algorithm to compute these conditional decompositions (contextual symmetries in our case) and show how they can be used in practice for efficient probabilistic inference.

Our contextual symmetries are also analogous to *conditional symmetries* in constraint satisfaction problems (CSPs) [Gent *et al.*, 2005; Walsh, 2006; Gent *et al.*, 2007]. CSP symmetries are called conditional if symmetry groups exist only in a sub-problem of the original CSP, i.e., in a CSP with one or more additional constraints. The CSP problem setting and their actual manifestation in algorithms are quite different from lifted inference, but their definition and use of conditional symmetries is in the same spirit as ours.

3.6 Conclusions

We present a novel framework for contextual symmetries in probabilistic graphical models. Contextual symmetries generalize and extend previous notions of variable symmetry. Given any context, we can efficiently compute these symmetries by reducing it to the problem of colored graph isomorphism. While our framework is independent of any inference algorithm, we illustrate its applicability by proposing CON-MCMC, an MCMC approach that exploits contextual symmetries. Our experiments on several domains validate the efficacy of CON-MCMC, where it outperforms existing state-of-the-art techniques for symmetry-based MCMC by wide margins. Finally, we have released a reference implementation of CON-MCMC for wider use by the research community.

Chapter 4

Variable-Value and Non-EquiCardinal Symmetries in PGMs

To the best of our knowledge, both variable and contextual symmetries consider a limited notion of symmetries, which we call *count symmetries*. A count symmetry in a Boolean-valued domain is a symmetry between two states where the total number of zeros and ones exactly match. An illustrative algorithm for Boolean-valued PGMs (which we build upon) is Orbital MCMC [Niepert, 2012]. It first uses graph isomorphism to compute symmetries and later uses these symmetries in an MCMC algorithm. Symmetries are represented via permutation groups in which variables interchange values to create other symmetric states. Notice, that if a state has k ones then any permutation of that state will also have k ones; this algorithm can only compute count symmetries.

Similarly, lifted inference algorithms for multi-valued PGMs (e.g., [Poole, 2003; Bui *et al.*, 2013]), only compute a weak extension of count symmetries for multi-valued domains – they allow symmetries only between those sets of variables that have the same domain. And, the count, i.e. the number of occurrences, of any value (from the domain) within this set of variables remains the same between two symmetric states.

In response, we develop extensions to existing frameworks to enable computation of *non-count symmetries* in which the count of a value between symmetric states can change. We can also compute a special form of non-count symmetries, *non-equicardinal symmetries* in multi-valued domains, in which two variables that have different domain sizes may be symmetric. Our key insight is the framework of symmetry groups over variable-value (VV) pairs, instead of just variables. It allows interchanging a specific value of a variable with a different value of a different variable.

Orbital MCMC suffices for downstream inference over most kinds of symmetries except non-equicardinal ones, for which a Metropolis Hastings extension is needed. Our new symme-

tries lead to substantial computational gains over Orbital MCMC and vanilla Gibbs Sampling, which does not exploit any symmetries.

Overall, in this chapter, we introduce the following concepts: We develop a novel framework for symmetries between variable-value (VV) pairs, which generalize existing notions of variable symmetries (Section 4.1). We develop an extension of this framework, which can also identify Non-Equicardinal (NEC) symmetries, i.e., among variables of different cardinalities (Section 4.2). We design a Metropolis Hastings version of Orbital MCMC called NEC-Orbital MCMC to exploit NEC symmetries (Section 5.2). We experimentally show that our proposed algorithms significantly outperform strong baseline algorithms (Section 4.4). We also release the code for wider use[1].

4.1 Variable-Value (VV) Symmetries

Existing work on variable as well as contextual symmetries (chapter 3) has defined symmetries in terms of variable permutations. We observe that these can only represent orbits in which all states have exactly the same count of 0s and 1s. The simple reason is that any variable permutation only permutes the values in a state and hence the total count of each value remains the same. We name such type of symmetries as *count symmetries*.

We now give a formal definition of count symmetries for a general multi-valued graphical model, since our work applies equally to both Boolean-valued as well as any other discrete valued domains.

Let $\mathcal{X} = \{X_1, X_2, \cdots, X_n\}$ denote a set of variables where each X_i takes values from a discrete valued domain D_i. A permutation θ of $\mathcal{X}$ is a *valid variable permutation* if it defines a mapping between variables having the same domain. Analogously, we define a *valid variable symmetry*. We will say that two domains D_i and D_j are *equicardinal* if $|D_i| = |D_j|$. We call such variables equicardinal variables.

Definition 4.1.1. *Given a set of variables $X \subseteq \mathcal{X}$ sharing the same domain D and a $v \in D$, $count_X(s, v)$ computes the number of variables in X taking the value v in state s.*

Definition 4.1.2. *Given a domain D, let X_D denote the subset of all the variables whose domain is D. A (valid) variable symmetry θ is a* count symmetry *if for each such subset $X_D \subseteq \mathcal{X}$, $count_{X_D}(s, v) = count_{X_D}(\theta(s), v)$, $\forall v \in D, \forall s \in \mathcal{S}$.*

Theorem 4.1.1. *For a graphical model $\mathcal{G}$, every (valid) variable symmetry θ is a count symmetry.*

[1] https://github.com/dair-iitd/nc-mcmc

Proof. Let $\mathcal{D}$ be the set of all domains (of variables) present in $\mathcal{G}$. Further, let $D \in \mathcal{D}$ be a domain and X_D denote all the variables having the domain D in $\mathcal{G}$. Let θ be a (valid) variable symmetry of a graphical model $\mathcal{G}$. Since, every (valid) variable symmetry defines a mapping between variables having the same domain. According to definition of (valid) variable symmetry, if $X_i \in X_D$ then, $\theta(X_i) \in X_D$. Let the value of X_i be v_i in s, then, the value of $\theta(X_i)$ is v_i in $\theta(s)$ (as per definition of $\theta(s)$). Hence, every value v_i present in s at X_i is also present in $\theta(s)$ at $\theta(X_i)$. Therefore, the count for every value $v \in D$ present in s for variables in X_D is same as count for v in $\theta(s)$ for variables in X_D. This holds for all domains $D \in \mathcal{D}$. Hence, $count_{X_D}(s, v) = count_{X_D}(\theta(s), v)$ for all $v \in D$, $D \in \mathcal{D}$. Hence, every (valid) variable symmetry is count symmetry. □ □

We argue here that count symmetries are restrictive; a lot more symmetry can be exploited if we simultaneously look at the *values* taken by the variables in a state. To illustrate this, consider a very simple graphical model $\mathcal{G}_3$ with the following two formulas: (a) w_1: $A \wedge \neg B$ (b) w_2: $\neg A \wedge B$. It is easy to see that there is no non-trivial symmetry here. The permutation $\theta(A) = B, \theta(B) = A$ results in a different graphical model since the two formulas have different weights. On the other hand, if we somehow could permute A with $\neg B$ and B with $\neg A$, we would get back the same model. In this section, we will formalize this extended notion of symmetry which we refer to as *variable-value symmetry* (VV symmetry in short).

Definition 4.1.3. *Given a set of variables $\mathcal{X} = \{X_1, \cdots, X_n\}$ where each X_i takes values from a domain D_i, a* variable-value *(VV) set is a set of pairs $\{(X_i, v_l^i)\}$ such that each variable X_i appears exactly once with each $v_l^i \in D_i$ in this set where v_l^i denotes the l^{th} value in D_i. We will use $\mathcal{S}_\mathcal{X}$ to denote the VV set corresponding to $\mathcal{X}$.*

For example, given a set $\mathcal{X} = \{A, B\}$ of Boolean variables, the VV set is given by: $\{(A, 0), (A, 1), (B, 0), (B, 1)\}$.

Definition 4.1.4. *A* Variable-Value permutation ϕ *over the VV set $\mathcal{S}_\mathcal{X}$ is a bijection from $\mathcal{S}_\mathcal{X}$ onto itself.*

Recall that a variable permutation applied to a state in a Boolean domain always results in a valid state. However, that may not be true in multi-valued domains, since if two variables that have different domains are permuted, it may not result in a valid state. It is also not true for all VV permutations. For example, given the state $[(A, 0), (B, 0)]$, a VV permutation defined as $\phi(A, 0) = (B, 1), \phi(A, 1) = (A, 1)$, $\phi(B, 0) = (B, 0), \phi(B, 1) = (A, 0)$ results in the state $[(B, 1), (B, 0)]$ which is inconsistent. Therefore, we need to impose a restriction on the set of allowed VV permutations so that they result in only valid states.

Definition 4.1.5. *We say that a VV permutation ϕ is a valid VV permutation if each variable $X_i \in \mathcal{X}$ maps to a unique variable X_j under ϕ. In other words, ϕ is valid if, whenever $\phi(X_i, v_l^i) = (X_j, v_{l'}^j)$ and $\phi(X_i, v_t^i) = (X_k, v_{t'}^k)$, then $X_j = X_k$, $\forall v_l^i, v_t^i \in D_i$. In such a scenario, we say that ϕ maps variable X_i to X_j.*

It is easy to see that for any valid VV permutation ϕ, applying ϕ on a state s always results in a valid state $\phi(s)$. It also follows that if such a ϕ maps a variable X_i to X_j, then D_i and D_j must be equicardinal.

Theorem 4.1.2. *The set of all valid VV permutations over $\mathcal{S}_\mathcal{X}$ forms a group.*

Proof. A group is defined by a set of elements and an operation defined over them. The operation usually considered in permutation groups is composition. The set of (unrestricted) permutations over any set is a group, called as permutation group [Wielandt, 2014]. We need to prove that the sub-set of permutation group to a valid VV permutations set is also a group or the set of VV permutations is a sub-group of permutation group over $\mathcal{S}_\mathcal{X}$. A non-empty subset of a group is a sub-group if it satisfies the following two properties i) Closure ii) Inverse element exists. We will prove that both these properties hold for valid VV permutations set.
i) **Closure**: For proving closure, we need to prove that if ϕ_1, ϕ_2 are valid VV permutations over S_X, then, $\phi_2(\phi_1)$ is also a valid VV permutation. Since, ϕ_1 and ϕ_2 are valid, let ϕ_1 maps a variable X_i to a unique variable X_j, and let, ϕ_2 maps variable X_j to a unique variable X_k. Then, $\phi_2(\phi_1)$ maps the variable X_i to a unique variable X_k (by the composition of permutations). This is true for all the variables. Hence, $\phi_2(\phi_1)$ is also a valid VV permutation and the set of valid VV permutations over $\mathcal{S}_\mathcal{X}$ is closed under composition.
ii) **Inverse Element**: For this property, we need to prove that if ϕ is a valid VV permutation, then, ϕ^{-1} is also a valid VV permutation. As per definition of inverse in a permutation group, if $\phi(X_i, v_i) = (X_j, v_j)$, then, $\phi^{-1}(X_j, v_j) = (X_i, v_i)$, $\forall X_i, X_j \in \mathcal{X}$, $v_i, v_j \in Domain(X_i)$. Since, ϕ is a valid VV permutation, it will map X_i uniquely to X_j. Therefore, ϕ^{-1} will map X_j uniquely to X_i (mapping all values in inverse direction). Hence, ϕ^{-1} is a valid VV permutation and satisfies inverse property.

The set of all valid VV permutations satisfy both the closure and inverse property. Hence, the set of all valid VV permutation is a sub group of permutation group over S_X and a group in itself. □ □

Consider a graphical model $\mathcal{G}$ specified as a set of pairs $\{f_j, w_j\}$. Each feature f_j can be thought of as a Boolean function over the variable assignments of the form $X_i = v_l^i$. Hence, action of a VV permutation ϕ on a feature f_j results in a new feature f_j' (with weight w_j) obtained by replacing the assignment $X_i = v_l^i$ by $X_j = v_{l'}^j$ in the underlying functional form of f_j where $\phi(X_i, v_l^i) = (X_j, v_{l'}^j)$. Hence, application of ϕ on a graphical model $\mathcal{G}$ results in

a new graphical model $\mathcal{G}'$ where each feature (w_j, f_j) is transformed through application of ϕ. We are now ready to define the symmetry of a graphical model under the application of VV permutations.

Definition 4.1.6. *We say that a (valid) VV permutation is a* VV symmetry *of a graphical model $\mathcal{G}$ if application of ϕ on $\mathcal{G}$ results back in $\mathcal{G}$ itself.*

All other definitions of the previous section follow analogously. We can define an automorphism group over VV permutations, and also define an orbit of a state under this permutation group. VV symmetries strictly generalize the notion of variable symmetries.

Theorem 4.1.3. *Each (valid) variable symmetry θ can be represented as a VV symmetry ϕ. There exist valid VV symmetries that cannot be represented as a variable symmetry.*

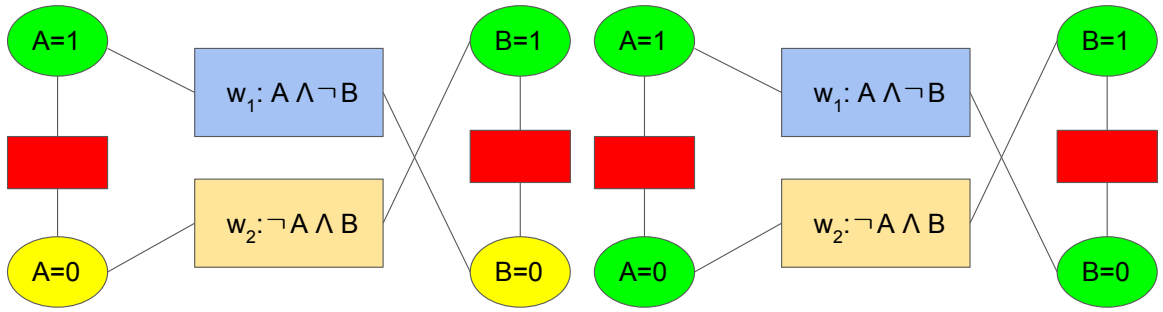
Proof. For proving the first part, recall that a variable permutation θ is *valid* if it always maps between variables that have exactly the same domain. Say, $\theta(X_i) = X'_i$ with both variables having domains D_i. It is easy to see that ϕ defined such that $\phi(X_i, v^i_l) = (X'_i, v^i_l)$ for all $v^i_l \in D_i$, will result in the same sets of symmetric states. Hence, every (valid) variable symmetry can be represented as a VV symmetry.

To prove the second part, consider a PGM $\mathcal{G}_4$ with two Boolean variables X_1 and X_2. Let there be four features $f_{00}, f_{01}, f_{10}, f_{11}$, one corresponding to each of the four states, with weights given as w_s, w_d, w_d, w_s, respectively. Then, we have a VV symmetry ϕ such that $\phi(X_1, 0) = (X_2, 1)$, $\phi(X_1, 1) = (X_2, 0)$, $\phi(X_2, 0) = (X_1, 1)$ and $\phi(X_2, 1) = (X_1, 0)$. Note that ϕ maps the state $[(X_1, 0), (X_2, 0)]$ to $[(X_1, 1), (X_2, 1)]$ and reverse, and similarly there is a symmetry ϕ' which maps $[(X_1, 0), (X_2, 1)]$ to $[(X_1, 1), (X_2, 0)]$ and reverse. There is no variable symmetry which can capture the symmetries induced by ϕ since counts are not preserved. This proves the theorem. But let us for a moment define a renaming of the form $X'_1 = \neg X_1$. Variable symmetries will now be able to capture the symmetries due to ϕ but will miss out on the ones due to ϕ'. This is illustrative because there is no single problem formulation which can capture both the state symmetries above using the notion of variable symmetries alone. □

□

To prove that VV symmetries preserve joint probability, we prove the lemma which captures relationship between feature and a state s to transformed feature and transformed state as done for Variable and contextual symmetries.

Lemma 4.1.1. *If ϕ is a VV symmetry of a graphical model $\mathcal{G}$, then, then, the transformed feature $f'_j = \phi(f_j)$ holds true in transformed state $s' = \phi(s)$ if and only if f_j holds true in state s*



(a) (b)

Figure 4.1: (a) Variable Symmetry Graph for toy example $\mathcal{G}_3$ (b) VV-Symmetry Graph for $\mathcal{G}_3$

Proof. As usual, without loss of generalization, we assume that, features are expressed as disjunction of variable-value pairs. If f_j holds true in state s, then, $\exists$ a variable-value pair (X_i, v_i), which is present in feature f_j and $s(X_i) = v_i$. If $\phi(X_i, v_i) = X_j, v_j$, then, (X_j, v_j) is present in f'_j. Similarly, by definition of $\phi(s)$, (X_i, v_i) pair is replaced by (X_j, v_j) pair and hence, $s'(X_j) = v_j$. Hence, f'_j holds true in state s' . The other side is true, by considering another VV-symmetry $\phi^{-}1$ for these two features and applying the same argument. □

Theorem 4.1.4. *VV symmetry permutation group results in a weight signature preserving state partition and hence, VV symmetries also preserve joint probabilities. Specifically, given any VV symmetry ϕ, state s, let $W(s)$ denotes the weight signature of state s, then, we have $W(s) = W(\phi(s))$ and $P(s) = P(\phi(s))$.*

Proof. The proof to this theorem is exactly the same as for variable and contextual symmetry (theorem 2.4.2) provided lemma 4.1.1 holds true. Since, we have proved the lemma, the weight-signature is preserved. Then, if weight-signature of two states s and s' are same, then, probabilities of those two state are equal by theorem 2.4.2.

□

4.1.1 Computing Variable-Value Symmetries

We now adapt the procedure in Section 2.2.2 to compute VV symmetries in multi-valued domains. For a PGM $\mathcal{G}$ with clausal theory or conjunctive theory (as before), we construct a colored graph $G_{VV} = T'(\mathcal{G})$ with a node for each variable-value pair. We also have a node for each feature, which is connected to the specific VV nodes it contains. We need to additionally impose a mutual exclusivity constraint to assert that a variable can only take exactly one of its many values. This is accomplished by adding exactly-one features with ∞ weight between all values of each variable. When assigning colors to each node, we assign all values of any

variable the same color, as opposed to different values getting different colors. This allows the isomorphism solver to attempt discovering symmetries between different value nodes. As before, all features with the same weight get the same color. Figure 4.1 illustrates this on $\mathcal{G}_1$ where Variable symmetry assigns different colors to 0 and 1 while VV-Symmetry assigns a single color (green) to both 0 and 1 assignments of all variables.

We run Saucy [Darga *et al.*, 2008] over $G_{VV}(\mathcal{G})$ to compute its automorphism group via a set of permutations. These permutations are valid VV-permutations (by construction of G_VV), and, collectively, represent a VV automorphism group of $\mathcal{G}$.

Theorem 4.1.5. *Any permutation ϕ that preserves graph isomorphism in $G_{VV} = T'(\mathcal{G})$ is a valid VV-permutation for $\mathcal{G}$.*

Proof. Any permutation ϕ that preserves graph automorphism in the graph G_{VV} creates a mapping between features such that same weight features are mapped to same weight features, i.e. graphical model is preserved. Further, the feature having infinity weight (mutual exclusiveness) between variables ensure that all values of a variable are mapped to all values of another variables. This is similar to valid VV permutation. Hence, restricting the phi obtained from graph automorphism to variable-value pairs results in a valid VV permutation which preserves the original graphical model.

□

Theorem 4.1.6. *The automorphism group of colored graph $G_{VV} = T'(\mathcal{G})$ constructed above computes a VV-automorphism group of graphical model $\mathcal{G}$.*

Proof. As per theorem 4.1.1, every permutation in G_{VV} gives rise to an analogous valid VV permutation. Since the permutations of G_{VV} form a group, it also gives rise to an analogous a VV-automorphism group of graphical model $\mathcal{G}$ (since the same set of permutations are obtained in two cases).

□

4.2 Non-Equicardinal (NEC) Symmetries

While VV symmetries can compute many new symmetries compared to variable symmetries, they only consider mapping between equicardinal variables. In this section, we will deal with symmetries which can be present across variables having different domain sizes. Consider the following example graphical model $\mathcal{G}_5$ with two features: (1)w: $A = 1$ (2) w: $B = 1 \vee B = 2$. Let A and B have the domains D_A and D_B, respectively, specified as $D_A = \{0, 1\}$ and $D_B = \{0, 1, 2\}$. Clearly, there is no VV symmetry between A and B since they have different domain

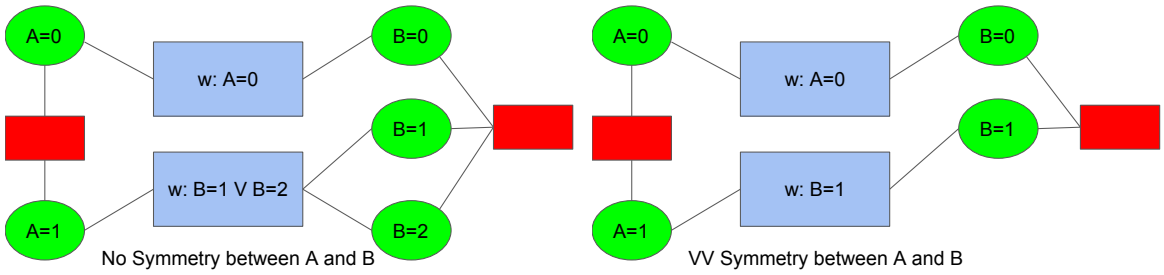


Figure 4.2: (a)Unreduced multi-valued domain $\mathcal{G}_5$ (b) Reduced multi-valued domain $\mathcal{G}_5$

sizes. But intuitively, the two states given as $[(A,1),(B,0)]$ and $[(A,0),(B,1)]$ are symmetric to each other since in each case, exactly one of the two features having the same weight is satisfied. Similarly, for $[(A,1),(B,0)]$ and $[(A,0),(B,2)]$. Further, it is easy to see that the two values of $B=1$ and $B=2$ are symmetric to each other in the sense states of the form $[(A,v),(B,1)]$ have the same probability as the states $[(A,v),(B,2)]$ where $v \in \{0,1\}$.

We will combine the above two ideas together to exploit symmetries using domain reduction. We first identify all the equivalent values of each variable and *replace* them by a single representative value. In this reduced graphical model, we then identify VV symmetries and finally translate them back to the original graphical model. In the following, we will assume that we are given a graphical model $\mathcal{G}$ defined over a set of n variables $\mathcal{X}$ where each $X_i \in \mathcal{X}$ takes values from a domain D_i. Further, we will use the symbol $\mathcal{D} = D_1 \times D_2, \cdots D_n$ to denote the cross product of the domains.

Definition 4.2.1. *Consider a variable $X_i \in \mathcal{X}$ and let $v, v' \in D_i$. Let $\phi^i_{v \leftrightarrow v'}$ denote a VV permutation which maps the VV pair (X_i, v) to (X_i, v') and back. For all the remaining VV pairs (X_k, v''), $\phi^i_{v \leftrightarrow v'}$ maps the pair back to itself. We refer to $\phi^i_{v \leftrightarrow v'}$ as a* value swap permutation *for variable X_i.*

In the example above, $\phi^B_{1 \leftrightarrow 2}$ is a value swap permutation for B which permutes the variable assignments $B=1$ and $B=2$, and keeps the remaining variable assignments, i.e., $B=0$ and $A=1$, fixed.

Definition 4.2.2. *A value swap permutation $\phi^i_{v \leftrightarrow v'}$ is a* a value swap symmetry *of $\mathcal{G}$ if it maps $\mathcal{G}$ back to itself.*

In our running example, $\phi^B_{1 \leftrightarrow 2}$ is a value swap symmetry of $\mathcal{G}_6$. Next, we show that the set of all value swap symmetries corresponding to a variable X_i divides its domain D_i into equivalence classes.

Definition 4.2.3. *Given a graphical model $\mathcal{G}$, we define a relation SS_i (swap symmetry) over the set $D_i \times D_i$ as follows. Given $v, v' \in D_i$, $(v, v') \in SS_i$ if $\phi^i_{v \leftrightarrow v'}$ is a value swap symmetry of $\mathcal{G}$.*

It is easy to see that relation SS_i is an equivalence relation and hence, partitions the domain D_i into a set of equivalence classes. Given a value $v \in D_i$, we choose a representative value from its equivalence class based on some canonical ordering. We denote this value by $rep_i(v)$.

Next, we will define a reduced domain D_i^R obtained by considering one value from each equivalence set.

Definition 4.2.4. *Let SS_i divide the domain D_i into r equivalence classes. We define the reduced domain D_i^R as the r-sized set $\{v_j^*\}_{j=1}^r$ where v_j^* is the representative value for the j^{th} equivalence class. We will use $\mathcal{D}^R = D_1^R \times D_2^R \times \cdots D_n^R$ to denote the cross product of the reduced domains.*

Revisiting our example, the reduced domain for B is given as $D_B^R = \{0, 1\}$. Next we define a reduced graphical model $\mathcal{G}^R$ over the reduced set of domains $\{D_i^R\}_{i=1}^n$.

Definition 4.2.5. *Let $\mathcal{G}$ be a graphical model with the set of weighted features $\{w_j, f_j\}$. Let $X_i = v$ be a variable assignment appearing in the Boolean expression for f_j. We construct a new feature f_j' by replacing every such expression $X_i = v$ by $false$ (and further simplifying the expression) whenever $v \neq rep_i(v)$. If $v = rep_i(v)$, then we leave the assignment $X_i = v$ in f_j' as is. The reduced $\mathcal{G}^R$ is the graphical model having the set of features $\{w_j, f_j'\}$ defined over the set of variables $\mathcal{X}$ with X_i having the domain D_i^R.*

Intuitively, in $\mathcal{G}^R$, we restrict each variable X_i to take only the representative value from each of its equivalence classes. In our running example, the reduced graphical model is given as $\{w\colon\ A = 0;\ w\colon\ (B = 1) \vee false\}$ which is same as $\{w\colon\ A = 0;\ w\colon\ B = 1\}$. Since the domains have been reduced in $\mathcal{G}^R$, we may now be able to discover mappings which were not possible earlier. For instance in our running example, we now have a VV symmetry ϕ^R which maps $(A, 0)$ to $(B, 1)$ and back.

Let the joint distributions specified by $\mathcal{G}$ and $\mathcal{G}^R$ be given by P_G and P_{G^R}, respectively. The next theorem describes the relationship between these two distributions.

Theorem 4.2.1. *Let $\mathcal{G}$ be a graphical model and let $\mathcal{G}^R$ be the corresponding reduced graphical model. Consider a state s specified as $\{X_i, v_i\}_{i=1}^n$ where each $v_i \in D_i^R$. By definition, $v_i \in D_i$. We claim that $P_{\mathcal{G}^R}(s) = k * P_{\mathcal{G}}(s)$ where k is some constant $k \geq 1$ independent of the specific state s.*

Proof. Note that the reduced graphical model G^R is emulating the distribution specified by $\mathcal{G}$ where the space of possible variable assignments is now restricted to those belonging to the representative set, i.e., for each variable X_i the allowed set of values is now $D_i^R = \{v_i | v_i = rep_i(v), v \in D_i\}$. Therefore, G^R can be thought of as enforcing a conditional distribution over the underlying space given the fact that assignments can now only come from cross product set

$\mathcal{D}^R$. Recall that state s is valid assignment in the original as well as the reduced graphical model. Therefore, we have $P_{\mathcal{G}^R}(s) = P_{\mathcal{G}}(s|s \in \mathcal{D}^R) = P_{\mathcal{G}}(s)/P_{\mathcal{G}}(s \in \mathcal{D}^R)$. Here, the denominator term $P_{\mathcal{G}}(s \in \mathcal{D}^R)$ is simply the probability that a randomly chosen state s in the original distribution belongs to the restricted domain set. Clearly, this is independent of the state s and let this given as $1/k$, where $k \geq 1$ is a constant independent of s. Then, $P_{\mathcal{G}^R}(s) = k*P_{\mathcal{G}}(s)$. □

Above theorem gives us a recipe to discover additional symmetries across variables having different domain sizes. Let $s = \{X_i, v_i\}_{i=1}^n$ be a state in $\mathcal{G}$. Let $rep(s)$ denote the representative state for s given as $\{(X_i, rep_i(v_i)\}_{i=1}^n$).Following steps describe a procedure to get a new state s' symmetric to s using the idea of domain reduction.

Procedure NonEquiCardinalSym:

- Let $u = rep(s)$ denote the representative state for s.
- Apply a VV symmetry $\phi^R(u)$ over u in the reduced graphical model. Resulting state u' is symmetric to u in $\mathcal{G}^R$.
- Apply a series of n value swap symmetries of the form $\phi^i_{v'_i \leftrightarrow v''_i}$ over state u', one for each variable X_i such that $X_i = v'_i$ in u', $v''_i \in D_i$. Resulting state s' is symmetric to s in $\mathcal{G}$.

Definition 4.2.6. *Let τ be a permutation over the state space $\mathcal{S}$ of $\mathcal{G}$ defined using the Procedure NonEquiCardinalSym, i.e., $\tau(s) = \phi^n_{v'_n \leftrightarrow v''_n}(\phi^{n-1}_{v_{n-1} \leftrightarrow v''_{n-1}}(\cdots \phi^1_{v'_1 \leftrightarrow v''_1}(\phi^R(rep(s)))\cdots))$, where ϕ^R is a VV symmetry of $\mathcal{G}^R$ and each $\phi^i_{v'_i \leftrightarrow v''_i}$ is a value swap symmetry for variable X_i in $\mathcal{G}$. We refer to τ as a non-equicardinal symmetry of $\mathcal{G}$.*

Unlike VV symmetries whose action is defined over a VV pair, non-equicardinal symmetries directly operate over the state space. Their transformation of the underlying graphical model is implicit in the symmetries that compose them. Finally, we need to show that action of non-equicardinal symmetries indeed results in states which have the same probability.

Theorem 4.2.2. *Let τ be a non-equicardinal symmetry of a graphical model $\mathcal{G}$. Then, $P_{\mathcal{G}}(s) = P_{\mathcal{G}}(\tau(s))$.*

Proof. Let $s' = \tau(s)$. Let $u = rep(s)$. Since u' is obtained by application of VV symmetry $\phi^R(u)$ in $\mathcal{G}^R$, we have $P^R_{\mathcal{G}}(u) = P^R_{\mathcal{G}}(u')$. Using Theorem 4.2.1, this implies that $P_{\mathcal{G}}(u) = (1/k) * P_{\mathcal{G}^R}(u) = (1/k) * P_{\mathcal{G}^R}(u') = P_{\mathcal{G}}(u')$ for some constant k. Hence, u and u' have the same probability under $P_{\mathcal{G}}$.

Since $u = rep(s)$ can be obtained by application of n value swap symmetries over s (one for each variable), $P_{\mathcal{G}}(s) = P_{\mathcal{G}}(u)$. Similarly, since s' is obtained by an application of n value swap symmetries over u', we have $P_{\mathcal{G}}(u') = P_{\mathcal{G}}(s)$. Combining this with the fact that, $P_{\mathcal{G}}(u) = P_{\mathcal{G}}(u')$, we get $P_{\mathcal{G}}(s) = P_{\mathcal{G}}(s')$. □

4.2.1 Computing Non-Equicardinal Symmetries

We adapt the procedure in Section 4.1.1 by running graph isomorphism over a series of two colored graphs. Our first colored graph is constructed as in Section 4.1.1, except that all features are given different colors. This disallows any mapping between (X_i, v_i) and (X_j, v_j) for $X_i \neq X_j$, and only allows mapping between different values of a single variable. For example, in the running example, this would determine that $(B, 1)$ and $(B, 2)$ are symmetric. We then retain only the representative value for each equivalent set of VV pairs, and removes nodes and edges for other values.

We take this reduced colored graph and recolor all mutual exclusivity features with a single color. We run graph isomorphism again to obtain the VV symmetries of the reduced model. These permutations together with the single-variable permutations from the previous step gives the non-equicardinal symmetries of the original model.

4.3 MCMC with VV & NEC Symmetries

Recall from Section 2.2.3 that variable symmetries are used in approximate inference via the Orbital MCMC algorithm. It alternates original MCMC move with a symmetry move, which uniformly samples from the orbit of the current state. We first observe that the same algorithm will work for VV symmetries computed in Section 4.1.1, except that the orbital move will now sample from the orbit induced by VV permutations – we call this algorithm VV-Orbital MCMC.

We now consider the case of non-equicardinal symmetries in multi-valued PGMs. The main idea from Orbital MCMC remains valid – we need to alternate between original chain and symmetry move. However, sampling a random state from an orbit is tricky now, because a non-equicardinal orbit may have a two-level hierarchical structure – it is an orbit over suborbits. The top level orbit is in the reduced model and is an orbit over representative states. At the bottom level, each representative state may represent multiple states via application of a *variable* number of value-swap symmetries.

As an example, consider the state partition in our running example, as illustrated in Figure 4.3. Each orbit is shown by a unique color, and suborbits by large ovals. The green orbit (top level) has two representative states (0,0) and (1,1) in the reduced model. If we make an orbital move in the reduced model, we can easily pick a representative state uniformly at random. However, the state (1,1) has a suborbit – it further represents two states in the original model, (1,1) and (1,2), via value-swap symmetries on variable b. Our sampling goal is to pick uniformly at random from an orbit in the *original* model, which means we need to pick a representative state in the reduced model proportional to the size of suborbit it represents. Once a suborbit is picked, we can easily pick a state uniformly at random from within it. To pick a representative

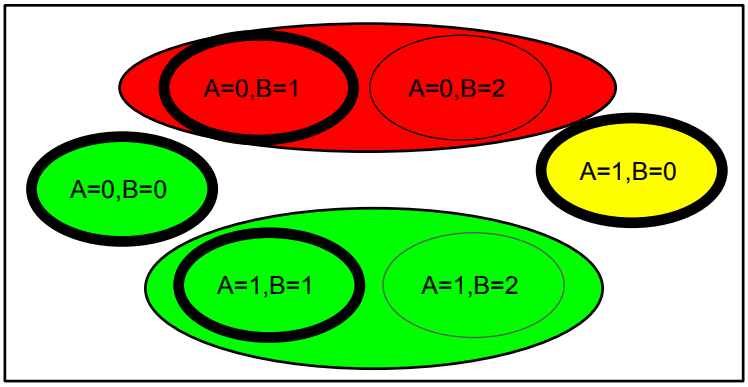


Figure 4.3: State Partition for Toy Example $\mathcal{G}_5$. Same Colored States are in same orbit. Large Ovals show sub-orbits and representative states of sub-orbits are with dark outline.

proportional to the size of the suborbit, we use Metropolis Hastings in the reduced model – we name the resulting algorithm NEC-Orbital MCMC.

Let $c(s)$ represent the cardinality of the suborbit of state s, i.e., the number of states for which the representative state is the same as that of s: $|\{s'|rep(s') = rep(s)\}|$. Let $c^i(s)$ represent the number of states in the orbit of s which differ from s at most on the value of X_i , i.e., $|\{s'|rep(s') = rep(s), s.X_j = s'.X_j \forall j \neq i\}|$, where $s.X_j$ represents the value of X_j in s.

Given a Markov chain $\mathcal{M}$ over a graphical model $\mathcal{G}$, a sample from s_t to s_{t+1} in NEC-Orbital MCMC is generated:

- Generate s'_t by sampling from transition distribution of $\mathcal{M}$ starting from s_t.
- Let $u'_t = rep(s'_t)$. Sample u''_t (in $\mathcal{G}^R$) from the orbit $\Gamma_{\phi^R}(u'_t)$ via a Metropolis Hastings step using the uniform proposal distribution $q(\cdot) = \frac{1}{|\Gamma_{\Phi^R}(u'_t)|}$, and desired distribution $p(\cdot) \propto c(u''_t)$.
- Apply a series of n value swap symmetries of the form $\phi^i_{v''^i_t \leftrightarrow v^i_{t+1}}$ over state u''_t, one for each variable X_i, where $u''_t.X_i = v''^i_t$, and v^i_{t+1} is chosen uniformly at random from the set of values equivalent with v''^i_t with probability $\frac{1}{c^i(u''_t)}$. This is equivalent to sampling uniformly from the suborbit of u''_t.

Notice that sampling from the proposal distribution (uniform) from an orbit is easily accomplished by Product Replacement Algorithm [Pak, 2000]. MH accepts or rejects the sample with an Acceptance probability A, which can be computed by MH's detailed balance equation:

$$\begin{aligned} A(u'_t \rightarrow u''_t) &= min\left(1, \frac{p(u''_t) * q(u'_t|u''_t)}{p(u'_t) * q(u''_t|u'_t)}\right) \\ &= min\left(1, \frac{p(u''_t)}{p(u'_t)}\right) = min\left(1, \frac{c(u''_t)}{c(u'_t)}\right) \end{aligned}$$

The second equality above follows form the fact that $q(.)$ is a uniform proposal.

Let the transition probability from s_t to s_{t+1} in NEC-Orbital MCMC be given by $T^{NEC}(s_t \to s_{t+1})$. And, let $T_2(s'_t \to s_{t+1})$ denote the transition probability because of the last 2 steps given in Section-5. Then, we can write T^{NEC} as:

$$T^{NEC}(s_t \to s_{t+1}) = \sum_{s'_t \in \Gamma_\Phi(s_{t+1})} T^M(s_t \to s'_t) \cdot T_2(s'_t \to s_{t+1})$$

where $\Gamma_\Phi(s_{t+1})$ denotes the orbit of s_{t+1} with respect to the original domain

Obeying the notation in Section 5, $u'_t = rep(s'_t)$ and $u''_t = rep(s_{t+1})$. Also, since the last step (given in Section 5) is equivalent to sampling s_{t+1} uniformly at random from the sub-orbit u''_t, and, if we represent T^{MH} as transition probability for the second step, we get:

$$\begin{aligned} T_2(s'_t \to s_{t+1}) &= T^{MH}(u'_t \to u''_t) \cdot P(s_{t+1}|u''_t) \\ &= T^{MH}(u'_t \to u''_t) \cdot \frac{1}{c(u''_t)} \end{aligned}$$

Therefore, we get:

$$T_2(s'_t \to s_{t+1}) = \begin{cases} \left(\frac{1}{|\Gamma_{\Phi^R}(u'_t)|} \cdot A(u'_t \to u''_t) \right) \cdot \frac{1}{c(u''_t)}, & \text{if } u'_t \neq u''_t \\ \left(\frac{1}{|\Gamma_{\Phi^R}(u'_t)|} + \sum_{z \in \Gamma_{\Phi^R}(u'_t)} \frac{1}{|\Gamma_{\Phi^R}(u'_t)|} \cdot (1 - A(u'_t \to z)) \right) \cdot \frac{1}{c(u''_t)}, & \text{if } u'_t = u''_t \end{cases}$$

Having defined T_2, concretely, we show that it obeys detailed balance.

Lemma 4.3.1. *$\forall s_1 \forall s_2$ such that $u_1 = rep(s_1), u_2 = rep(s_2)$ and $\Gamma_{\Phi^R}(u_1) = \Gamma_{\Phi^R}(u_2)$, we have:*

$$T_2(s_1 \to s_2) = T_2(s_2 \to s_1)$$

This lemma basically shows that T_2 obeys detailed balance with respect to the uniform distribution over states in the (original) orbit of s_1 and s_2.

Proof. **Case 1:** $u_1 \neq u_2$

$$\begin{aligned} T_2(s_1 \to s_2) &= \left(\frac{1}{|\Gamma_{\Phi^R}(u_1)|} \cdot A(u_1 \to u_2)\right) \cdot \frac{1}{c(u_2)} \\ &= \frac{1}{|\Gamma_{\Phi^R}(u_1)|} \cdot min\left(1, \frac{c(u_2)}{c(u_1)}\right) \cdot \frac{1}{c(u_2)} \quad = \frac{1}{|\Gamma_{\Phi^R}(u_1)|} \cdot min\left(\frac{1}{c(u_2)}, \frac{1}{c(u_1)}\right) \\ &= \frac{1}{|\Gamma_{\Phi^R}(u_1)|} \cdot min\left(\frac{c(u_1)}{c(u_2)}, 1\right) \cdot \frac{1}{c(u_1)} \\ &= \left(\frac{1}{|\Gamma_{\Phi^R}(u_1)|} \cdot A(u_2 \to u_1)\right) \cdot \frac{1}{c(u_1)} \\ &= T_2(s_2 \to s_1) \end{aligned}$$

Case 2: $u_1 = u_2$

It trivially holds for this case because the expression of T_2 does not depend on the states s_1 and s_2 but just their sub-orbit (which is the same in this case).

Now, let π denote the unique stationary distribution of the Markov Chain $\mathcal{M}$. We show that T^{NEC} also has π as its stationary distribution, i.e. for any state $s \in \mathcal{S}$, we show that:

$$\pi(s) = \sum_{s_0 \in S} \pi(s_0) \cdot T^{NEC}(s_0 \to s)$$

□

Having shown that, T_2 obeys detailed balance, we show that NEC-Orbital Markov Chain converges to the same stationary distribution as original distribution.

Theorem 4.3.1. *The Markov Chain constructed by NEC-Orbital MCMC converges to the unique stationary distribution of original markov chain $\mathcal{M}$.*

Proof. Now, let π denote the unique stationary distribution of the Markov Chain $\mathcal{M}$. We show that T^{NEC} also has π as its stationary distribution, i.e. for any state $s \in \mathcal{S}$, we show that:

$$\pi(s) = \sum_{s_0 \in S} \pi(s_0) \cdot T^{NEC}(s_0 \to s)$$

Starting from RHS,

$$= \sum_{s_0 \in \mathcal{S}} \pi(s_0) \cdot T^{NEC}(s_0 \to s) \tag{4.1}$$

$$= \sum_{s_0 \in \mathcal{S}} \pi(s_0) \cdot \left[\sum_{s'_0 \in \Gamma_\Phi(s)} T^M(s_0 \to s'_0) \cdot T_2(s'_0 \to s) \right] \tag{4.2}$$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{s'_0 \in \Gamma_\Phi(s)} \pi(s_0) \cdot T^M(s_0 \to s'_0) \cdot T_2(s'_0 \to s) \tag{4.3}$$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{s'_0 \in \Gamma_\Phi(s)} \pi(s'_0) \cdot T^M(s'_0 \to s_0) \cdot T_2(s'_0 \to s); \text{ (detailed balance of } T^M) \tag{4.4}$$

$$= \sum_{s_0 \in \mathcal{S}} \sum_{s'_0 \in \Gamma_\Phi(s)} \pi(s) \cdot T^M(s'_0 \to s_0) \cdot T_2(s'_0 \to s); \; s'_0 \text{ and } s \text{ are in same orbit}) \tag{4.5}$$

$$= \pi(s) \cdot \sum_{s_0 \in \mathcal{S}} \sum_{s'_0 \in \Gamma_\Phi(s)} T^M(s'_0 \to s_0) \cdot T_2(s'_0 \to s) \tag{4.6}$$

$$= \pi(s) \cdot \sum_{s'_0 \in \Gamma_\Phi(s)} \left[\sum_{s_0 \in \mathcal{S}} T^M(s'_0 \to s_0) \right] \cdot T_2(s'_0 \to s) \tag{4.7}$$

$$= \pi(s) \cdot \sum_{s'_0 \in \Gamma_\Phi(s)} T_2(s'_0 \to s) \tag{4.8}$$

$$= \pi(s) \cdot \sum_{s'_0 \in \Gamma_\Phi(s)} T_2(s \to s'_0); \text{ because of the lemma} \tag{4.9}$$

$$= \pi(s) \cdot 1 \tag{4.10}$$

$$= LHS \tag{4.11}$$

$$\tag{4.12}$$

Hence, proved. □

4.4 Experiments

We empirically evaluate our extensions of Orbital MCMC for both Boolean and multi-valued PGMs. In both settings, we compare against the baselines of vanilla MCMC, and Orbital MCMC [Niepert, 2012]. In all orbital algorithms including ours, the base Markov chain $\mathcal{M}$ is set to Gibbs. We build our source code on existing code of Orbital MCMC.[2] It uses the Group Theory package Gap [GAP, 2015] for implementing the group-theoretic operations in the algorithms. We release our implementations for further research. [3] All our experiments

[2] https://code.google.com/archive/p/lifted-mcmc/

[3] Available at *https://github.com/dair-iitd/nc-mcmc*

are performed on Intel core i-7 machine. All our reported times include the time taken for computing symmetries.

Our experiments are aimed to assess the comparative value of our algorithms against baselines in those domains where a large number of symmetries (beyond count symmetries) are present. To this end, we construct two such domains. The first is a simple Boolean domain that shows how simple value renaming can affect baseline algorithms. The second is a multi-valued domain showcasing the potential benefits of non-equicardinal symmetries. The domains are:

Value-Renamed Ring Message Passing Domain: In this simple domain, N people with equal number of males and females are placed in a ring structure alternately with every male followed by a female, and they pass a bit of message to their immediate neighbor over a noisy channel. If X_i denoted the bit received by the i^{th} person, then we would have a formula for PGM $X_i \rightarrow X_{i+1}$ with weight w_1 if i is a male and weight w_2 if i is female. As a small modification to this domain, we randomly rename some X_is to mean ¬bit received by that agent, and change all formulas analogously. All the symmetries in the original ring should remain after this renaming. Our experiments test the degree to which the various algorithms are able to identify these.

Student-Curriculum Domain: In this multi-valued domain, there are K students taking courses from $|A|$ areas (e.g., theory, architecture, etc.). Each area $a \in A$ has a variable number of $N(a)$ courses numbered 1 to $N(a)$. Each student has to fulfill their breadth requirements by passing one course each from any two areas. A student has no specific preference to which of the $N(a)$ courses they take in an area. However, each student has a prior seriousness level, which determines whether they will pass *any* course. This scenario is modeled by defining a random variable P_{sa}, which is a multi-valued variable where value 0 denotes that student s failed the course in the area a, and value $i \in \{1 : N(a)\}$ denotes which course they passed. The weight for failing depends on the student but not on area. Finally, the variable $C_{saa'}$ denotes that s completed their requirements by passing courses from areas a and a'.

The Curriculum domain is interesting, because, for a given s, various values of P_{sa} other than 0 are all symmetric for all areas. And once, all P_{sa}s are converted to a representative value in the reduced model, all areas become symmetric for a student.

We compare different algorithms by plotting the KL-divergence of true marginals and an algorithm's marginals with time. True marginals are calculated by running Gibbs sampling for a sufficiently large duration of time. Figure 4.5 compares VV-Orbital MCMC with baselines on the message passing domain. The dramatic speedups obtained by VV-Orbital MCMC underscores Orbital MCMC's inability to identify the huge number of variable-renamed symmetries present in this domain, whereas VV-Orbital MCMC is able to benefit from these tremendously.

Before describing results on Curriculum domain, we first highlight that, out of the box, Orbital MCMC cannot run on this domain, because both its theory and implementation have

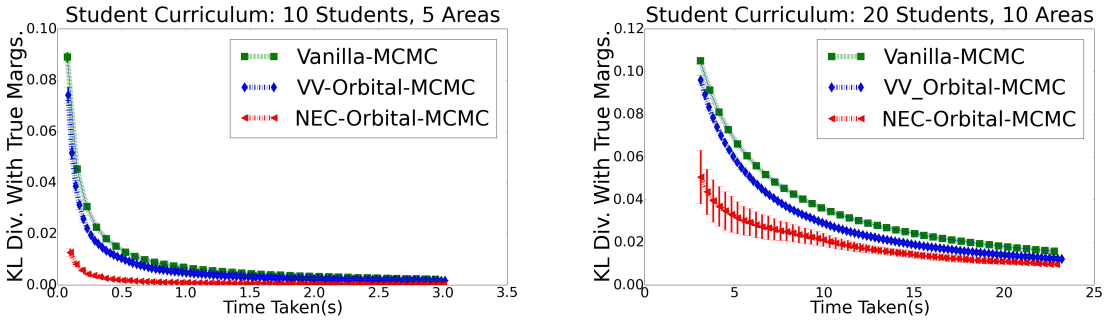


Figure 4.4: NEC-Orbital MCMC outperforms VV-Orbital MCMC and Vanilla-MCMC on student-curriculum domain.

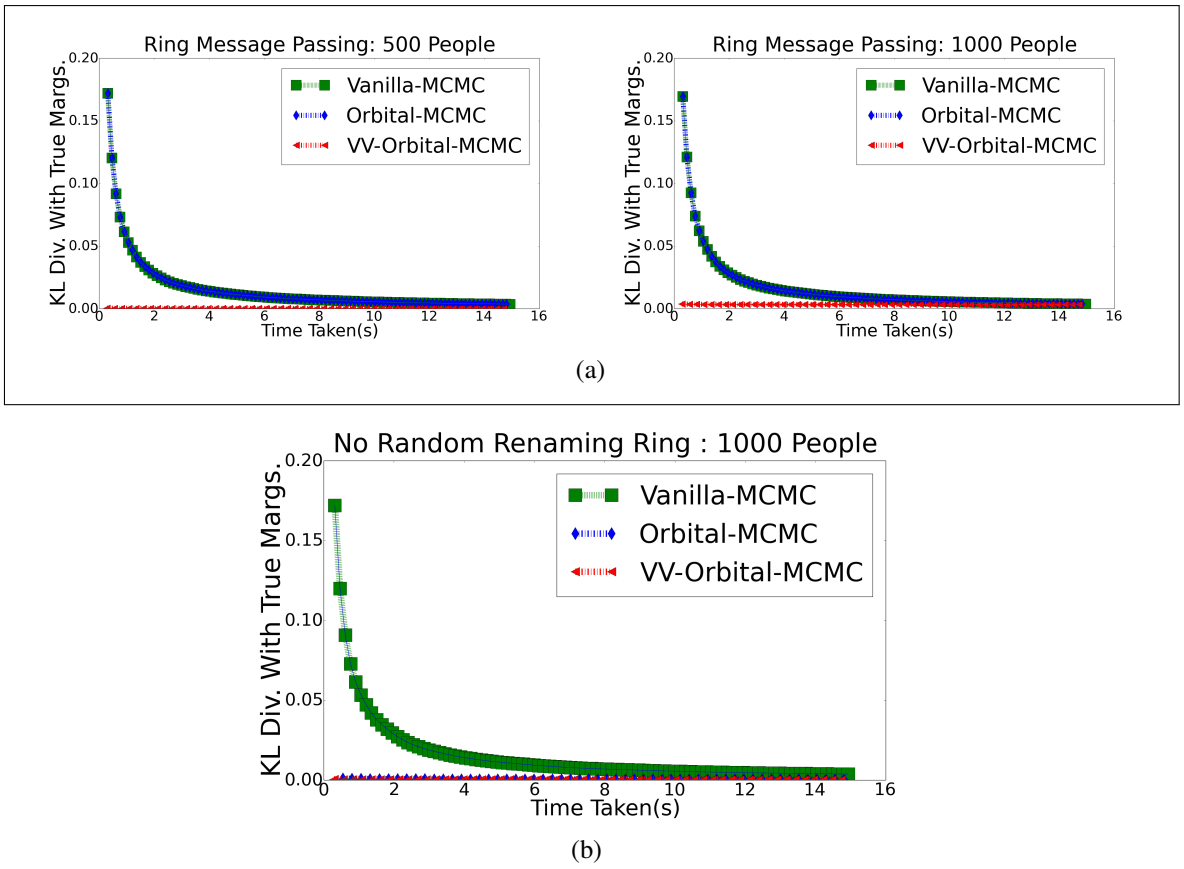


Figure 4.5: a)VV-Orbital-MCMC outperforms Orbital MCMC and Vanilla MCMC with different sizes of people on ring-message passing. b) VV-Orbital MCMC has negligible overhead compared to Orbital-MCMC

only been developed for Boolean-valued PGMs. To meaningfully compare against Orbital MCMC, we first *binarize* the domain, by converting each multi-valued random variable P_{sa} into many Boolean variables P_{sac}, one for each value c. We need to add an infinite-weighted exactly-one constraint for each original variable before giving it to Orbital MCMC. A careful reader may observe that this binarization is already very similar to the VV construction of Section 4.1, but without non-equicardinal symmetries. Thus, this is already a much stronger baseline than currently found in literature.

Figure 4.4 shows the results on this domain. NEC-Orbital MCMC outperforms both baselines by wide margins. Orbital MCMC does improve upon vanilla Gibbs since it is able to find that all P_{sac}s for different cs are equivalent, however, it is unable to combine them across areas.

In domains where symmetries beyond count symmetries are not found, the overhead of our algorithms is not significant, and they perform almost as well as (binarized) Orbital MCMC (e.g., see Figure 4.5(b)). This is also corroborated by the fact that the time for finding symmetries is relatively small compared to the time taken for actual inference on both the domains. Specifically, this time is 0.250 sec and 0.009 sec for curriculum and ring domains, respectively.

In summary, both VV-Orbital MCMC and NEC-Orbital MCMC are useful advances over Orbital MCMC.

4.5 Conclusion and Future Directions

Existing lifted inference algorithms capture only a restricted set of symmetries, which we define as *count symmetries*. To the best of our knowledge, this is the first work that computes symmetries beyond count symmetries. To compute these *non-count symmetries*, we introduce the idea of computation over variable-value (VV) pairs. We develop a theory of VV automorphism groups, and provide an algorithm to compute these. These can compute equicardinal non-count symmetries, i.e., between variables that have the same cardinality. An extension to this allows us to also compute *non-equicardinal symmetries*. Finally, we provide MCMC procedures for using these computed symmetries for approximate inference. In particular, the algorithm to use non-equicardinal symmetries requires a novel Metropolis Hastings extension to existing Orbital MCMC. Experiments on two domains illustrate that exploiting these additional symmetries can provide a huge boost to convergence of MCMC algorithms.

We believe that many real world settings exhibit VV symmetries. For example, in the standard Potts model used in Computer Vision [Koller and Friedman, 2009], the energy function depends on whether the two neighboring particles take the same value or not, and not on the specific values themselves (hence, 00 would be symmetric to 11). Exploring VV symmetries in the context of specific applications is an important direction for future research.

We will also work on extending the theoretical guarantees of variable symmetries [Niepert,

2012] to VV symmetries. Several notions of symmetries already exist in the Constraint Satisfaction literature [Cohen *et al.*, 2006]. It will be interesting to see how our approach can be incorporated into the existing framework of symmetries in CSPs.

Chapter 5

Block-Value Symmetries

Permutations over variables [Niepert, 2012] and over variable-value (VV) pairs [Anand *et al.*, 2017] have been studied, with latter being a generalization of the former, capturing many more state symmetries. While more general, VV permutations clearly do not capture all possible state symmetries in a domain. For example, state $s_1 = (0, 0, 0, 0)$ is symmetric to $s_2 = (0, 1, 1, 1)$ in Figure 5.1(b), but VV permutations cannot represent it.

A natural question arises: are there more general representations which can capture (a subset of) these larger set of symmetries? We note that the problem of computing all possible symmetries is intractable since there is an exponential number of permutations over an exponentially large state space, each of which could be a symmetry (or not). Nevertheless, we hope there are representations which can capture additional symmetries compared to current approaches in bounded polynomial time. More so, it would be interesting to come up with a representation that enables computation of larger and larger sets of symmetries, while paying additional costs, which could be controlled as a function of a parameter of the representation.

As a significant step toward this research question, we develop the novel notion of symmetries defined over *block-value (BV) pairs*. Here, a block is a set of variables, and its value

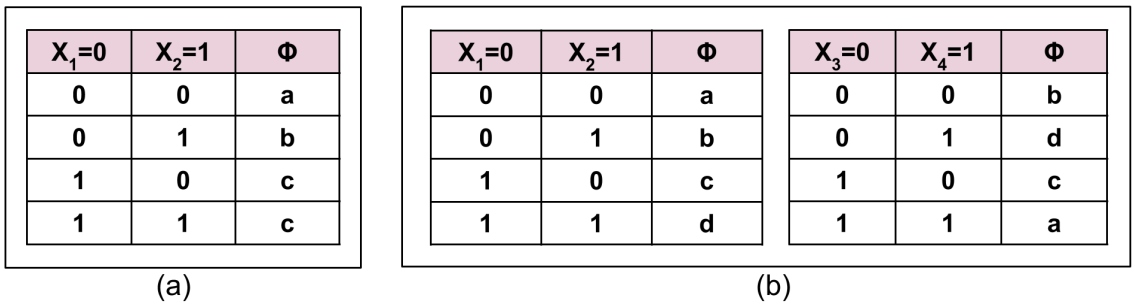


Figure 5.1: Block-Value Symmetries **(a)** BV Symmetries within a block **(b)** BV Symmetries across blocks

is an assignment to these variables. Intuitively, BV pairs can capture all such VV pairs that are not permuted independently, instead, are permuted in subsets together. For example, it can capture symmetry of states s_1 and s_2 via a BV permutation which maps $\{(X_1, 0), (X_2, 0)\} \leftrightarrow \{(X_3, 1), (X_4, 1)\}$ and $\{(X_1, 0), (X_2, 1)\} \leftrightarrow \{(X_3, 0), (X_4, 0)\}$.

Clearly, symmetries defined over BV pairs are a strict generalization of those over VV pairs, since each VV pair is a BV pair with a block of size 1. Our blocks can be of varying sizes and the size of each block essentially controls the set of symmetries that can be captured; larger the blocks, more the symmetries, coming at an additional cost (exponential in the max size of a block).

In this chapter, we formally develop the notion of symmetries as permutations defined over a subset of BV pairs. Some of these permutations will be invalid (when blocks overlap with each other) and their application may lead to inconsistent state. In order to ensure valid permutations, we require that the blocks come from a disjoint set of blocks, referred to as a *block partition*. Given a block partition, we show how to compute the corresponding set of symmetries by reducing the problem to one of graph isomorphism. We also show that our BV symmetries can be thought of as VV symmetries, albeit over a transformed graphical model, where the new variables represent the blocks in the original graph.

Next, we show that jointly considering symmetries obtained from different block partitions can result in capturing symmetries not obtainable from any single one. Since, there is an exponential number of such block partitions, we provide an efficient heuristic for obtaining a promising partition of blocks, referred to as a *candidate set*.

Use of BV symmetries in an MCMC framework requires uniform sampling of a state from each orbit, i.e., a set of symmetric states. This turns out to be a non-trivial task when the orbits are defined over symmetries corresponding to different block partitions. In response, we design an aggregate Markov chain which samples from orbits corresponding to each (individual) candidate set in turn. We prove that our aggregate Markov chain converges to the desired distribution. As a proof of the utility of our BV symmetries, we show that their usage results in significantly faster mixing times on two different domains.

This work was done jointly with Gagan Madan. While the initial idea was developed by Gagan, Ankit filled in important details, and also contributed to the experiments. The part done by Gagan appeared in his master's thesis.

5.1 Block-Value Symmetries

In this section, we will present symmetries defined over blocks of variables, referred to as *BV Symmetries* which strictly generalize the earlier notions of symmetries defined over VV

pairs. As a motivating example, Figure 5.1 shows two Graphical Models $\mathcal{G}_7$ and $\mathcal{G}_8$. For ease of explanation these have been represented in terms of potential tables. These can easily be converted to the weighted feature representation, as defined previously. In $\mathcal{G}_7$, state $(1, 0)$ has the same joint probability as $(1, 1)$ and in $\mathcal{G}_8$, state $(0, 0, 0, 0)$ has the same joint probability as $(0, 1, 1, 1)$. However, none of these can be captured by Variable or VV symmetries.

We start with some definitions.

Definition 5.1.1. *Let $B = \{X_1, X_2, \ldots, X_r\}$ denote a set of variables ($X_i \in \mathcal{X}$) which we will refer to as a* **block**. *Similarly, let $b = \{v_1, v_2, \ldots, v_r\}$ denote a set of (corresponding) assignments to the variables in the block B. Then, we refer to the pair (B, b) as a* **Block-Value (BV) pair**.

Definition 5.1.2. *A BV pair (B, b) is said to be* **consistent** *with a state s if $\forall X_i \in B$, $s(X_i) = v_i$ where v_i is the value for variable X_i in block B.*

Let Δ_V^r denote some subset of all possible BV pairs defined over blocks of size less than equal to r. For ease of notation, we will drop superscript r and denote Δ_V^r as Δ_V where r is a pre-specified constant for maximum block size. Then, we are interested in defining permutations over the elements of the set Δ_V. Considering any set of block-value pairs in Δ_V and allowing permutation among them may lead to inconsistent states. Consider a graphical model defined over four variables: $\{X_1, X_2, X_3, X_4\}$. Let us consider all possible blocks of size ≤ 2. Then, a BV permutation permuting the singleton block $\{X_1\}$ to itself (with identity mapping on values) while at the same time, permuting the block $\{X_1, X_3\}$ to the block $\{X_2, X_4\}$ is clearly inconsistent since X_1's value can not be determined uniquely. A natural way to avoid this inconsistency is to restrict each variable to be a part of single block while applying permutations. Therefore, we restrict our attention to sets of blocks which are non overlapping.

Definition 5.1.3. *Let $\Delta = \{B_1, B_2, \ldots, B_L\}$ denote a set of blocks. We define Δ to be a* **partition** *if each variable $X_i \in \mathcal{X}$ appears in exactly one block in Δ. For a partition Δ, we define the* **block value set** *Δ_V as a set of BV pairs where each block $B_l \in \Delta$ is present with all of its possible assignments.*

We would now like to define permutations over the block value set Δ_V, which we refer to as *BV-permutations*. To begin, we define the action of a BV-permutation $\psi : \Delta_V \to \Delta_V$ on a state s. The action of a BV-permutation $\psi : \Delta_V \to \Delta_V$ on a state s results in a state $s' = \psi(s)$ such that $\forall (B, b) \in \Delta_V$, (B, b) is consistent with s if and only if $\psi(B, b)$ is consistent with s'

However, similar to the case of VV symmetries, any bijection from $\Delta_V \to \Delta_V$ may not always result in a consistent state. For instance, consider a graphical model with 4 variables. Let the partition $\Delta = \{(X_1, X_2), (X_3, X_4)\}$. Consider the state $s = (0, 1, 1, 0)$. In case ψ is defined

as $\psi(\{X_1, X_2\}, \{0, 1\}) = (\{X_1, X_2\}, \{1, 0\})$ and $\psi(\{X_3, X_4\}, \{1, 0\}) = (\{X_1, X_2\}, \{1, 1\})$, the action of ψ results in an inconsistent state, since the action of ψ would result in a state with X_2 equal to both 0 and 1 simultaneously. To address this issue, we define a BV-permutation to be valid only under certain conditions.

Definition 5.1.4. *A BV-permutation $\psi : \Delta_V \to \Delta_V$ is said to be* **valid** *if $\forall (B_i, b_i) \in \Delta_V, \psi(B_i, b_i) = (B_j, b_j) \Rightarrow \forall b'_i, \exists b'_j$ such that $\psi(B_i, b'_i) = (B_j, b'_j)$*

Intuitively a BV-permutation ψ is valid if it maps all assignments of a block B to assignments of a fixed block B'.

Presently, it is tempting to define a new graphical model where each block is a multi valued variable, with domain of this variable describing all of the possible assignments. This would be useful in a lucid exposition of symmetries. To do this we must suitably transform the set of features as well to this new set of variables. Given a block partition Δ, we transform the set of features f_j such that for each block either all the variables in this block appear in the feature or none of them appear in the feature, while keeping all features logically invariant. We denote the set of all variables over which feature f_j is defined as $\mathcal{V}(f_j)$. Further, for a block B_l and a feature f_j, let $\bar{B}_l = B_l - \mathcal{V}(f_j)$ i.e $\bar{B}_l$ contains the additional variables in the block which are not part of feature f_j.

Definition 5.1.5. *Given a variable X_i, which appears in a block $B_l \in \Delta$ and a feature f_j,* **a block consistent** *representation of the feature, denoted by f'_j, is defined over the variables $\mathcal{V}(f_j) \cup \bar{B}_l$, such that, $f'_j(\mathbf{x}_j, \bar{b}_l) = f_j(\mathbf{x}_j)$ where $\mathbf{x}_j$, $\bar{b}_l$ denote an assignment to all the variables in $\mathcal{V}(f_j)$ and $\bar{B}_l$, respectively.*

For instance consider the feature $f = (X_2)$. Let the block B_l be $\{(X_1, X_2)\}$. Then the block consistent feature f' is given by $f' = (X_1 \wedge X_2) \vee (\neg X_1 \wedge X_2)$.

We extend the idea of block consistent representation to get a partition consistent representation $\hat{f}_j$.

Definition 5.1.6. A partition consistent *representation of a feature f_j, $\hat{f}_j$ is defined by iteratively converting the feature f_j to its block consistent representation for each $X_i \in \mathcal{V}(f_j)$.*

The set of partition consistent features $\{(\hat{f}_j, w_j)\}_{j=1}^m$ has the property that for all $B_l \in \Delta$, $B_l \subseteq Var(\hat{f}_j)$ or $B_l \cap Var(\hat{f}_j) = \phi$, i.e. all variables in each block either appear completely, or do not appear at all in any given feature. This property allows us to define a transformed graphical model $\hat{\mathcal{G}}$ over a set of multi valued variables $\mathcal{Y}$, where each variable $Y_l \in \mathcal{Y}$ represents a block $B_l \in \Delta$. The domain size of Y_l is the number of possible assignments of the variables in the block B_l. The set of features in this new model is simply the set of transformed features $\{(\hat{f}_j, w_j)\}_{j=1}^m$.

As the blocks are non overlapping, such a transformation can always be carried out.

Since the transformation of features to partition consistent features always preserves logical equivalence, it seems natural to wonder about the relationship between the graphical models $\mathcal{G}$ and $\hat{\mathcal{G}}$. We first note that each state s in $\mathcal{G}$ can be mapped to a unique state $\hat{s}$ in $\hat{\mathcal{G}}$ by simply iterating over all the blocks $B_l \in \Delta$, checking which BV pair (B_l, b_l) is consistent with the state s and assigning the appropriate value y_l to the corresponding variable Y_l. In a similar manner, each state $\hat{s} \in \hat{\mathcal{G}}$ can be mapped to a unique state in $s \in \mathcal{G}$.

Theorem 5.1.1. *Let s denote a state in $\mathcal{G}$ and let $\hat{s}$ be the corresponding state in $\hat{\mathcal{G}}$. Then, this correspondence is probability preserving i.e., $\mathcal{P}(s) = \hat{\mathcal{P}}(\hat{s})$ where $\mathcal{P}$ and $\hat{\mathcal{P}}$ are the distributions defined by $\mathcal{G}$ and $\hat{\mathcal{G}}$, respectively.*

Proof. For every feature f_j which holds true in state s, $\exists$ a partition consistent feature $\hat{f}_j$ which holds true in state $\hat{s}$ since f_j and $\hat{f}_j$ features are logically equivalent. The unnormalized probability of s is given by $e^{\sum_{j=1\to m} w_j f_j(s)}$. This expression will evaluate exactly same for $\hat{s}$ in transformed graphical model with f_j replaced by $\hat{f}_j$ and s replaced by $\hat{s}$ with numerical values exactly same. Hence, probability is preserved for s and $\hat{s}$, i.e. $\mathcal{P}(s) = \hat{\mathcal{P}}(\hat{s})$.

□

Similar to the mapping between states, every BV-permutation ψ of $\mathcal{G}$ corresponds to an equivalent VV-permutation $\hat{\phi}$ of $\hat{\mathcal{G}}$ obtained by replacing each BV pair in $\mathcal{G}$ by the corresponding VV pair in $\hat{\mathcal{G}}$ (and vice-versa). Since the distributions defined by the two graphical models are equivalent, we can define BV symmetries in $\mathcal{G}$ as follows:

Definition 5.1.7. *Under a given partition Δ, a BV-permutation ψ of a graphical model $\mathcal{G}$ is a* **BV-symmetry** *of $\mathcal{G}$ if the corresponding permutation $\hat{\phi}$ under $\hat{G}$ is a VV-symmetry of $\hat{\mathcal{G}}$.*

We can now state the following results for BV-symmetries.

Theorem 5.1.2. *BV-symmetries preserve weight-signature. Also, they are probability preserving transformations, i.e., for a BV-symmetry ψ, $\mathcal{P}(s) = \mathcal{P}(\psi(s))$ for all states $s \in \mathcal{S}$.*

Proof. This proof trivially extends using the similar theorem 4.1 for VV-symmetries. BV-symmetries are obtained by transforming the model and then, applying VV-symmetries over the transformed model. Since, VV symmetries preserve weight-signature. Hence, BV-symmetries preserve weight-signature. Further, given two states s and s' such that $s' = \psi(s)$, consider their equivalent states in transformed graphical model $\hat{\mathcal{G}}$, $\hat{s}$ and $\hat{s'}$ respectively. For every ψ, there is a corresponding VV-symmetry ϕ in transformed model. Using theorem 4.1, $\mathcal{P}(\hat{s}) = \mathcal{P}(\hat{s'})$. Also, from theorem 5.1.1, $\mathcal{P}(s) = \hat{\mathcal{P}}(\hat{s})$ and $\mathcal{P}(s') = \hat{\mathcal{P}}(\hat{s'})$. Hence, $\mathcal{P}(s) = \mathcal{P}(s')$ and BV-symmetries are probability preserving transformations. □

It is easy to that the set of all BV symmetries under a given partition Δ form a group Ψ. Similar to the VV orbits, we define the BV orbit of a state s as $\Gamma_\Psi(s) = \{\psi(s)|\psi \in \Psi\}$.

When the partition Δ is such that each variable appears in a block by itself, all the BV-symmetries are nothing but VV-symmetries.

Theorem 5.1.3. *Any VV-symmetry can be represented as a BV-symmetry for an appropriate choice of* Δ.

Proof. We can define the Δ as each variable belonging to a separate block. Then, we can transform a permutation on variable-values into singleton sets of variables and their values or blocks and their values. Hence, any VV-symmetry can be seen as BV-symmetry for blocks of singleton variable sets. □

Computing BV Symmetries

Since BV symmetry on a graphical model $\mathcal{G}$ is defined in terms of VV symmetry of a transformed graphical model $\hat{\mathcal{G}}$, BV symmetry can be trivially computed by constructing the transformed graphical model and then computing VV symmetry on $\hat{\mathcal{G}}$ as described by Anand et al. [2017].

5.2 Aggregate Orbital Markov Chains

Given a block partition Δ, BV symmetry group Ψ of $\mathcal{G}$ can be found by computing VV symmetry group Φ in the auxiliary graphical model $\hat{\mathcal{G}}$. We further setup a Markov chain *BV-MCMC(α)* over Ψ to exploit BV symmetries where $\alpha \in [0, 1]$ is a parameter.

Definition 5.2.1. *Given a graphical model $\mathcal{G}$, a Markov chain $\mathcal{M}$ and a BV symmetry group Ψ, one can define* **a BV-MCMC(α) Markov chain** *$\mathcal{M}'$ as follows: From the current sample s_t*
a) Sample a state s' from original Markov chain $\mathcal{M}$
b) i) With probability α, sample a state $s_{t+1} = \Gamma_\Psi(s')$ uniformly from BV orbit of s' and return s_{t+1} as next sample.
ii) With probability $1 - \alpha$, set state $s_{t+1} = s'$ and return it as the next sample

BV-MCMC(α) Markov chain is defined similar to VV-MCMC except that it takes an orbital move only with probability α instead of taking it always. For $\alpha = 1$, it is similar to VV-MCMC, and reduces to the original Markov chain $\mathcal{M}$ for $\alpha = 0$. When $\alpha = 1$, sometimes, it is observed that the gain due to symmetries is overshadowed by the computational overhead of the orbital step. The parameter α captures a compromise between these two contradictory effects.

Theorem 5.2.1. *Given a Graphical Model $\mathcal{G}$, if the original Markov chain $\mathcal{M}$ is regular, then, BV-MCMC(α) Markov chain $\mathcal{M}'$, constructed as above, is regular and converges to the unique stationary distribution of the original Markov chain $\mathcal{M}$.*

Proof. $\mathcal{M}'$ is regular since there is non-zero probability of returning to the same state. Firstly, $\mathcal{M}$ is regular and so, it brings to the same state with non-zero probability. Further, step b brings to the same state for all values of α since both part i) and ii) bring to the same state with some probability (part i i.e B.V orbital step also returns to same state). Hence, $\mathcal{M}'$ is regular and converges to a unique stationary distribution. We need to prove that it converges to the same unique stationary distribution of $\mathcal{M}$. Let π be unique stationary distribution of $\mathcal{M}$ and P be its transition probability matrix i.e

$$\pi(s) = \sum_{r \in \mathcal{S}} \pi(r) P(r \to s) \tag{5.1}$$

Let P' be transition probability matrix of $\mathcal{M}$. Then, we can write P' as follows:

$$P'(r \to s) = \alpha \sum_{s' \in \Omega(s)} \frac{1}{|\Omega(s)|} P(r \to s') + (1-\alpha) P(r \to s) \tag{5.2}$$

where the first first denotes step i) of b while second part of summation denotes step ii) of b and $\Omega(s')$ denotes BV orbit of s'. We need to prove this :

$$\pi(s) = \sum_{r \in \mathcal{S}} \pi(r) P'(r \to s) \tag{5.3}$$

Taking R.H.S,

$$\sum_{r \in \mathcal{S}} \pi(r) P'(r \to s) = \sum_{r \in \mathcal{S}} \pi(r) [\alpha \sum_{s' \in \Omega(s)} \frac{1}{|\Omega(s)|} P(r \to s') + (1-\alpha) P(r \to s)] \tag{5.4}$$

$$= \alpha \sum_{s' \in \Omega(s)} \frac{1}{|\Omega(s')|} \sum_{r \in \mathcal{S}} \pi(r) P(r \to s') + (1-\alpha) \sum_{r \in \mathcal{S}} \pi(r) P(r \to s) \tag{5.5}$$

$$= \alpha \sum_{q \in \Omega(s')} \frac{1}{|\Omega(s')|} \pi(s') + (1-\alpha) \times \pi(s) \tag{5.6}$$

$$= \alpha \times \pi(s') + (1-\alpha) \times \pi(s) = \alpha \times \pi(s) + (1-\alpha) \times \pi(s) = \pi(s) \tag{5.7}$$

The above calculation uses simple algebra and the fact that P has stationary distribution π. Hence, $\mathcal{M}'$ converges to the unique stationary distribution π.

□

It should be noted that two different block partitions may capture different BV symmetries and hence may have different BV symmetry groups. In order to fully utilize all symmetries which may be present in multiple block partitions, we propose the idea of **Aggregate Orbital Markov Chain**.

Consider K different block partitions $\Delta_1, \Delta_2, \ldots, \Delta_K$. We set up K independent BV-MCMC(α) Markov chains, where each chain generates samples as per BV-MCMC(α) corresponding to partition Δ_k. Let these chains be $\mathcal{M'}_1, \mathcal{M'}_2, \cdots, \mathcal{M'}_K$, and let the corresponding automorphism groups be $\Psi_1, \Psi_2, \ldots, \Psi_K$. Given an intermediate state s', we would like to sample uniformly from the union of orbits $\bigcup_k \Psi_k(s')$. Since these orbits may overlap with each other, sampling a state uniformly from the union of orbits is unclear. We circumvent this problem by setting up a new Markov chain, **Aggregate Orbital Markov Chain**. This Aggregate Orbital Markov Chain utilizes all available symmetries and converges to the true stationary distribution.

Definition 5.2.2. *Given K different BV-MCMC(α) Markov chains, $\mathcal{M'}_1, \mathcal{M'}_2, \cdots, \mathcal{M'}_K$, an* **Aggregate Orbital Markov Chain** *$\mathcal{M}^*$ can be constructed in the following way: Starting from state s_t a) Sample a BV-MCMC(α) Markov chain $\mathcal{M'}_k$ uniformly from $\mathcal{M'}_1, \mathcal{M'}_2, \cdots, \mathcal{M'}_K$ b) Sample a state s_{t+1} according to $\mathcal{M'}_k$.*

Theorem 5.2.2. *The aggregate orbital Markov chain $\mathcal{M}^*$ constructed from K BV-MCMC(α) Markov chains, $\mathcal{M'}_1, \mathcal{M'}_2, \cdots, \mathcal{M'}_K$, all of which have stationary distribution π, is regular and converges to the same stationary distribution π.*

Proof. Given each of BV-MCMC(α) Markov chains $\mathcal{M'}_k$ are regular, firstly, we prove that the aggregate Markov chain is regular. In each step of aggregate chain, one of the BV-MCMC(α) is applied and since, there is non-zero probability of returning to the same state in BV-MCMC(α) chain, there is non-zero probability of returning to the same state in $\mathcal{M}^*$. Hence, aggregate chain so defined is regular and therefore, it converges to a unique stationary distribution. [Koller and Friedman, 2009].

The only fact that remains to be shown is that the stationary distribution of $\mathcal{M}^*$ is π. Let $T^*(s \rightarrow s')$ represent the transition probability of going from state s to s' in aggregate chain $\mathcal{M}^*$. We need to show that

$$\pi(s') = \sum_{s \in \mathcal{S}} \pi(s) * T^*(s \rightarrow s') \tag{5.8}$$

Let $T_k(s \rightarrow s')$ represent the transition probability of going from state s to s' in $\mathcal{M'}_k$

$$\sum_{s \in \mathcal{S}} \pi(s) * T^*(s \rightarrow s') = \sum_{s \in \mathcal{S}} \pi(s) * \frac{1}{K} * \sum_{k=1}^{K} T_k(s \rightarrow s') \tag{5.9}$$

$$= \frac{1}{K} \sum_{k=1}^{K} \sum_{s \in \mathcal{S}} \pi(s) * T_k(s \rightarrow s') = \frac{1}{K} \sum_{k=1}^{K} \pi(s') = \pi(s') \tag{5.10}$$

Equation 5.9 follows from the definition of aggregate chain while equation 5.10 holds since $\mathcal{M'}_k$ converges to stationary distribution π.

□

Aggregate Markov chain $\mathcal{M}^*$ so obtained not only converges to the correct stationary distribution but also results in faster mixing since it can exploit the symmetries associated with each of the individual orbital Markov chains.

5.3 Heuristics for Block Partitions

We have so far computed BV symmetries *given* a specific block partition. We now discuss our heuristic that suggests candidate block partitions for downstream symmetry computation (see supplementary material for pseudo-code). At a high level, our heuristic has the following two desiderata. Firstly, it ensures that there are no overlapping blocks, i.e., one variable is always in one block. Secondly, it guesses which blocks might exhibit BV-symmetries, and encourages such blocks in a partition.

The heuristic takes the hyperparameter r, the maximum size of a block, as an input. It considers only those blocks (upto size r) in which for each variable in the block, there exists at least one other variable from the same block, such that some clause in $\mathcal{G}$ contains both of them. This prunes away blocks in which variables do not directly interact with each other, and thus are unlikely to produce symmetries. Note that these candidate blocks can have overlapping variables and hence not all can be included in a block partition.

For these candidate blocks, for each block-value pair, the heuristic computes a weight signature. The weight signature is computed by multiplying weights of all the clauses that are made true by the specific block-value assignment. The heuristic then buckets all BV pairs of the same size based on their weight signatures. The cardinality of each bucket (i.e., the number of BV pairs of the same size that have the same weight signature) is calculated and stored.

The heuristic samples a block partition as follows. At each step it samples a bucket with probability proportional to its cardinality and once a bucket is selected, then it samples a block from that bucket uniformly at random, as long as the sampled block does not conflict with existing blocks in the current partition i.e., it has no variables in common with them. This process is repeated until all variables are included in the partition. In the degenerate case, if a variable cannot be sampled from any block of size 2 or higher, then it gets sampled as an independent block of size 1. Once a partition is fully sampled, it is stored and the process is reset to generate another random block partition.

This heuristic encourages sampling of blocks that are part of a larger bucket in the hope that multiple blocks from the same bucket will likely yield BV symmetries in the downstream computation. At the same time, the non-conflicting condition and existence of single variable blocks jointly ensure that each sample is indeed a bona fide block partition.

Domain	Rules	Weights	Variables
Job Search	$\forall$ x TakesML(x) $\wedge$ GetsJob(x)	$+w_1$	TakesML(x),
	$\forall$ x $\neg$TakesML(x) $\wedge$ GetsJob(x)	$+w_2$	GetsJob(x),
	$\forall$ (x,y) Connected(x,y) $\wedge$ TakesML(x) $\Rightarrow$ TakesML(y)	w_3	Connected(x,y)
Student Curriculum	$\forall$ x Maths(x) $\wedge$ CS(x)	$+w_1$	
	$\forall$ x Maths(x) $\wedge$ $\neg$CS(x)	$+w_2$	
	$\forall$ x $\neg$Maths(x) $\wedge$ CS(x)	$+w_3$	Maths(x)
	$\forall$ x $\neg$Maths(x) $\wedge$ $\neg$CS(x)	$+w_4$	CS(x)
	$\forall$ (x,y) $\in$ Friends, Maths(x) $\Rightarrow$ Maths(y)	w	
	$\forall$ (x,y) $\in$ Friends, CS(x) $\Rightarrow$ CS(y)	w	

Table 5.1: Description of the two domains used in experiments. A weight of the form $+w_1$ indicates that the weight is randomly sampled for each object.

5.4 Experiments

Our experiments attempt to answer two key research questions. (1) Are there realistic domains where BV symmetries exist but VV symmetries do not? (2) For such domains, how much faster can an MCMC chain mix when using BV symmetries compared to when using VV symmetries or not using any symmetries?

5.4.1 Domains

To answer the first question, we construct two domains. The first domain models the effect of an academic course on an individual's employability, whereas the second domain models the choices a student makes in completing their course credits. Both domains additionally model the effect of one's social network in these settings. Table 5.1 specifies the weighted first order formulas for both the domains. The domains are constructed in such a way that no or limited VV symmetry is present but the domains exhibit BV symmetries. The aim is to validate that our approach is really effective if BV symmetry is present in the domain.

Job Search: In this domain, there are N people on a social network, looking for a job. Given the AI hype these days, their employability is directly linked with whether they have learned machine learning (ML) or not. Each person x has an option of taking the ML course, which is denoted by $TakesML(x)$. Furthermore, the variable $Connected(x, y)$ denotes whether two people x and y are connected in the social network or not. Finally, the variable $GetsJob(x)$ denotes whether x gets employment or not.

In this Markov Logic Network (MLN)[Domingos and Lowd, 2009], each person x participates in three kinds of formulas. The first one with weight w_1 indicates the (unnormalized) probability of the person getting a job and taking the ML course ($TakesML(x) \wedge GetsJob(x)$). The second formula with weight w_2 indicates the chance of the person getting a job while not

taking the course ($\neg TakesML(x) \wedge GetsJob(x)$). Our domain assigns different weights w_1 and w_2 for each person, modeling the fact that each person may have a different capacity to learn ML, and that other factors may also determine whether they get a job or not. Finally, x is more likely to take the course if their friends take the course. This is modeled by an additional formula for each pair (x, y), with a fixed weight w_3.

In this domain, there are hardly any VV symmetries, since every x will likely have different weights. However there are *intra-block* BV symmetries for the block $(TakesML(x), GetsJob(x))$ for every x. This is because within the potential table of this block the block values (0, 0) and (1, 0) are symmetric and can be permuted.

Student Curriculum: In this domain, there are N students who need to register for two courses, one from Mathematics and one from Computer Science to complete their course credits. There are two courses (basic or advanced) on offer in both disciplines. Variables $Math(x)$ and $CS(x)$ denote whether the student x would take the advanced course in each discipline. Since courses for Mathematics and CS could be related, each student needs to give a joint preference amongst the 4 available options. This is modeled as a potential table over $(Math(x), CS(x))$ with weights chosen randomly from a fixed set of parameters. Further, some students may also be friends. Since students are more likely to register in courses with their friends, we model this as an additional formula, which increases the probability of registering for a course in case a friend registers for the same.

In this domain, VV pairs can only capture symmetries when the potential tables (over $Math$ and CS) for two students are exactly the same. However, there are a lot more *inter-block* BV symmetries since it is more likely to find pairs of students, whose potential tables use the same set of weights, but in a different order.

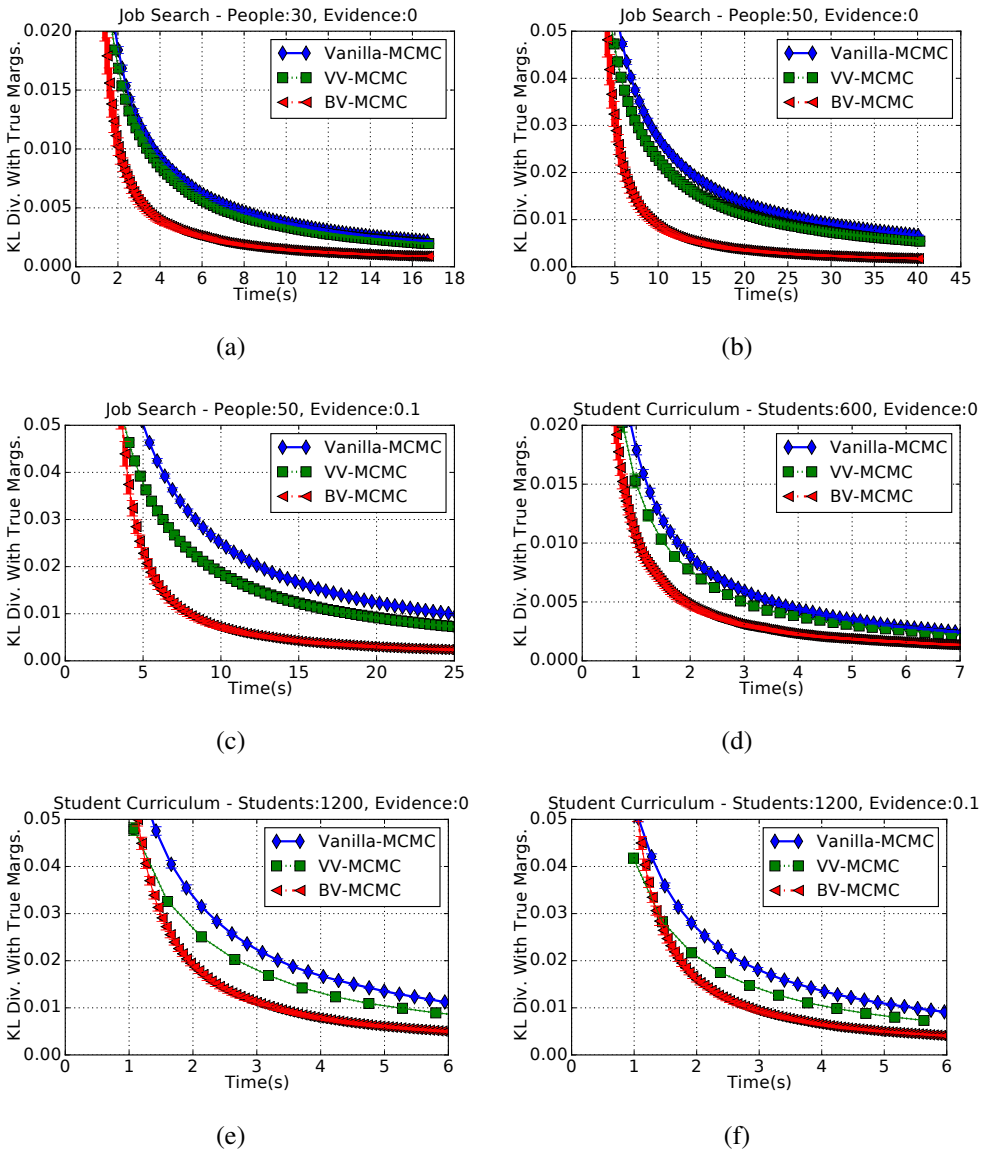


Figure 5.2: BV-MCMC($\alpha = 1$) and BV-MCMC($\alpha = 0.02$) outperforms VV-MCMC and Vanilla MCMC on Job Search and Student Curriculum domains respectively with different size and evidence variations

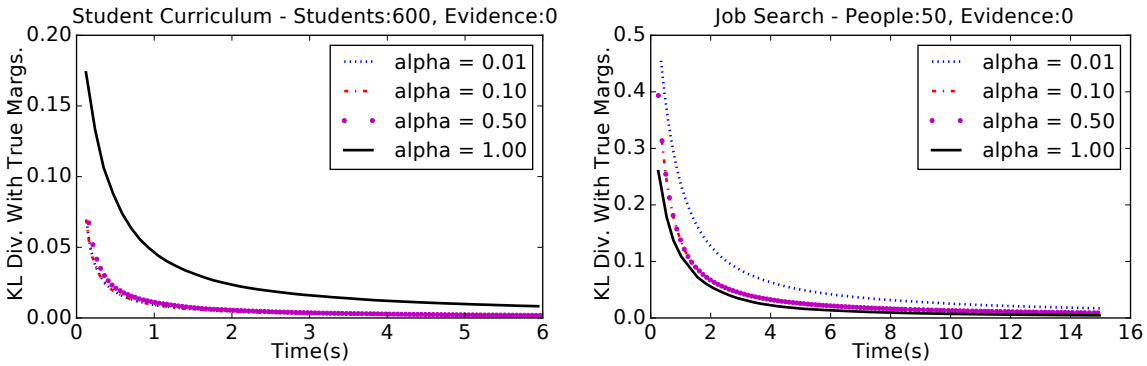


Figure 5.3: Variation on α $\alpha < 1$ is significantly better than $\alpha = 1$ in Student-Curriculum domain while $\alpha = 1$ is best in Job-Search domains

5.4.2 Comparison of MCMC Convergence

We now answer our second research question by comparing the convergence of three Markov chains – Vanilla-MCMC, VV-MCMC, and BV-MCMC(α). All three use Gibbs sampling as the base MCMC chain. All experiments are done on Intel Core i7 machines. Following previous work, and for fair comparison, we implement all the three Markov chains in group theoretic package - GAP [GAP, 2015]. This allows the use of off-the-shelf group theoretic operations. The code for generating candidate lists is written in C++. We solve graph isomorphism problems using the Saucy software [Darga *et al.*, 2008]. We release our implementation for future use by the community [1].

In all experiments, we keep the maximum block size in a block partition to be two. For each chain we plot the KL divergence of true marginals and computed marginals for different runtimes. We estimate true marginals by running the Gibbs sampling algorithm for a sufficiently long period of time. Each algorithm is run 20 times to compute error bars indicating 95% confidence interval.

For VV-MCMC and BV-MCMC, the run time on x-axis includes the preprocessing time of computing symmetries as well. For BV-MCMC, this includes the time for generating candidate lists, running Saucy for each candidate list, and initializing the Product Replacement algorithm for each candidate lists. The total preprocessing time for Job Search domain is around 1.6 sec and for Student Curriculum domain is around 0.6 sec.

Figures 5.2 shows that BV-MCMC substantially outperforms VV-MCMC and Vanilla-MCMC in both the domains. The parameter α is set to 1.0 for Job Search Domain and 0.02 for Student Curriculum Domain. Since these domains do not have many VV-Symmetries, VV-MCMC only marginally outperforms Vanilla MCMC. On the other hand BV-MCMC is able to exploit a considerably larger number of symmetries and leads to faster mixing. BV-MCMC scales well with

[1] https://github.com/dair-iitd/bv-mcmc

domain size, significantly outperforming other algorithms as domain size is changed from 30 to 50 people in Job Search and 600 to 1200 in Student Curriculum domain. This is particularly due to more symmetries being captured by BV-MCMC for larger domain sizes. [2]

Figure 5.2(c) and 5.2(f) plot the variation with introduction of 10% evidence in each domain. BV MCMC still outperforms VV-MCMC and Vanilla-MCMC and is robust to presence of evidence.

Finally, we also test the sensitivity of BV-MCMC with the α parameter. Figure 5.3 plots this variation on both these domains. We find that for Job Search, a high value $\alpha = 1$ performs the best, whereas a lower value is better in Student Curriculum. This is because Job Search mostly has intra-block BV symmetries, which can be computed and applied efficiently. This makes sampling an orbital step rather efficient. On the other hand, for Student Curriculum, the inter-block symmetry between different pairs of people makes the orbital step costlier, and reducing the fraction of times an orbital move is taken improves the overall performance.

5.5 Conclusion

Permutations defined over variables or variable-value (VV) pairs miss a significant fraction of state symmetries. We define permutations over block-value (BV) pairs, which enable a subset of variables (block) and their assignment to jointly permute to another subset. This representation is exponential in the size of the maximum block r, but captures more and more state symmetries with increasing r.

Novel challenges arise when building the framework and algorithms for BV permutations. First, we recognize that all BV permutations do not lead to valid state symmetries. For soundness, we impose a sufficient condition that each BV permutation must be defined on blocks with non-overlapping variables. Second, to compute BV symmetries, we describe a graph-isomorphism based solution. But, this solution expects a block partition as an input, and we cannot run it over all possible block partitions as they are exponential in number. In response, we provide a heuristic that outputs candidate block partitions, which will likely lead to BV symmetries. Finally, since the orbits from different block partitions may have overlapping variables, they cannot be explicitly composed in compact form. This makes it difficult to uniformly sample from the aggregate orbit (aggregated over all block partitions). To solve this challenge, we modify the Orbital MCMC algorithm so that in the orbital step, it uniformly samples from the orbit from any one of the block partitions (BV-MCMC). We prove that this aggregate Markov chain also converges to the true posterior.

Our experiments show that there exist domains in which BV symmetries exist but VV sym-

[2] Most of the error-bars are negligible in size.

metries may not. We find that BV-MCMC mixes much more rapidly than base MCMC or VV-MCMC, due to the additional mixing from orbital BV moves. Overall, our work provides a unified representation for existing research on permutation groups for state symmetries. In the future, we wish to extend this notion to approximate symmetries, so that they can be helpful in many more realistic domains as done in earlier works [Habeeb *et al.*, 2017].

Chapter 6

Complete Space of State Symmetries

Having introduced multiple novel notions of state symmetries, we analyze the complete space of all state symmetries in this chapter. As discussed previously, variable symmetries defined as permutation over variables are simplest of state symmetries. This was generalized to VV symmetries by defining permutations over variable-value pairs. We also proposed contextual symmetries which arise only under a given context. We further proposed BV symmetries which defined permutation over Block-value pairs. To make things complete, we begin by defining a class of BV-K symmetries and then analyze the relationship between all these sets of symmetries. The goal of this chapter is to develop a complete hierarchy of state symmetries in PGMs. In this chapter, we assume that contextual symmetries are defined for the context size of 1 though ideas can be extended to context having multiple variables. We begin by defining a class of BV-K symmetries.

Definition 6.0.1. *Given a graphical model $\mathcal{G}$,* BV-K symmetry class *includes all the BV symmetries defined over block partitions having block-size less than or equal to K*

Going from bottom to top, we know that variable symmetries are simplest notion of state symmetries. According to theorem 3.1.1, contextual symmetries subsume variable symmetries. Also, as per theorem 4.1, VV-symmetries subsume variable symmetries. The following theorem relates the relationship between VV-symmetries and BV-symmetries.

Theorem 6.0.1. *Given a graphical model $\mathcal{G}$, VV-symmetry class is equivalent to the BV-1 symmetry class.*

Proof. BV-1 symmetry class is identified over block-partitions having maximum block size 1. This means each block has a singleton variable and symmetries are defined over this single variable and its value which is precisely the class of VV symmetries. Hence, they result in equivalent state-partitions. □

The following theorem relates the relationship between BV-(K-1) symmetry class and BV-K symmetry class.

Theorem 6.0.2. *Given a graphical model $\mathcal{G}$, BV-K symmetry class subsumes all symmetries identified by BV-(K-1) symmetry class.*

Proof. This simply results from the definition of BV-K symmetries. Since, BV-K symmetries are defined over block-partitions having blocks of size less than or equal to K. Hence, BV-symmetries are also defined over block-partitions which have block-size less than or equal to (K-1). This is the definition of BV-(K-1) symmetries. Hence, BV-K symmetries subsumes all symmetries identified by BV-(K-1) symmetries. □

Next, we prove that BV symmetries with block-size equal to the number of variables capture the coarsest weight-signature preserving state-partition.

Theorem 6.0.3. *Given a graphical model $\mathcal{G}$, the state partition resulting from BV-N symmetries results in the coarsest weight-signature preserving state partition.*

Proof. BV-symmetries preserve weight signature as per theorem 5.1 and hence, they identify weight-signature preserving state partition. To prove BV-N partition is coarsest, we need to prove that any two states, having same weight-signature can be expressed as BV-N symmetry. Consider the two states s and s' having same weight-signature W. Now, we can define a BV-N symmetry ψ which captures exactly this state equivalence. In BV-N, symmetry class consider the partition where there is only one block of size N. Now, each state in itself is a BV pair. We define ψ such that s is mapped to s' and vice-versa and rest all other, states or BV pairs have identity mappings. Hence, all states having same weight-signature can be expressed as a BV-N symmetry. Therefore, BV-N captures the coarsest weight-signature preserving state partition. □

Although BV-N symmetries can represent the coarsest weight-signature preserving state partition, in the current setup BV-N symmetries will require explicitly enumerating exponentially large number of states. This is because for any given K, BV-K has to worry about permutation over all possible values that a K-sized block can take and clearly, number of such values is exponential in K. This points us to multiple open questions: a) An important research question is to explore whether BV-N symmetries can be represented and computed in a tractable fashion. The first and foremost thing could be to explore the specific conditions which must be met that render BV-N symmetries tractable. b) Secondly, we would to explore if a bigger block size symmetry can be captured by combining two smaller block size BV symmetries? A related question is how to combine symmetries from two different 2-BV partitions such that resulting symmetric group has compact representation? c) Lastly, it would be interesting to explore the

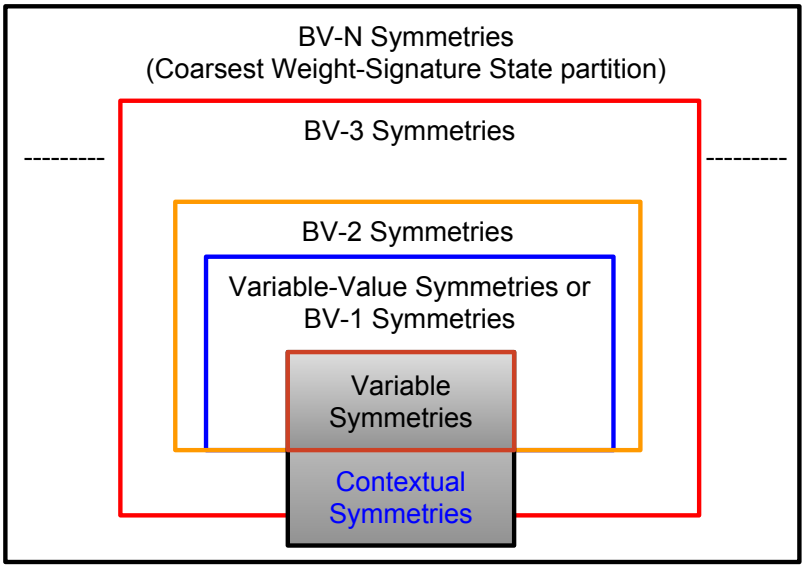


Figure 6.1: Hierarchy of different types of state symmetries

sampling approach where one samples a symmetry on the fly without explicitly computing the complete symmetric group. We believe answering these questions satisfactorily will result in developing further theoretical insights as well as new algorithms to compute symmetries.

The complete hierarchy looks as shown in the Figure 6.1. The innermost core consists of variable symmetries which lies inside VV-symmetries or BV-1 symmetries. BV-1 symmetries is successively enveloped by BV-2, BV-3 .... BV-K symmetries where BV-K strictly subsumes BV-(K-1) symmetry class. The outermost layer is of BV-N symmetries which captures the coarsest weight-signature preserving state partition. Further, we denote the contextual symmetries with the shaded region which has slightly complex relationship with other classes.

Relations with Contextual Symmetry: It is interesting to see how contextual symmetry is placed in this hierarchy. Since, BV-N symmetries capture the coarsest weight-signature preserving state partition, it subsumes contextual symmetries as well. But, contextual symmetries with a single-variable context is able to capture symmetries which cannot be identified even with a BV-(N-1) symmetry having block-sizes less than or equal to $N-1$. Consider a graphical model $\mathcal{G}$ having N-variables $\mathcal{X} = \{X_1, X_2 \cdots X_N\}$ and $N-1$ features all having different weight $f_1 = \{w_1 : X_1 \vee X_2\}$, $f_2 = \{w_2 : X_1 \vee X_3\} \cdots \mathrm{f}_{N-1} = \{w_{N-1} : X_1 \vee X_{N-1}\}$. Since, all variables have different weights, if $X_1 = 0$, then, there is no symmetry in reduced model, all variables $(X_2, X_3 \cdots X_N$ are different. But there is are many contextual symmetries for $X_1 = 1$, since, all variables are identical. This cannot be captured by BV-(N-1) symmetries. Hence, contextual symmetries are very powerful. Even with a single variable context, they are able to capture some symmetries which cannot be identified even with BV-(N-1) symmetries.

Further, consider another graphical model $\mathcal{G}'$ which has 5 variables $\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5\}$. The features are defined such that for $X_1 = 0$, X_2 and X_3 permute their values and similarly, for $X_1 = 0$, X_4 and X_5, permute their values. These symmetries can be clearly captured by BV-5 symmetries but intuitively, we will like to have these identified by BV-3 since at maximum 3 variables interact together in a feature. This contextual symmetry cannot be identified with BV-3 since context variable is global and shared across multiple blocks while the current BV framework does not allow overlapping BV-partitions.

Also, one should note that there is no contextual symmetry which is not variable symmetry but a BV-1 or BV-2 symmetry since context variable at least needs 2 other variables in the block to permute. This could be done if we define contextual symmetries over variable-value and block-value pairs as contextual VV or contextual BV with some additional changes in the definition of these symmetries. To formally define these frameworks is an interesting direction for future work.

This work has studied the spectrum of state symmetries comprehensively in Probabilistic Graphical Models. In this work, state symmetries have only been applied for improvements in MCMC algorithms. Though some of the works [Bui *et al.*, 2012; Kopp *et al.*, 2015] have used some of these state symmetries or similar ideas in other algorithms, how the formal framework defined here can be used in other algorithms both for Marginal and MAP inference is still not clear and an interesting direction to be explored. This will help in more widespread usage of these generic notion of state symmetries. More importantly, most of the experimentation in these works is performed on synthetic benchmarks illustrating the efficacy of the symmetries in question. An important direction is to apply these ideas in the real world problems where some of these symmetries naturally exist.

It should be noted that the current work only finds symmetry in states based on how parameters are shared. It does not consider equivalence among states that depend on exact values of parameters. For example, it will not be able to find equivalence between two states which have same probabilities ($P(s_1) = 0.4 * 0.1$ and $P(s_2) = 0.2 * 0.2$) due to equal product of weights though the actual weights being considered are different. We believe that computing these types of symmetries in general will require factorization of parameter values in potential tables and characterizing the symmetries in terms of prime factors. Developing complete end-to-end methods and techniques for computing and using these symmetries is an interesting future work direction.

Part III

Symmetries in Sequential Decision Making

Chapter 7

Abstraction of State-Action Pairs in UCT

"Decision making is an art only until the person understands the science."

Pearl Zhu

The science of decision making is central to the areas of operations research as well as artificial intelligence. In fact, automated decision making [Russell and Norvig, 2003] is considered one of the fundamental problems in the design of any autonomous agent. In this part of the thesis, we deal with the task of *sequential decision making* where an agent interacts with the environment to make a sequence of decisions. In addition, the agent operates in the real world and has to deal with the uncertainty present there, which makes the problem more complex. This problem of sequential decision making under uncertainty is often modeled as a Markov Decision Process (MDP) [Puterman, 1994] in the fields of AI planning and reinforcement learning. The key difference between AI planning and reinforcement learning is whether the environment dynamics are known to the agent apriori or not. If the environment dynamics is known to the agent, the problem is studied as an AI planning problem. On the other hand, if the environment dynamics are not known, the problem is studied as a reinforcement learning [Sutton and Barto, 1998] problem. For the purpose of our discussion, we restrict ourselves to the area of AI planning, though many ideas in the thesis can be extended to reinforcement learning field.

Traditional MDP planning algorithms (value iteration and variants) perform offline dynamic programming in flat state spaces and scale poorly with the number of domain features due to the curse of dimensionality. A well-known approach to reduce computation is through domain abstractions. Domain abstractions merges some state and actions together to obtain a smaller model which can be solved efficiently. The solution obtained for the smaller model is then transferred back to obtain the solution for the original unabstracted model. One of the popular way to abstract a domain without compromising any quality is to reduce the model by exploiting

symmetries. Although, there are multiple ways of applying abstractions in MDPs like state abstractions [Li *et al.*, 2006], action abstractions and temporal abstractions [Sutton *et al.*, 1999], we restrict ourselves to only symmetry based state and action abstractions. The advantage of symmetry based abstractions is that they are exact and hence, abstracted problem can return optimal solution for original problems. For the ease of readability and following the literature, we call symmetry based model reductions as abstractions in this part of thesis.

Existing offline abstraction techniques [Givan *et al.*, 2003; Ravindran, 2004] compute equivalence classes of states such that all states in an equivalence class have the same value or return. This projects the original MDP computation onto an abstract MDP, which is typically of a much smaller size. In this work, we develop a novel notion of abstraction, *state-action pair* (SAP) abstraction, where in addition to computing equivalence classes of states, we also compute equivalence classes of state-action pairs, such that return or value of state-action pairs in the same equivalence class are the same. This is different from merging similar actions within a state as done in some of previous works. Instead, we also merge state-action pairs even when the parent states of those actions are not equivalent. SAP abstractions generalize previous notions – abstractions studied in Givan *et al.* [2003] and Ravindran [2004]. These are special cases of SAP abstractions. Moreover, SAP abstractions may find symmetries even when there are not many available state abstractions.

Recently, Monte-Carlo Tree Search (MCTS) algorithms have become quite an attractive alternative to traditional MDP algorithms. MCTS algorithms, exemplified by the well-known UCT algorithm [Kocsis and Szepesvári, 2006], intelligently sample parts of the search tree in an online fashion. They can be stopped anytime and usually return a good next action. A UCT-based MDP solver [Keller and Eyerich, 2012] won the last two probabilistic planning competitions [Sanner and Yoon, 2011; Grzes *et al.*, 2014]. Unfortunately, UCT builds search trees in the original flat state space too, which is wasteful if there are useful symmetries and abstractions in the domain.

In our work, we implement SAP abstractions inside a UCT framework and call the resulting algorithm ASAP-UCT – **A**bstraction of **S**tate-**A**ction **P**airs in **UCT**. ASAP-UCT is an example of joint symmetry aware learning, since abstractions are computed within the UCT algorithm. Experiments on several probabilistic planning competition problems show that ASAP-UCT significantly outperforms both vanilla UCT as well as UCT with state abstractions obtaining upto 26% performance improvements. We implement and release[1] ASAP-UCT, an algorithm that exploits SAP abstractions in a UCT framework..

We begin by introducing the notation and giving the required background on MDPs, the history of abstractions in MDPs and Monte Carlo Tree Search algorithms. We also discuss

[1] Available at *https://github.com/dair-iitd/asap-uct*

some related work that uses abstractions within MCTS algorithms. Then, we formally define state-action pair abstractions and discuss the ASAP-UCT algorithm in detail. We also illustrate the effectiveness of ASAP-UCT on several benchmark domains and conclude this chapter by pointing out some interesting future problems.

7.1 Background and Related Work

The problem of sequential decision making under uncertainty is modelled as a Markov Decision Process (MDP) [Puterman, 1994]. Specifically, we restrict ourselves to the class of Stochastic Shortest-Path (SSP) MDPs [Bertsekas, 1995]. SSP MDPs are used to model scenarios where an agent needs to find the minimum expected cost of reaching any one of goal states in a stochastic environment. In this section, we introduce the notation of SSP MDPs which will be followed throughout this part of thesis. Further, we describe popular model abstraction techniques for MDPs developed in the past. We also give a brief overview of Monte Carlo Tree Search Algorithms (MCTS) like UCT [Kocsis and Szepesvári, 2006]. Finally, we discuss relations of our work with a few recent works that applied abstractions in Monte Carlo Tree Search Algorithms.

7.1.1 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical framework for sequential decision making under uncertainty. Depending on the time-frame and number of decisions, an MDP can either be infinite horizon, finite horizon or indefinite horizon MDP. Infinite horizon discounted MDPs have infinite number of decision steps, so the cost of future decisions is discounted (by a factor γ) to keep the longterm cost bounded. Finite horizon MDPs have limited and known number of decision steps. Indefinite horizon MDPs have fixed but unknown number of decision steps. As described in [Mausam and Kolobov, 2012], the most general framwork to handle all the above three scenarios is captured by Stochastic Shortest-Path (SSP) MDPs. A SSP MDP is modeled as a 5-tuple $(S, A, \mathcal{T}, \mathcal{C}, \mathcal{G})$ where:

- S is the set of states
- A is the set of actions
- $\mathcal{T}$ is the stationary transition function defined as $\mathcal{T} : S \times A \times S \rightarrow [0, 1]$
- C is the stationary cost function associated with the actions applicable at a given state denoted by $C : S \times A \rightarrow \mathcal{R}$

- $G \subseteq S$ is the set of goal states where $\forall s_g \in \mathcal{G}$, $s \notin G$, $a \in A$, we have $\mathcal{T}(s_g, a, s) = 0$ and $C(s_g, a) = 0$.

An agent in a state $s \in S$ executes an action $a \in A$ and makes a transition to $s' \in S$ with a probability $\mathcal{T}(s, a, s')$. The agent incurs a cost $C(s, a)$. A policy $\pi : S \rightarrow A$ is a mapping that returns an action to be executed in a state $s \in S$. In general, a policy can be stochastic or deterministic. Further, we define a proper policy π as a policy such that starting from any state s and following the policy π, we end up in one of the goal states $s_g \in \mathcal{G}$ with probability 1 in finite number of time steps. One of the strict conditions associated with a SSP MDP is existence of at least one proper policy [Mausam and Kolobov, 2012].

Given a starting state $s_0 \in S$, the total expected cost from state s or *the state-value function* $V^{\pi}(s)$ associated with a policy π is given by $V^{\pi}(s) = E[\sum_{t=0}^{\infty} C(s^t, a^t) | \pi(s^t) = a^t, t \geq 0, s = s_0]$ where the expectation is taken over the transition probability $\mathcal{T}(s^t, a^t, s^{t+1})$ of moving from state s^t to s^{t+1} under action a_t. Similarly, the total expected cost of a state-action pair or *the state-action value function* $Q^{\pi}(s, a)$ under the policy π denotes the cost of firstly taking action a in state s and then following π thereafter. The aim of an agent is to find a policy that minimizes the expected cost from the initial state s_0. The optimal policy π^* is given as: $\pi^*(s_0) = argmin_{\pi} V^{\pi}(s_0)$. Further, we use $Q^*(s, a)$ and $V^*(s)$ shorthand notations for $Q^{\pi^*}(s, a)$ and $V^{\pi^*}(s)$ respectively and call them optimal value functions. The expected cost of reaching any goal state can be recursively described by Bellman equations across multiple time steps as follows:

$$V_{t+1}(s) = \underset{a \in A}{argmin} \, [C(s, a) + \sum\nolimits_{s' \in S} \mathcal{T}(s, a, s') \times V_t(s')]$$

The value iteration algorithm [Bellman, 1957] starts with random initial V-Values (state-value function) and solves this equation repeatedly till there is no change in state-value function of all states. The V-Value so obtained gives the optimal value function V^* and the optimal policy is obtained $\pi^*(s) = argmin_{a \in A} V^*(s_0)$.

An MDP can be equivalently represented as a acyclic AND-OR graph [Mausam and Kolobov, 2012] in which OR nodes are MDP states and AND-nodes represent state-action pairs whose outgoing edges are multiple probabilistic outcomes of taking the action in that state. Value Iteration [Bellman, 1957] and other dynamic programming MDP algorithms can be seen as message passing in the AND-OR graph where AND and OR nodes iteratively update Q(s,a) and V(s) (respectively) until convergence.

For the purpose of our discussion in this thesis, we restrict ourselves to the problem of Finite-Horizon MDPs. Finite-Horizon MDPs have a finite fixed number of maximum decision steps D explicitly defined. All the states at the depth of D are treated as goal or absorbing states.

We will refer to this fixed number of decision steps as execution horizon of MDPs. Since these MDPs have fixed number of decision steps, the same world state at different decision step should be treated as a different state. Hence, any finite horizon MDP can be converted to an SSP by augmenting the state with a depth variable i.e., if S is the state space of finite-horizon MDP, then, the state-space of the corresponding SSP is given by $S' = S \times D$. Further, we distinguish between the terms planning horizon and execution horizon. Planning horizon refers to the maximum depth explored by an MDP planner while taking a single decision. On the other hand, execution horizon refers to the maximum number of decisions to be executed by a planner.

7.1.2 Model Abstraction Techniques in MDPs

Many states and actions in MDPs behave similar to each other (are symmetric) and can be abstracted out. Several works in the past have attempted to reduce the original MDP model by exploiting symmetries and have developeds a variety of model abstraction techniques. Most of these model abstraction techniques have three key steps. The first step is to obtain a reduced model by merging states and actions based on some criteria. The second step is to solve the reduced model using a standard MDP algorithm. Finally, the values and policies obtained from the reduced model are mapped back to the original model. The premise behind such an approach is that solving a reduced model would be substantially faster or would be the only feasible method when it is not possible to directly solve the original MDP. Broadly, all these model abstraction techniques can be divided into exact and approximate techniques. Exact model abstraction methods reduce the model in such a way that solving the abstracted MDP optimally and mapping back the values yield value functions and policies which are optimal in the original MDP. Approximate model abstraction techniques yield values and policies which may not be exactly same or optimal but are hopefully near those obtained with original model. Approximate techniques could be of great value in many practical scenarios with limited time and memory.

Typically, the model is abstracted out by merging states and actions that behave identically to one another. Earlier works have exploited state abstractions by merging states based on notions of bisimulations [Givan *et al.*, 2003] and graph homomorphism [Ravindran, 2004]. We describe these two notions and illustrate them with a toy example of soccer domain (Figure 7.1). Here, a player wishes to score a goal and can be in any of four positions. A player in central position (S0) can pass the ball left (L), right (R) or shoot at the goal straight (S). Hence, for a state (S0), there are 3 feasible actions: L,R,S. The player in top position (S1) can hit the ball right to shoot the goal. A player at the bottom positions (S2, S3) can hit the ball left for a goal. All the actions for simplicity are assumed to have uniform probability distribution over

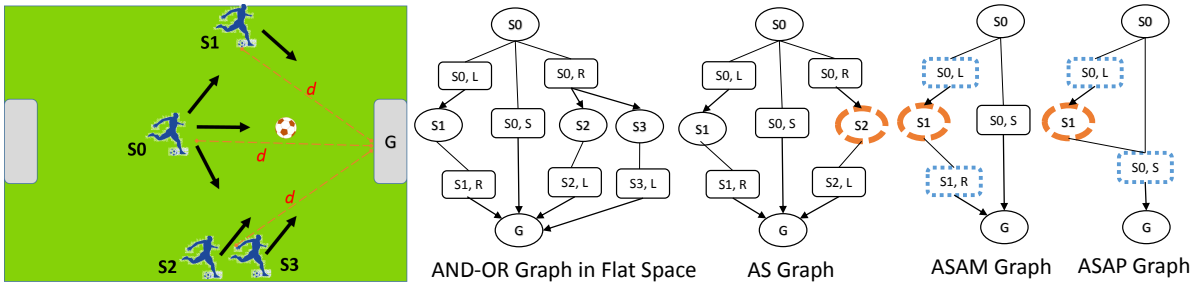


Figure 7.1: An example showing abstractions generated by various algorithms on a soccer domain. Givan's AS, Ravindran's ASAM and our ASAP frameworks successively discover more and more symmetries.

its outcome states and the cost of taking any action is 1. It should be noted that right action (S0,R) of central position (S0) leads to (S2) and (S3) with uniform probabilities. The equivalent AND-OR graph for this domain is the leftmost graph in the figure.

7.1.2.1 Abstractions as Bisimulations

The initial idea of bisimulations was used to define state equivalence in Finite State Machines (FSMs) and concurrent processes literature [Park, 1981; Hartmanis, 1966]. Later, this was extended to Markov Decision Processes by Givan *et al.* [2003]. Given an MDP $M(S, A, \mathcal{T}, \mathcal{C}, \mathcal{G})$ where the notations are as defined in Section 7.1.1, we define a relation $\mathcal{E}^{BS} \subseteq S \times S$ as a bisimulation if it satisfies the below mentioned conditions. Additionally, these conditions also ensure that $\mathcal{E}^{BS}$ is reflexive, symmetric and transitive and hence, we define the set of equivalence classes under $\mathcal{E}^{BS}$ by the set $\mathcal{X}$ and the corresponding equivalence function by $\mu_{\mathcal{E}^{BS}}$.

Definition 7.1.1. *A relation $\mathcal{E}^{BS} \subseteq S \times S$ is a bisimulation if and only if $\forall\ (s, s') \in \mathcal{E}^{BS}$, it satisfies the following conditions: $\forall a \in A$*

- *$\mathcal{C}(s, a) = \mathcal{C}(s', a)$, i.e for any action label, cost of taking that action in both the states s and s' should be equal.*
- *Secondly,*

$$\forall x \in \mathcal{X}, \sum_{s'' \in S} \mathbb{1}[\mu_{\mathcal{E}^{BS}}(s'') = x] \times \mathcal{T}(s, a, s'') = \sum_{s'' \in S} \mathbb{1}[\mu_{\mathcal{E}^{BS}}(s'') = x] \times \mathcal{T}(s', a, s'')$$

where $\mathbb{1}$ is the indicator function. Intuitively, both the states s and s' should transition to the same set of equivalences classes with matching probabilities.

For example, in Figure 7.1, $\mathcal{E}^{BS} = \{(S2, S3), (S3, S2), (S2, S2), (S3, S3)\}$. It is trivial to find that both $S2$ and $S3$ have a single action L which transitions deterministically to goal State G.

It should be noted that this bisimulation relation $\mathcal{E}^{BS}$ abstracts a given MDP M into a smaller MDP M' defined in terms of equivalence classes $\mathcal{X}$ induced by the equivalence relation. In addition, the original bisimulation relation is fairly general and can also be defined over state spaces of two different MDPs. We have restricted that definition to a single MDP for the purpose of abstraction of states of a single MDP. We specifically call the abstractions based on bisimulations as *Abstraction of States (AS)* in this work.

7.1.2.2 Abstractions as Homomorphisms

The above definition of state abstraction using bisimulation is strict and does not allow abstraction of two states when the action labels of two states do not match. For instance, in Figure 7.1, state S1 behaves similar to state S2 (or S3) if we replace the action label R of S1 with L of S2 (or S3). This strict notion was relaxed by the idea of homomorphisms [Ravindran and Barto, 2004]. Given an MDP $M(S, A, \mathcal{T}, \mathcal{C}, \mathcal{G})$, a homomorphism is a surjection $h : M \rightarrow M'$ where $M'(\mathcal{X}, A', \mathcal{T}', \mathcal{C}', \mathcal{G}')$ is a smaller MDP which preserves the structure and dynamics of M. Let A_s denotes the set of actions applicable in a state s.

Definition 7.1.2. *A homomorphism $h : M \rightarrow M'$ is a surjection defined by a 2-tuple $(f : S \rightarrow \mathcal{X}, \{g_s : A_s \rightarrow A'_x | s \in S; x \in \mathcal{X}\})$ where f is a surjection over state space and $\{g_s | s \in S\}$ is a set of surjections from ground actions of state $s \in S$ to actions of an equivalent state x in smaller MDP M'. Further, f and $\{g_s | \forall s \in S\}$ are defined as follows:*

- *Two states $s, s' \in S$ have $f(s) = f(s')$ if $\forall a \in A_s$, $\exists a' \in A_{s'}$ for which $g_s(a) = g_{s'}(a')$ and vice versa.*
- *Further, we have $g_s(a) = g_{s'}(a')$ iff*
  1. *$\mathcal{C}(s, a) = \mathcal{C}(s', a')$ i.e cost of taking action a in state s is equal to cost of taking action a' in state s'.*
  2. *Similar to bisimulations, transition effect of taking action a in state s is equivalent to effect of taking action a' in state s'*

$$\forall x \in \mathcal{X}, \sum_{s'' \in S} \mathbb{1}[f(s'') = x] \times \mathcal{T}(s, a, s'') = \sum_{s'' \in S} \mathbb{1}[f(s'') = x] \times \mathcal{T}(s', a', s'') \quad (7.1)$$

It should be carefully noted that conditions for $g_s(a) = g_{s'}(a')$ can also be applied to define action equivalence within a state by having $s = s'$ and hence, can be used to prune redundant actions of a state. For subsequent discussions, we refer to abstractions based on homomorphisms as Abstraction of States with Action Mapping (ASAM). Further, the partition of state-space induced by bisimulation is finer than the partition of state-space induced by homomorphism.

Theorem 7.1.1. *For any given MDP M, If $\mathcal{E}^{BS}$ is the bisimulation relation, $\mu_{\mathcal{E}^{BS}}$ is the corresponding equivalence function then, $\exists$ a homomorphism $h : (f, \{g_s\})$ s.t. $\forall s, s' \in S$ if $\mu_{\mathcal{E}^{BS}}(s) = \mu_{\mathcal{E}^{BS}}(s')$, then, $f(s) = f(s')$.*

Proof. If $\mu_{\mathcal{E}^{BS}}(s) = \mu_{\mathcal{E}^{BS}}(s')$, then, $\forall a \in A_s, \exists a' \in A_{s'}$ s.t $a' = a$ and $g_s(a) = g_{s'}(a)$ is true. This is because Condition 1 and 2 are true $\forall a \in A$ according to definition of bisimulations. Hence, $f(s) = f(s')$ as per definition of homomorphisms. □

Therefore, the notion of homomorphism is strictly more general than the notion of bisimulations as it reduces the original model much more than abstractions based on bisimulations. This can also be seen in Figure 7.1 where the model based on homomorphism is smaller than that obtained by bisimulations.

7.1.2.3 Other Notions of Abstractions

Apart from bisimulations and homomorphisms, many other notions of abstraction have been proposed in the literature. Li *et al.* [2006] have comprehensively surveyed different notions of state abstractions for MDPs. These also include approximate bisimulations and approximate homomorphisms. In addition, they define a special type of state abstractions for MDPs called Q-irrelevance abstraction. Two states belong to a same abstract class under Q-irrelevance abstraction if Q-values (for same action) of those two states are exactly equal. This Q-irrelevance abstraction could then preserve Q-values for all policies or only the optimal policy. They also define an α^* irrelevance criteria where all states in the abstract class have same optimal action and value functions. This criteria is relaxed in the π^* irrelevance criteria where ground states in the abstract class have only same optimal action. Ferns et. al. [2004] have also studied various bisimulation metrics for finite MDPs.

Our proposed ASAP framework unifies and extends the exact notions of abstractions particularly bisimulations and homomorphisms. Further, we go beyond just an equivalence relation $\mathcal{E}$ over states, and compute equivalences of *state-action pairs*. This additional notion of abstractions leads to many more nodes getting merged together and can obtain significant computational savings. Next, we describe an alternative approach to handle large state and actions spaces by a popular sampling based method, called Monte Carlo Tree Search (MCTS) algorithms.

7.1.3 Monte-Carlo Tree Search (MCTS)

Traditional offline MDP algorithms store the whole state space in memory and scale poorly with number of domain features. Sampling-based MCTS algorithms offer an attractive alternative.

They solve finite-horizon MDPs in an online manner by interleaving planning and execution steps. A popular variant is UCT [Kocsis and Szepesvári, 2006], in which during the planning phase, starting from the root state, a tree is constructed based on sampled trajectories. It maintains an estimate for the Q value for each state at a given depth. UCT chooses an action a in a state s at depth d based on the UCB rule, $argmin_{a \in A} \left(Q(s,d,a) - K \times \sqrt{\frac{log(n(s,d))}{n(s,d,a)}} \right)$ where $K > 0$. Here, $n(s,d)$ denotes the number of trajectories that pass through the state s at depth d and $n(s,d,a)$ is the number of trajectories that take action a after visiting state s at depth d. Note that above rule means that if a particular action a has not been tried at some state depth pair (s,d)[2], then its score will be $-\infty$ and hence, will be preferred over already explored actions in that state.

Evaluation of a leaf node is done via a random *rollout*, in which actions are randomly chosen based on some default rollout policy until a goal or some planning horizon is reached. This rollout results in an estimate of the Q-value at the leaf node. Finally, this Q-value is backed up from the leaf to the root. UCT operates in an anytime fashion – whenever it needs to execute an action it stops planning and picks the best action at the root node based on the current Q-values. The planning phase is then repeated again from the newly transitioned node. Due to the clever balancing of the exploration-exploitation trade off, MCTS algorithms can be quite effective and have been shown to have significantly better performance in many domains of practical interest [Gelly and Silver, 2011; Balla and Fern, 2009].

There have been multiple works which studied UCT in the context of probabilistic planning. Specifically, the state-of-the-art planner Prost [Keller and Eyerich, 2012] that has won International Probabilistic Planning Competition-2011 and IPPC-2014 [Sanner and Yoon, 2011; Grzes *et al.*, 2014] is based on UCT. It employs multiple additional heuristics to improve the performance of UCT. In addition, many works [Bonet and Geffner, 2012; Kolobov *et al.*, 2012] have compared performance of UCT with other planning techniques. Bonet *et al.* [2012] compared performance of UCT with non-admissible heuristic algorithms. It argues that heuristic algorithms have stronger theoretical properties but UCT performs better when the planning time is very small. Similar to this, Kolobov *et al.* [2012] compared performance of UCT with dynamic programming based LRTDP algorithms. It demonstrated scenarios of large finite horizon MDPs where UCT algorithms are likely to make mistakes. Inspite of these results, UCT has been the algorithm of choice for many real-world probabilistic planners.

7.1.4 Abstractions in MCTS

Some recent works [Hostetler *et al.*, 2014; Hostetler *et al.*, 2015; Jiang *et al.*, 2014] have realized that abstraction based approaches and MCTS have complementary advantages for improv-

[2]Such states are called the partially explored states

ing the efficiency of planning in large domains. While abstraction based approaches reduce the search space by merging similar states and actions together, MCTS explores only useful parts of search space by carefully managing exploration-exploitation trade-off. In light of this insight, many recent works applied abstractions in MCTS algorithms.

Hostetler *et al.* [2014] develop a theoretical framework for defining a series of state abstractions in sampling-based algorithms for MDP. But they do not provide any automated algorithm to compute the abstractions themselves. Another work [Jiang *et al.*, 2014] applies Givan et. al. [2003]'s definitions of state abstractions within UCT. The key insight is that instead of an offline abstraction algorithm, they test abstractions only for the states enumerated by UCT. Their approach first puts all partially explored nodes (not all of whose actions are present in the UCT tree) in a single abstract state per depth. Since UCT solves finite-horizon MDPs, only the states at the same depth will be considered equivalent. Then, at any given depth, they test Givan's conditions (transition and cost equality) on pairs of states to identify ones that are in the same equivalence class. This algorithm proceeds bottom-up starting from last depth all the way to the root. Jiang *et al.* [2014] also consider two approximation parameters $\epsilon_{\mathcal{T}}$ and $\epsilon_{\mathcal{C}}$ which aggregate two states together if the corresponding transition probabilities and costs are within $\epsilon_{\mathcal{T}}$ and ϵ_C, respectively. Their paper experimented on a single deterministic game playing domain and its general applicability to planning was not tested. In this chapter, we advance Jiang's ideas in our algorithm ASAP-UCT by applying our novel state-action pair (SAP) abstractions in UCT, and show that they are more effective on a variety of domains.

Another work, [Hostetler *et al.*, 2015] applies adaptive abstraction computation in an MCTS framework called Progressive Abstraction Refinement for Sparse Sampling (PARSS). PARSS applies abstractions to sparse sampling framework [Kearns *et al.*, 2002]. It begins from a fully abstract tree, and in later iterations, splits some of the merged nodes. This process of splitting an abstract node into two or more than two parts is called as *refinement*. The refinement procedure is somewhat ad hoc in that it splits an abstract node into two nodes arbitrarily and does not use any domain information whatsoever. It is also not clear how PARSS scales with increase in depth. Its strength is its ability to find better solutions when planning time is extremely small due to heavy initial abstraction. Moreover, it is an incremental algorithm i.e. computes abstractions in a non-batch manner.

There is also recent work on improving exploration in UCT when rewards are sparse using local manifold learning [Srinivasan *et al.*, 2015]. They improve exploration by generalizing the rollout values across all nearest neighbor states based on a distance metric determined by manifold learning of state space. Another recent work [Jiang *et al.*, 2015] investigates selecting an appropriate level of abstraction among candidate abstractions in a model-based reinforcement learning framework.

7.2 Abstraction of State-Action Pairs (ASAP) Framework

In this section, we formalize our abstraction framework. The key contribution of our work is to introduce the notion of state-action pair (SAP) abstractions. Earlier approaches to abstraction in MDPs dealt either with state abstractions only [Givan *et al.*, 2003] or state abstractions with a (complete) mapping of actions belonging to two different states [Ravindran, 2004]. This can be restrictive in cases where two states may not behave identically to each other but application of a subset of allowed actions in them results in identical transition behavior (and leads to same or similar behaving states). Our SAP abstractions overcome this limitation by effectively allowing for partial mapping of actions between two different states. We will first formally define our abstraction framework followed by an illustrative example. We will then show that our approach strictly generalizes existing notions of abstractions.

To formally define the framework we introduce some notation. Consider an MDP $M = (S, A, \mathcal{T}, C, \mathcal{G})$. We use P to denote the set of State-Action Pairs (SAPs) i.e. $P = S \times A$. We define an equivalence relation $\mathcal{E}$ over pairs of states as earlier i.e. $\mathcal{E} \subseteq S \times S$ and $\mathcal{X}$ denotes the set of equivalence classes under the relation $\mathcal{E}$. Let $\mu_{\mathcal{E}} : S \to \mathcal{X}$ denote the corresponding equivalence function mapping each state to the corresponding equivalence class. Similar to states, we define an equivalence relation $\mathcal{H}$ over pairs of SAPs, i.e. $\mathcal{H} \subseteq P \times P$. Let $\mathcal{U}$ denote the set of equivalences classes under the relation $\mathcal{H}$, and, analogously, let $\mu_{\mathcal{H}} : P \to \mathcal{U}$ denote the corresponding equivalence function mapping state-action pairs to the corresponding equivalence classes. Following previous work, we will recursively define state equivalences using state-action pair equivalences and vice-versa.

Suppose we are given SAP abstractions, and $\mu_{\mathcal{H}}$. Intuitively, for state equivalence to hold, there should be a correspondence between applicable actions in the two states such that the respective state-action pair nodes are equivalent. Formally,

Definition 7.2.1. State Abstractions: *Let $a, a' \in A$ denote two actions applicable in s and s', respectively. We say that two states s and s' are equivalent to each other (i.e, $\mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$) if for every action a applicable in s, there is an action a' applicable in s' (and vice-versa) such that $\mu_{\mathcal{H}}(s, a) = \mu_{\mathcal{H}}(s', a')$.*

For stochastic shortest path MDPs, all goal states are in an equivalence class: $\forall s, s' \in G, \mu_{\mathcal{E}}(s) = \mu_{\mathcal{E}}(s')$.

To define SAP abstractions, assume we are given state abstractions and the $\mu_{\mathcal{E}}$ function.

Definition 7.2.2. SAP Abstractions: *Two state-action pairs (s, a), $(s', a') \in P$ are said to be equivalent i.e. $\mu_{\mathcal{H}}(s, a) = \mu_{\mathcal{H}}(s', a')$ iff:*

- $\forall x \in \mathcal{X}$, $\sum_{s_t \in S} \mathbb{1}[\mu_{\mathcal{E}}(s'') = x]\mathcal{T}(s, a, s'') = \sum_{s'' \in S} \mathbb{1}[\mu_{\mathcal{E}}(s'') = x]\mathcal{T}(s', a', s'')$ *where $\mathbb{1}$ is indicator function (Condition 1)*

- $C(s,a) = C(s',a')$ *(Condition 2)*

In other words, for state-action pair equivalence to hold, the sum of transition probabilities, to each abstract state that these state-action pair transition to, should match. Second condition requires the costs of applying corresponding actions to be identical to each other.

It is important to note that ASAP framework does not reduce the original MDP M into an abstract MDP as done in earlier work. The reason is that the framework may abstract two state-action pairs whose parent states are not equivalent. It is not possible to represent this as an MDP, since MDP does not explicitly input a set of SAPs. However, ASAP framework can be seen as directly abstracting the equivalent AND-OR graph Gr, in which there is an OR node for each abstract state $x \in \mathcal{X}$, and an AND node for each abstract SAP $u \in \mathcal{U}$. There is an edge from x to u in Gr if there is a state-action pair (s,a) such that a is applicable in s, $\mu_{\mathcal{E}}(s) = x$ and $\mu_{\mathcal{H}}(s,a) = u$. The associated cost with this edge is $C(s,a)$ which is the same for every such (s,a) pair (follows from Condition 2). Similarly, there is an edge from u to x if there is a state-action-state triplet (s,a,s_t) such that application of a in s results in s_t (with some non-zero probability), $\mu_{\mathcal{H}}(s,a) = u$ and $\mu_{\mathcal{E}}(s_t) = x$. The associated transition probability is $\sum_{s'_t \in S} \mathbb{1}[\mu_{\mathcal{E}}(s'_t) = x]\mathcal{T}(s,a,s'_t)$ (follows from Condition 1).

Example: Figure 7.1 illustrates the ASAP abstractions on this soccer domain. Our ASAP framework in this domain will additionally recognize that S0's straight is equivalent to S1's right and merge these two SAP nodes in addition to states and actions merged by AS and ASAM. Overall, ASAP will identify the maximum symmetries in the problem. □

ASAP framework is a strict generalization of past approaches. Here, it is important to state that there are certain key differences between our SAP abstraction definitions and Ravindran's ASAM. Past work constrains SAPs to be equivalent only if the parent states were equivalent too. It did not directly use SAP abstractions in planning – they used those merely as a subroutine to output state abstractions. Our definitions differ by computing SAP equivalences irrespective of whether the parent states are equivalent or not, and thereby reducing the AND-OR graph of the domain further.

Theorem 7.2.1. *Both AS and ASAM are special cases of ASAP framework. ASAP subsumes all abstractions computed by AS and ASAM.*

Proof. As proved in Section 7.1.2.2, abstractions computed by bisimulations are a special case of abstractions computed by homomorphisms. Hence, ASAM computes all symmetries computed by AS. The only thing remaining to prove is that ASAP subsume the abstractions computed by ASAM. If ASAM symmetries consider s_1 and s_2 as equivalent, then, the same states will also be considered equivalent in ASAP since if $g_{s_1}(a_1) = g_{s_2}(a_2)$ then, it should have $\mu_{\mathcal{H}}(s_1,a_1) = \mu_{\mathcal{H}}(s_2,a_2)$ by the same definition. Hence, ASAP abstractions computes all abstractions computed by ASAM. On the other hand, ASAM merges only those state-action pairs

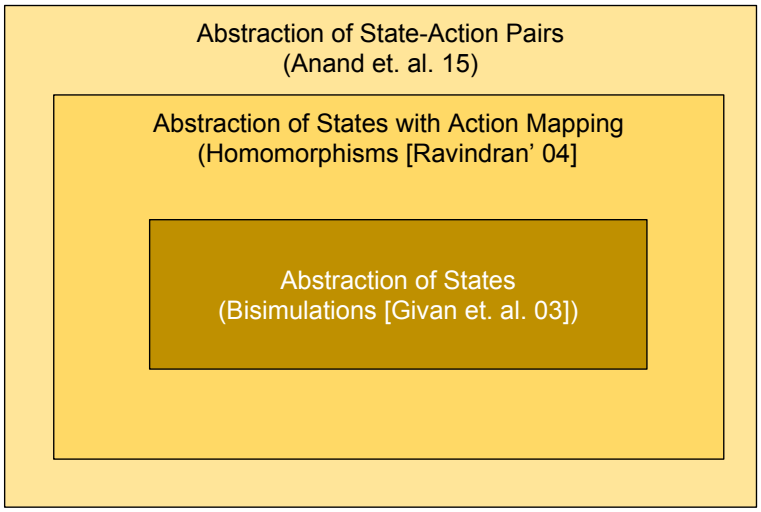


Figure 7.2: ASAP abstractions subsume ASAM abstractions which in turn subsume AS abstractions

whose parent states match, i.e it will not merge any $p_1 = \{s_1, a_1\} \in P$ and $p_2 = \{s_2, a_2\} \in P$ if $\mu_{\mathcal{E}}(s_1) \neq \mu_{\mathcal{E}}(s_2)$ since it only discusses equivalences of states and does not exploit equivalence of state-actions pairs if their corresponding states are not equivalent. ASAP abstractions, will consider p_1 and p_2 as equivalent, if the required conditions match, and hence, lead to a much more reduction in graph. Figure 7.2 shows the relation among different types of abstractions defined here. □

Finally, we can also prove correctness of our framework, i.e., an optimal algorithm operating on our AND-OR graph converges to the optimal value function.

Theorem 7.2.2. *Optimal value functions $V^*_{Gr}(x)$, $Q^*_{Gr}(u)$, computed by Value Iteration on a reduced AND-OR graph Gr, return optimal value functions $V^*_M(s)$, $Q^*_M(s,a)$ for the original MDP M. Formally, $V^*_{Gr}(x) = V^*_M(s)$, and $Q^*_{Gr}(u) = Q^*_M(s,a)$, $\forall s \in S,\ a \in A$ s.t. $\mu_{\mathcal{E}}(s) = x, \mu_{\mathcal{H}}(s,a) = u$.*

Proof. Given an MDP $M = (S, A, \mathcal{T}, C, \gamma)$, Let Gr be the reduced AND-OR graph constructed from M by replacing the OR nodes in original AND-OR Graph, $s \in S$, with $\mu_{\mathcal{E}}(s)$ and the AND nodes, $p \in P$, with $\mu_{\mathcal{H}}(p)$. We now show, by induction, that the t-step discounted Q-value for any pair, p, is equivalent to the t-step discounted Q-value of the abstract pair, $\mu_{\mathcal{H}}(p)$. The base case for this induction is the initialization step. The Q-value and V-value for each abstract state-action pair and state is initialized to the same value as the ground state-action pair and state respectively. Hence, base-case trivially holds. For the t^{th} time step,

$$Q^t_M(p) = C(p) + \gamma \sum_{s' \in S} T(p, s') V^{t-1}_M(s') \text{[Definition]} \quad (7.2)$$

$$= C(p) + \gamma \sum_{\mu_{\mathcal{E}}(s') \in \mathcal{X}} T(p, \mu_{\mathcal{E}}(s')) V^{t-1}_{Gr}(\mu_{\mathcal{E}}(s')) \text{[Abstraction Defn.]} \quad (7.3)$$

$$= C(\mu_{\mathcal{H}}(p)) + \gamma \sum_{\mu_{\mathcal{E}}(s') \in \mathcal{X}} T(\mu_{\mathcal{H}}(p), \mu_{\mathcal{E}}(s')) V^{t-1}_{Gr}(\mu_{\mathcal{E}}(s')) \text{[Abstraction Defn.]} \quad (7.4)$$

$$= Q^t_{Gr}(\mu_{\mathcal{H}}(p)) \quad (7.5)$$

The two simplification steps above follow from the definitions of AS and ASAP respectively. Since the Q-values do not change when abstracting M to Gr, the value V for any state at any time step (defined as the minimum of all state-action pair Q-values for that state) is also retained and hence, the optimality is preserved.

□

7.3 ASAP-UCT

We now present ASAP-UCT, a UCT-based algorithm that uses the power of abstractions computed via the ASAP framework. Since UCT constructs a finite-horizon MDP tree (see Section 7.1.3), states at different depths have to be treated differently. Therefore, ASAP-UCT tests state equivalences only for the states at the same depth. To compute abstractions over UCT tree, we adapt and extend ideas presented by Jiang et al. [2014].

How to Compute Abstractions: Algorithm 2 gives the pseudocode for computing abstractions using ASAP-UCT . The algorithm takes as input a UCT Search Tree (ST) and outputs an Abstracted Search Tree (AST). Starting from the leaves of the UCT tree, it successively computes abstractions at each level (depth) all the way up to the root. At each level, it calls the functions for computing state and state-action pair abstractions alternately. It is helpful to understand each depth as consisting of a layer of state nodes and a layer of SAP nodes above it. We use $\mu^d_{\mathcal{E}}$ ($\mu^d_{\mathcal{H}}$) to denote the state (SAP) equivalence function at depth d. Similarly, we use S^d to denote the set of states at depth d and P^d to denote the set of SAP nodes at depth d. To keep the notation simple, we overload the equivalence function (map) $\mu^d_{\mathcal{E}}$ to also represent the actual equivalence relationship over state pairs (and similarly for $\mu^d_{\mathcal{H}}$).

Algorithm 3 gives the pseudocode for the computation of state abstractions at a depth using conditions from the previous section. *getPartiallyExplored*(S^d) returns the subset of states at depth d not all of whose applicable actions have been explored. *Apply*(s) returns the set of all the actions applicable in s. Following Jiang et al. [2014], we mark all the partially explored nodes at each depth to be in the same equivalence class. This is an approximation, but is necessary due to the limited information about states available in UCT tree at any given time.

For the states all of whose applicable actions have been explored, we put states (s, s') in the same equivalence class if conditions for state equivalence (Definition 1) are satisfied.

Algorithm 4 gives the pseudocode for computing SAP abstractions. Out(s,a) returns the set of all states sampled in the UCT tree after application of a in s. $\mathcal{T}_X$ and $\mathcal{T}'_X$ are arrays storing the total transition probabilities to each abstract state at next level for the two SAPs being compared, respectively. Two SAP nodes (s, a) and (s', a') are put in same equivalence class if conditions for state-action pair equivalence (Definition 2) are satisfied.

Algorithm 2 Computing Abstract Search Tree

ComputeAbstractSearchTree(SearchTree ST)
  $d_{max} \leftarrow$ getMaxDepth(ST), $\mu_{\mathcal{H}}^{d_{max}+1} \leftarrow \{\}$
  for $d := d_{max} \rightarrow 1$ **do**
    $\mu_{\mathcal{E}}^{d} \leftarrow$ ComputeAS(S^d, $\mu_{\mathcal{H}}^{d+1}$);
    $\mu_{\mathcal{H}}^{d} \leftarrow$ ComputeASAP(P^d, $\mu_{\mathcal{E}}^{d}$);
  $AST \leftarrow$ SearchTree with Computed Abstractions
  Initialize Q-Values of abstract nodes
  return AST

Algorithm 3 Abstraction of States

ComputeAS(States S^d, Eq-Map $\mu_{\mathcal{H}}^{d+1}$)
  $S_L^d \leftarrow$ getPartiallyExplored(S^d)
  $\forall s, s' \in S_L^d, \mu_{\mathcal{E}}^{d}(s) = \mu_{\mathcal{E}}^{d}(s')$ (base case)
  $S_I^d \leftarrow S^d \setminus S_L^d$
  for all $s, s' \in S_I^d$ **do**
    $mapping = True$;
    for all $a \in Apply(s)$ **do**
      If($\nexists a' \in Apply(s')$: $\mu_{\mathcal{H}}^{d+1}(s, a) = \mu_{\mathcal{H}}^{d+1}(s', a')$)
        $mapping = False$;
    for all $a' \in Apply(s')$ **do**
      If($\nexists a \in Apply(s)$: $\mu_{\mathcal{H}}^{d+1}(s', a') = \mu_{\mathcal{H}}^{d+1}(s, a)$)
        $mapping = False$;
    if($mapping$) **then** $\mu_{\mathcal{E}}^{d}(s) = \mu_{\mathcal{E}}^{d}(s')$
  return $\mu_{\mathcal{E}}^{d}$

Updating Q-Values: For all the nodes belonging to the same equivalence class, we maintain a single estimate of the expected cost to reach the goal both in the state as well as state-action layers. In the beginning, Q-values for an abstract node are initialized with the average of the expected cost of its constituents as shown in Algorithm 2. During the backup phase, any Q-value update for a node in the original tree is shared with all the nodes belonging to the same equivalence class in the abstract tree, effectively replicating the rollout sample for every node.

It should be noted that node expansion as well as the random roll out after expansion in the UCT tree are still done in the original flat (unabstracted) space. It is only during the upward update of the Q-value computation where abstractions are used.

When to Compute Abstractions: In order to compute abstractions, we need to construct the sampled UCT tree upto a certain level. But waiting until the full expansion of tree is not helpful, since the tree would already be constructed and we would not be able to use the abstractions. In our approach, we start by allocating a decision time τ for the current step using a dynamic time allocation strategy (described below). Similar to Jiang et al. [2014], we then interleave the steps of tree expansion with abstraction computation. Abstractions are computed after every $\tau/(l+1)$ time units where l is a parameter which can be tuned (we used $l = 1$ in our experiments). Since future expansions might invalidate the currently computed abstractions, every phase of abstraction computation is done over the flat tree. Algorithm 5 provides the pseudocode for the above procedure. ST denotes the search tree in the flat space and AST denotes the search tree in the abstract space.

Algorithm 4 Abstraction of State-Action Pairs

ComputeASAP(States-Action Pairs P^d, Eq-Map $\mu^d_{\mathcal{E}}$)
  for all $p = (s, a)$, $p' = (s', a') \in P^d$ **do**
    $\forall x \in \mu^d_{\mathcal{E}}$, $\mathcal{T}_X[x] = 0$ and $\mathcal{T}'_X[x] = 0$;
    for all $s_i \in Out(s, a)$ **do**
      $\mathcal{T}_X[\mu^d_{\mathcal{E}}(s_i)] += \mathcal{T}(s, a, s_i)$
    for all $s'_i \in Out(s', a')$ **do**
      $\mathcal{T}'_X[\mu^d_{\mathcal{E}}(s'_i)] += \mathcal{T}(s', a', s'_i)$
    if($\forall x \in \mu^d_{\mathcal{E}}$: $\mathcal{T}_X(x) = \mathcal{T}'_X(x)$ & $C(s, a) = C(s', a')$)
      then $\mu^d_{\mathcal{H}}(s, a) = \mu^d_{\mathcal{H}}(s', a')$
  return $\mu^d_{\mathcal{H}}$

Algorithm 5 ASAP-UCT Algorithm

ASAP-UCT(StartNode s_0, NumAbstractions l)
  $ST, AST \leftarrow s_0$ //Single Node Search Tree
  $\tau \leftarrow$ getDecisionTime()
  while τ is not exhausted **do**
    $AST \leftarrow$ ExpandTreeAndUpdateQ(AST)
    After every $\tau/(l + 1)$ time units
      $ST \leftarrow$ getFlatTree(AST)
      $AST \leftarrow$ ComputeAbstractSearchTree(ST)
  return $argmin_{a \in A} Q^*(s_0, a)$ //best action at S_0

Adaptive Planning Time Allocation: We assume that the agent is given an execution horizon (maximum number of decisions to be taken) and time of trial, which represents total planning

time for taking all decisions. This setting has been previously used in the literature for online planning with UCT [Kolobov *et al.*, 2012]. A key meta-reasoning problem is the allocation of available time across various decision steps. Naïvely, we may decide to allocate equal time per decision. However, this can be wasteful in goal-directed settings since the goal can be reached sooner than the execution horizon. In such scenarios, the time allocated for remaining decisions will be wasted. To counter this, we adapt prior work on meta-reasoning algorithms for UCT [Baier and Winands, 2012; Baudiš and Gailly, 2012] to implement an adaptive time allocation strategy. Our approach uses random roll-outs performed during UCT to continually re-estimate the expected number of steps to reach the goal (effective execution horizon).

Specifically, Our allocation algorithm at each step observes all the rollout trajectories and makes a note of the depths at which the goal was reached, if at all. If the goal was not reached, then this value is taken to be the maximum remaining execution horizon (D). This allows the agent to re-estimate its average execution horizon ($\overline{D}$) as predicted through rollouts. If $\overline{D} < D$, then we have the potential to save future unused time. Moreover, as the agent gets closer to the goal it may need less and less planning time, since its planning horizon is reducing.[3] Since MCTS algorithms are linear in the planning horizon (the number of nodes generated at each depth in the rollouts is the same), we use a linearly-decreasing planning time allocation.

To operationalize this we can define a fixed multiplicative constant k, such that our next decision takes $k.\overline{D}$ time, second decision takes $k.(\overline{D}-1)$ time and so on. Since the sum of this time should result in the total remaining time TT_R, we get k to be

$$k.\left(\overline{D}+\overline{D}-1+...+1\right) = TT_R \implies k = \frac{2TT_R}{\overline{D}(\overline{D}+1)}$$

The algorithm computes the time for next decision as $k.\overline{D} = \frac{2TT_R}{\overline{D}+1}$. While this is a good estimate for the next decision, the current execution step may lead it to a really good or really bad outcome, making this average $\overline{D}$ value meaningless for the future. Thus, we discard this $\overline{D}$ estimate after an execution step and re-estimate it based on rollouts in current step, and find a new planning time for the next step and so on. It is important to note that this strategy can be applied to all the variants of UCT.

Efficiently Implementing Abstraction Computation: Computing SAP abstractions naïvely would require $O(n^2)$ comparisons, n being the total number of SAP nodes at each level (we would compare each SAP pair for equivalence). This can become a bottleneck for large problems with large number of states and applicable actions. We instead hash each SAP using its total transition probability to set of next abstract states as well the cost associated with the tran-

[3]Planning horizon determines the depth to which rollouts go, which is an input to the problem, but can reduce if a goal is reached earlier.

sition. Only the pairs falling in the same hash bucket now need to be compared bringing down the complexity to $O(rk^2)$ where r is the number of buckets and k is the number of elements in each bucket. This is crucial for scaling ASAP-UCT as observed in our experiments.

7.4 Experimental Evaluation

Our experiments aim to study the comparative performance of various abstraction approaches in UCT. In Section 7.4.3, we compare ASAP-UCT with vanilla UCT [Kocsis and Szepesvári, 2006], AS-UCT [Jiang *et al.*, 2014], and ASAM-UCT (our novel combination of UCT and Ravindran and Barto [2004]'s ASAM).

7.4.1 Domains

We experiment on three diverse domains in this chapter, Sailing Wind [Kocsis and Szepesvári, 2006; Bonet and Geffner, 2012], Game of Life [Sanner and Yoon, 2011], and Navigation [Sanner and Yoon, 2011]. Game of Life and Navigation domains were used in International Probabilistic Planning Comptetition (IPPC) -2011. We briefly describe these below.

Sailing Wind: An agent in this domain is assigned the task of sailing from one point to another on a grid. The agent can move in any direction to an adjacent cell on the grid, as long as it is not against the stochastic wind direction. The cost at any time step is a function of the agent movement and the wind direction. We report on two instances with grid dimensions 20×20 and 100×100 (#states: $3200, 80000$).

Game of Life: An oracle agent is given the objective to maximize the number of alive cells in a cellular automata environment modeled as a grid. Live cells continue into the next generation as long as there is no overpopulation or underpopulation as measured by the number of adjacent cells. Similarly, a dead cell can become alive in the next generation due to a reproduction between adjacent cells. Additionally, an agent can make exactly one cell live on to the next generation. The dynamics of the game are stochastic in nature, set differently for different cells. The number of live cells determines the reward at every time step. Our empirical results are reported on two IPPC-2011 instances, of dimensions 3×3 and 4×4 (#states: $2^9, 2^{16}$).

Navigation: A robot has to navigate from a point on one side of the river approximated as a grid to a goal on the other side. The robot can move in one of the four possible directions. For each action, the robot arrives in a new cell with a non-zero drowning probability and unit cost. If a robot drowns, it will retry navigating from the start state. We report on two IPPC instances of sizes 20×5 and 10×5 (#states: $100, 50$).

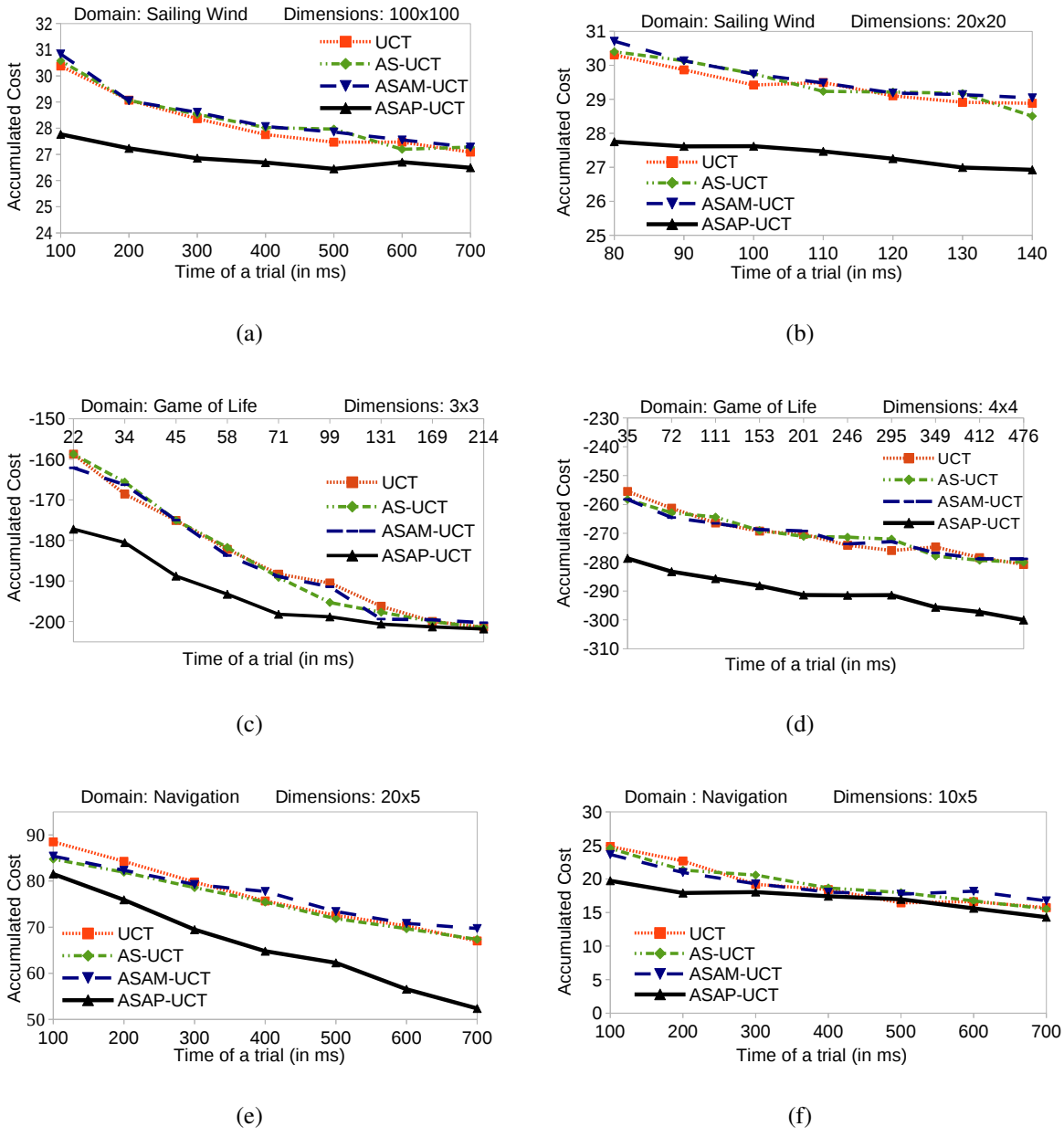


Figure 7.3: ASAP-UCT outperforms all other algorithms on problems from three domains.

7.4.2 Experimental Settings

All our experiments are performed on a Quad-Core Intel i-5 processor. For each parameter configuration we take an average of 1000 trials. We use 100 as the execution horizon for Sailing wind and Navigation domains and 40 for Game of Life domain (as per IPPC instance), although reaching the goal earlier will stop the execution. The planning horizon for a decision is an input to each problem. All UCT rollouts use random actions. The exploration constant K for the UCB equation is set as the negative of the magnitude of current Q value at the node (following [Bonet and Geffner, 2012]). Also, in cases of abstraction computation approaches, we need to set l, the number of times abstractions are computed for each decision. Since, computing abstractions can be expensive, l must be a small number. In our experiments we find that the setting of l as 1 works well for all systems. How to set this automatically is an important question for future work. All algorithms use the adaptive time-allocation strategy, which performs much better than equal time-allocation.

7.4.3 ASAP-UCT vs. Other UCT Algorithms

We compare the four algorithms, vanilla UCT, AS-UCT, ASAM-UCT and ASAP-UCT, in all three domains. For each domain instance we vary the total time per trial and plot the average cost obtained over 1000 trials. As the trial time increases each algorithm should perform better since planning time per step increases. Also, we expect the edge of abstractions over vanilla UCT to reduce given sufficient trial time.

Figures 7.3 shows the comparisons across these six problems in three domains. Note that time taken for a trial also includes the time taken to compute the abstractions. In almost all settings ASAP-UCT vastly outperforms both UCT, AS-UCT and ASAM-UCT. ASAP-UCT obtains dramatically better solution qualities given very low trial times for Sailing Wind and Game of Life, incurring up to 26% less cost compared to UCT. Its overall benefit reduces as the total trial time increases, but almost always it continues to stay better or at par. We conducted one tailed student's t-test and found that ASAP-UCT is statistically significantly better than other algorithms with a significance level of 0.01 (99% confidence interval) in 41 out of 42 comparisons (six graphs, 7 points each). The corresponding error bars are very small and hence, not visible in the figures.

Discussion: We believe that the superior performance of ASAP-UCT is because of its effectiveness in spite of noise in sampled trees. AS and ASAM conditions are strict as they look for all pairwise action equivalences before calling two states equivalent. Such complete set of equivalences are hard to find in UCT trees where some outcomes may be missing due to sampling. ASAP-UCT, in contrast, can make good use of any partial (SAP) equivalences found. ASAM's performance does not improve over AS probably because its gain due to symmetries

is undermined by increase in abstraction computation time.

We also experimented with the Canadian Traveler Problem (CTP) [Bonet and Geffner, 2012] but found that no abstraction algorithm gives performance gains over vanilla UCT. For CTP, we believe this is due to lack of symmetries in the domain. This suggests that some domains may not be that amenable to abstractions which is not too surprising.

Effect of Approximation Parameters: In our experiments, AS-UCT does not always perform better than UCT, rather, it often underperforms. This is surprising and contradicts previous observations [Jiang *et al.*, 2014], which were based upon a single, deterministic domain of Othello. This could be due to the fact that our experiments set the parameters $\epsilon_{\mathcal{T}}$ and $\epsilon_{\mathcal{C}}$ (see Section 7.1.4) zero for abstraction frameworks. However, we find that even after incorporating a range of approximation parameters in AS-UCT, ASAP-UCT without those parameters continues to perform significantly better.

7.5 Conclusion and Future Work

This chapter develops a novel class of state-action pair (SAP) abstractions, which generalizes and extends past work on abstractions in MDPs. SAP abstractions find more symmetries in a domain compared to existing approaches and convert an input MDP into a reduced AND-OR graph. We present a new algorithm, ASAP-UCT, which computes these abstractions in an online UCT framework. ASAP-UCT obtains significantly higher quality policies (up to 26% reduction in policy costs) compared to previous approaches on three benchmark domains. We have released our implementation for a wider use by the research community. It would be interesting to extend these ideas to domains expressed in a factored or first-order representation such as RDDL [Sanner, 2010].

In the next chapter, we study a more advance implementation of abstractions in UCT, called On-the-Go Abstractions (OGA) UCT which computes abstractions as the tree is built. We compare OGA-UCT with ASAP-UCT on these domains as well as some additional domains illustrating that OGA-UCT is robust across variety of domains.

Chapter 8

On-the-Go Abstractions in UCT (OGA-UCT)

As shown in previous chapter, the performance of MCTS algorithms can be further improved by incorporating domain abstractions. Original MCTS algorithms explore the flat state space, which can be wasteful since many states are actually symmetric and need not be considered separately. MCTS algorithms with abstractions such as AS-UCT [Jiang *et al.*, 2014] and ASAP-UCT [Anand *et al.*, 2015a] automatically compute (approximate) symmetric nodes (states, state-action pairs) in the search tree. They aggregate such nodes into an abstract node, thus reducing the subsequent UCT planning time considerably.

Both these algorithms alternate between two phases. One phase consists of an abstraction computation routine that uses the existing UCT tree to induce groups of symmetric nodes. These nodes are aggregated to construct an abstract search tree. The second phase is the (modified) UCT algorithm, which is run as per original UCT in the beginning, but is modified to incorporate the abstractions after the abstraction routine has been run at least once.

We believe that these algorithms do not achieve the full potential of abstractions since they do not capture the true characteristics of symmetry aware learning described in Chapter 1. Since abstractions are computed on a sampled tree, they are approximate. Erroneous abstractions computed as part of one batch of abstraction computation may get corrected only after a full phase of modified UCT – this wait could severely impact the solution quality. Moreover, while UCT prefers some states over others (due to UCB exploration rule), these algorithms treat all nodes at par while computing abstractions. This wastes valuable time on less important nodes, which likely have limited impact on further planning.

To address this issue, we first revisit the characteristics of symmetry aware inference with respect to planning algorithms. We first analyze the space of algorithmic design choices for MCTS algorithms with domain abstractions. An algorithm can be phased or incremental, ab-

straction computation may be done uniformly on all states or adaptively, and so on. We find that existing algorithms have complimentary strengths and weaknesses. In response, we propose *On-the-Go Abstractions* (OGA), which incorporates the best-of-all-worlds design choices and incorporates all the desired characteristics of symmetry aware decision making and inference.

On-the-Go Abstractions (OGA) is a novel (non-phased) approach in which abstraction computation is tightly integrated into MCTS. In line with previous work, we implement these on top of UCT and name the resulting algorithm OGA-UCT. OGA-UCT has several desirable properties. First, it completely eliminates the expensive batch abstraction computation routine. OGA-UCT is *incremental* in computing abstractions, i.e., as the tree gets built it is seamlessly reduced by abstraction spot checks. Second, this allows new knowledge, either for correcting old abstractions or finding new ones, to be useful without a significant wait. This keeps the reduced tree as accurate as possible leading to better quality solutions. Finally, where to compute abstractions is also *adaptive* – it is guided by the UCB exploration function, thus focusing computation on the more important part of the search space.

We experimentally compare OGA-UCT[1] against ASAP-UCT (the state-of-the-art abstraction-based UCT solver) as well as the vanilla UCT algorithm. We find that across a suite of planning competition and other MDP domains, OGA-UCT performs better or at par with the best technique on each domain obtaining up to 28% solution quality improvements.

We begin by discussing the various design choices for abstraction algorithms with respect to planning. We also characterize the existing work with respect to the design choices. Next, we describe our main contribution of On-the-Go Abstractions in UCT (OGA-UCT) which incorporates the best of these design choices. Lastly, we discuss some key characteristics of OGA-UCT giving some guarantees on its performance. Lastly, we illustrate the effectiveness of OGA-UCT on some benchmark domains and finally we conclude this part of thesis.

8.1 Design Choices for Abstraction Algorithms

Abstraction computation and tree construction mutually depend on each other. If the tree is complete, the abstractions computed will be accurate; if more abstractions are known, a more reduced tree can be built saving further computation. This interplay between MCTS and abstraction computation is relatively novel and admits several important algorithmic design choices. We briefly describe these choices and place existing algorithms in this context. We find that ASAP-UCT and PARSS have complimentary strengths and weaknesses; our proposal OGA-UCT combines the best of both worlds. Table 8.1 summarizes this analysis.

[1] https://github.com/dair-iitd/oga-uct

Design choice	AS-UCT	ASAP-UCT	PARSS	OGA-UCT
batch vs. incremental	batch	batch	incremental	incremental
uniform vs. adaptive	uniform	uniform	adaptive	adaptive
progressive vs. split-merge	split-merge	split-merge	progessive	split-merge
unit of abstraction: states or SAP	state	SAP	state	SAP
convergence to flat or aggregate nodes	aggregate	aggregate	flat	aggregate

Table 8.1: Properties and design choices for MCTS algorithms computing abstractions

Batch *vs.* Incremental Computation: A key design decision is "when to initiate the abstraction computation routine"? AS-UCT and ASAP-UCT are batch-style algorithms that clearly demarcate the tree construction phase from the abstraction computation phase. Abstraction computation is treated as an independent procedure that is called periodically at specific time points. Moreover, they throw away previous abstractions, and recompute them from scratch.

An alternative is an *incremental algorithm* that tightly couples abstraction computation with tree construction. For example, PARSS adds nodes to its tree *via* abstraction refinement. It only changes abstractions of a few nodes locally and does not compute everything from scratch.

We point out that batch-style algorithms are expensive and also suffer from *stale* abstractions – it can take a while to recover from erroneous abstractions (or good abstractions missed). This can potentially lead to worse solutions because one state's Q-value may be erroneously and repeated transferred to another state. Incremental algorithms have the danger of spending too much time computing abstractions and little time using them in planning.

Uniform *vs.* Adaptive Abstractions: The success of MCTS is, in part, due to adaptive sampling, which zooms in on important parts of search tree (e.g., using UCB formula). The same principle is applicable to choosing the subset of nodes on which to attempt abstraction computation. AS-UCT and ASAP-UCT treat all search nodes in the tree equally, wasting time on nodes that might be rarely visited. PARSS performs an explicit node selection and can incorporate a measure of importance in choosing nodes for refinement.

Progressive Refinement *vs.* Split-Merge: Abstraction computation may be monotonic. For example, it may start with the flattest nodes and progressively merge nodes to create aggregate super-nodes. Or it may start with the coarsest supernodes and progressively split those to create finer refinements. The former approach will create increasingly reduced trees whereas the latter will create increasingly finer trees. A very different solution is to allow the algorithm to split or merge as necessary based on new information. We call these *Split-Merge* abstractions. A newly added node or edge in the tree may identify that two previously abstracted nodes were, in fact, not symmetric and should be split, or that two nodes now appear symmetric, and can be merged. This allows the maximum use of available information in creating as accurate abstractions as possible at a given time.

PARSS uses progressive refinement. AS-UCT and ASAP- UCT, due to their complete recomputation of abstractions, allow both splits and merges. We know of no work that uses progressive abstractions, in part because there are not many different abstraction procedures available.

Convergence Conditions & Units of Abstraction: A related question is "what does the final tree converge to?" Does it converge to a completely flat search space? Or does it converge to a reduced space? PARSS splits abstract states somewhat arbitrarily and, in the limit, converges to a perfectly flat search tree. On the other hand, AS- and ASAP-UCT use the definitions of node symmetries to converge to a reduced search space, which is guaranteed to be an accurate reduction of the MDP in the limit. Of these two, AS-UCT abstracts only the states, whereas ASAP-UCT can abstract state-action pairs as well, leading to more compression and runtime savings as shown before [Anand *et al.*, 2015a]. PARSS operates only on states and not on state-action pairs.

Overall, we find that both PARSS and ASAP-UCT have some desirable and some undesirable properties. PARSS is an incremental and adaptive algorithm, but its abstractions are ad-hoc and do not discover accurate domain symmetries. It also reduces to a flat space in the limit, and only aggregates states. ASAP-UCT, on the other hand, abstracts SAPs but is a batch-style algorithm and uniformly computes abstractions on the whole tree. Our proposal, OGA-UCT, combines the strengths of both algorithms: it computes SAP abstractions in an incremental and adaptive manner.

8.2 OGA-UCT

In this section, we describe our algorithm OGA-UCT. Our algorithm is best understood in terms of the construction of the original UCT tree. The UCT computation can be broadly divided in two key phases 1) Sampling of a trajectory 2) Random rollout from a newly discovered leaf node. In OGA-UCT, during the first phase, along with sampling of the trajectory, an abstraction for each node is also maintained *on- the-go*. Abstraction for any node is computed using the recursive updates similar to the ones used by ASAP-UCT (see the Background section). But the key difference is that instead of doing the batch computation uniformly for each node, we do it incrementally and in an adaptive manner. Each node has an associated recency count, which stores the number of times the node was visited after its abstraction was last updated. If the recency count reaches a pre-decided threshold K, we re-compute the abstraction for this node and set the recency count back to 0. In the second phase when a rollout is performed, we initialize the abstraction of the newly created leaf and set its recency count to 0. Any Q-value updates in the UCT tree are now done over the abstract nodes rather than the original

nodes. This effectively means that we can utilize the information from a single rollout for all the nodes falling under the same abstraction. Since abstractions at a certain depth depend on the abstractions in the tree below, it may happen that when a node's abstraction changes, there is a change in the abstraction of its ancestor nodes. Therefore, any change in the abstraction of a node at depth d is propagated all the way up to the root of the tree, recomputing abstractions as necessary.

Our entire algorithm can be divided in four parts. 1) OGA-UCT procedure (Algorithm 6) which maintains the outer loop for sampling trajectories and is similar to the original UCT. 2) SampleTrajectory procedure (Algorithm 7) for sampling a single trajectory which is the key for OGA-UCT. 3) Procedures (Algorithm 8) for updating state and SAP abstractions and percolating the changes recursively to the ancestors once the recency count is reached for a node. 4) Procedures (Algorithm 9) for computing the current abstraction of a state and an SAP node. We next describe each one of them in detail.

Algorithm 6 OGA-UCT

```
1:  procedure OGA-UCT(S_0, N, K, Horizon)
2:      T ← EMPTYTREE()
3:      global K, Horizon, T
4:      Add state node (S_0, 0) to tree T
5:      INITIALIZESTATEABSTRACTION(S_0, 0)
6:      i ← 0
7:      while i < N do
8:          SAMPLETRAJECTORY(S_0, 0)
9:          i ← i + 1
10:     return SELECTBESTACTION(S_0, 0)
```

OGA-UCT (Algorithm 6): The procedure OGA-UCT is very similar to the traditional UCT algorithm. We start from a root node $(S_0, 0)$ and then sample the required number of trajectories (N). Note that we need to initialize the abstraction of the root node (line 5). Horizon determines the depth until which the tree is expanded. K controls the frequency for computing abstraction; abstractions are re-computed when the recency count of a node becomes equal to K.

Sampling Trajectory (Algorithm 7): This is the main procedure of our algorithm. Lines 2-6 check the base condition for stopping a trajectory. Lines 7-11 add a newly discovered leaf node to the tree, create an abstract node for it(initialize its abstraction) and perform a rollout. If the procedure comes to line 12, we have not discovered a new leaf node yet. Line 12 selects an action based on the UCB rule. Here, Q-Values and Counts in UCB formula are obtained from Q-Values and Counts of corresponding abstract node. In lines 13-17, we add a newly discovered SAP node to the tree, create a new abstract node for it (initialize its abstraction) and set the recency count as 0. Lines 18-19 sample a new state node based on the chosen action and

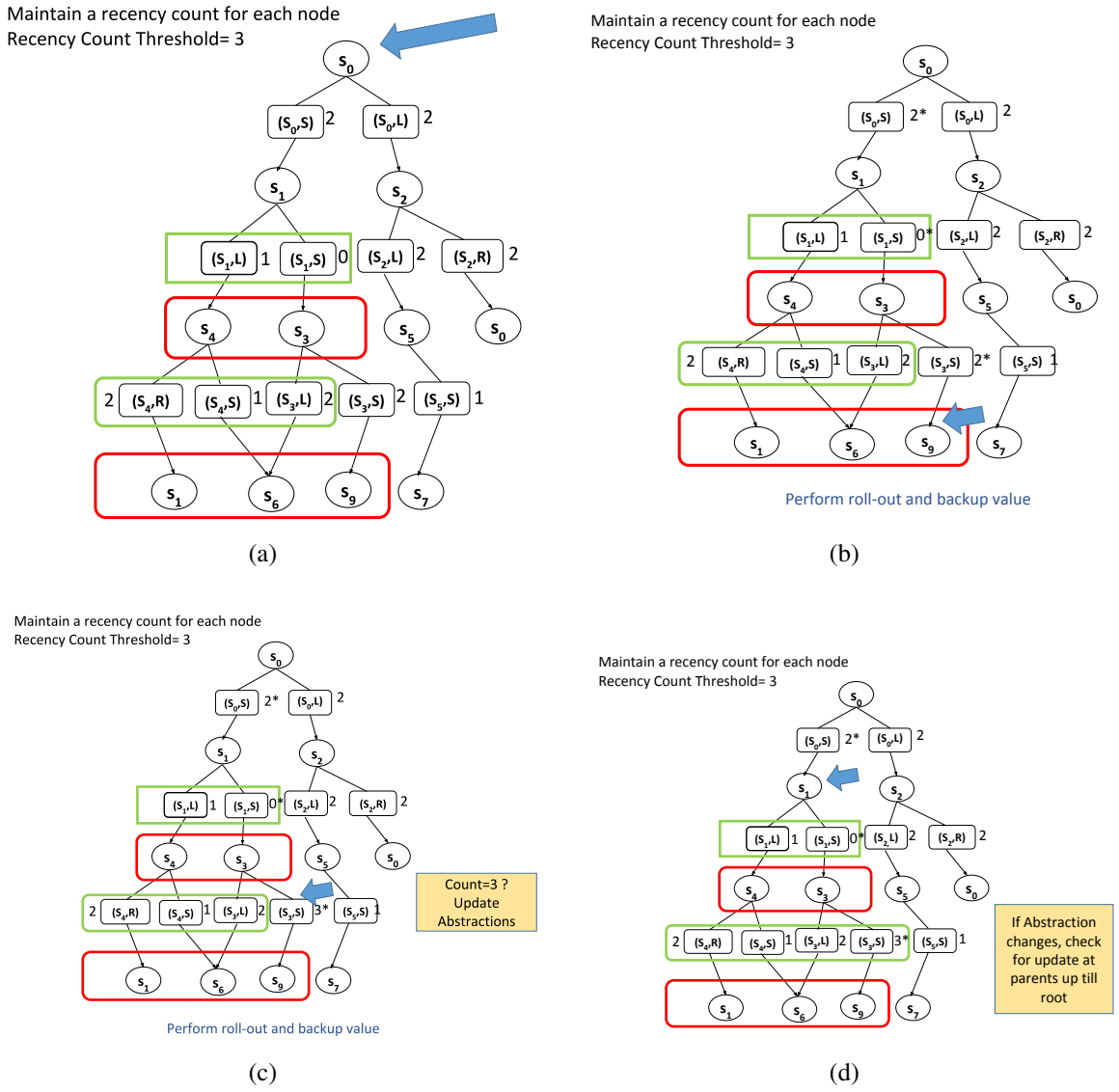


Figure 8.1: OGA-UCT in execution over a partially built MCTS tree at different stages. Number in parenthesis along each state gives recency count of that state. Rectangular boxes encapsulate symmetric elements at a particular level.

recursively call SAMPLETRAJECTORY. Lines 20-23 take care of maintaining the recency count and calling update abstractions if the count has reached the threshold K. Finally, lines 24-25 update counts and Q-Values for abstract node corresponding to (s, a, d). It is insightful to note that if we remove the lines for computing abstractions and maintaining the recency count (lines 9,15-16,20-23), the procedure becomes identical to what standard UCT would do with lines 24, 25 updating Q-Value and Count of ground node.

Algorithm 7 Sample Trajectory in UCT

```
1:  procedure VAL = SAMPLETRAJECTORY(s, d)
2:      if terminal(s) then
3:          return −reward(s)
4:      else if d == Horizon then
5:          return 0
6:      if (s, d) is not in tree T then
7:          Add state node (s, d) to tree T
8:          INITIALIZESTATEABSTRACTION(s, d)
9:          return GETROLLOUT(s, d)
10:     a ← SELECT-UCB-ACTION(s, d)
11:     if (s, a, d) is not in tree T then
12:         Add SAP node (s, a, d) to tree T
13:         INITIALIZE-SAP-ABSTRACTION(s, a, d)
14:         RecencyCount[s, a, d] ← 0
15:     s' ← SAMPLE(s, a)
16:     newVal ← SAMPLETRAJECTORY(s', d + 1)
17:     RecencyCount[s, a, d] + +
18:     if RecencyCount[s, a, d] == K then
19:         UPDATE-SAP-ABSTRACTION(s, a, d)
20:     INCREMENTCOUNT(μ^d_H(s, a))
21:     UPDATEQ(μ^d_H(s, a), newVal)
22:     return newVal
```

Updating Abstractions (Algorithm 8): We update the abstractions for nodes whose recency count has reached K. There are two different procedures, one for updating SAP abstractions and other for updating state abstractions. The symbol $\mu^d_{\mathcal{E}}$ ($\mu^d_{\mathcal{H}}$) denotes the mapping from a state (SAP) node to its abstraction, as defined in the Background section. In the procedure for updating SAP abstractions, we first reset the recency count. Line 3 retrieves the current abstraction (v) of this node and line 4 computes the new abstraction (u). In lines 5 to 9, if the new abstraction is different from the old abstraction (i.e. $u \neq v$), the data of the old abstract node (v, d) and new abstract node (u, d) needs to be updated (since a ground node is leaving one and entering the other).The details about updating data(Q-Values and Counts) in line 7 is discussed in detail in sub-section on Maintaining Q-Values and Counts later. In line 8, update

abstraction is then called on the parent node (s, d) to propagate up the influence of change in abstraction of node (s, a, d).

The procedure for updating state abstraction (starting line 12) is similar. Since, a state node could have resulted from multiple (s', a') nodes, abstractions have to be updated for each one of them (lines 17 - 19). Note that we do not need to maintain any data for state nodes. The V-value of a state node is not explicitly required in UCT, and, the count of a state node is obtained by summing up the count of its children SAP nodes.

Figure 8.1 shows partial MCTS trees at different stages of execution for a simple example. The first part (a) shows the state at the beginning of a trajectory while last part (d) indicates how abstractions are updated in the ancestors depending on whether abstraction has changed or not at the current node. The intermediate stages of reaching the leaf node and updation based on recency count are shown in (b) and (c).

Algorithm 8 Update Abstractions

```
1:  procedure UPDATE-SAP-ABSTRACTION(s, a, d)
2:      RecencyCount[s, a, d] ← 0
3:      v ← μ^d_H(s, a)
4:      u ← COMPUTE-SAP-ABSTRACTION(s, a, d)
5:      if u ≠ v then                                  ▷ if abstraction changed
6:          μ^d_H(s, a) ← u
7:          Update data of (v, d) and (u, d)
8:          UPDATE-STATE-ABSTRACTION(s, d)
9:
10: procedure UPDATE-STATE-ABSTRACTION(s, d)
11:     y ← μ^d_E(s)
12:     x ← COMPUTE-STATE-ABSTRACTION(s, d)
13:     if x ≠ y then                                  ▷ if abstraction changed
14:         μ^d_E(s) ← x
15:         for (s', a') ∈ Parents[s, d] do
16:             UPDATE-SAP-ABSTRACTION(s', a', d − 1)
```

Computing Abstractions (Algorithm 9): There are two different procedures for computing abstractions: one for SAP nodes and one for state nodes. Let us look at the case of SAP nodes (Compute SAP Abstractions) first. Recall that $\mathcal{X}$ denotes the set of abstract state nodes at a given depth ($d+1$ in this case). Let $T_{\mathcal{X}}$ denote the vector of transition probabilities to abstract state nodes in depth $d + 1$ from SAP node (s, a, d). Initially, these transition probabilities are set to 0 (line 2). In lines 3-5, $T_{\mathcal{X}}$ is populated by iterating over each node $(s', d + 1)$ in the next level, finding its abstraction $\mu_{\mathcal{E}}^{d+1}$ and adding the transition probability $\mathcal{T}(s, a, s')$ to the corresponding element of vector $T_{\mathcal{X}}$. We also, maintain a hash map $M_{\mathcal{X}}^{d}$ which stores the mapping from the pairs of form $[T_{\mathcal{X}}, C(s, a)]$ to an abstract SAP node at depth d. If the key $[T_{\mathcal{X}}, C(s, a)]$

already exists in $M^d_{\mathcal{X}}$, the desired abstraction of (s, a, d) is the corresponding value (w in the line 7). Else we create a new SAP abstraction (u) containing (s, a, d) as the only node and add it to map $M^d_{\mathcal{X}}$ (lines 10,11).
Similarly for computing abstraction of state node (s) at depth (d), we maintain a set $\mathcal{J}_{\mathcal{U}}$ consisting of abstract SAP nodes of form (s, a, d) at depth d. Hash map $M^d_{\mathcal{U}}$ stores the mapping from set of form $\mathcal{J}_{\mathcal{U}}$ to the state abstractions at depth d. If $\mathcal{J}_{\mathcal{U}}$ already exists in hash-map, then we return the corresponding state abstraction else a new abstraction is created and returned.

Maintaining Q-Values and Counts: The ideal design principle would be to set the Q-value of an abstract SAP node as the weighted average of Q-values of the constituent nodes, and the associated count to be the sum of the constituent counts. Unfortunately, operationalizing this (at the time of abstraction change) requires significant bookkeeping. Hence, we maintain data(Q-values and Counts) only for abstract SAP nodes instead of individual nodes.
Let the abstraction of an SAP node (s, a, d) change from v to u. Let the original counts for v and u be given by C_v and C_u respectively and new counts be given by C^{new}_v and C^{new}_u respectively. The new C^{new}_v and C^{new}_u can be computed in the following manner:

$$C^{new}_v = C_v - \frac{C_v}{|v|}, C^{new}_u = C_u + \frac{C_v}{|v|} \tag{8.1}$$

Here, $|v|$ is number of ground nodes present in abstract node v. Intuitively, since we maintain count only for abstract SAP nodes, we take proportionate count from v and add it to u. Next, let Q_v and Q_u denote the original Q-values for v and u respectively. Then, the new Q-values, Q^{new}_v and Q^{new}_u can be computed by:

$$Q^{new}_v = Q_v; Q^{new}_u = \frac{C_u \cdot Q_u + \frac{C_v}{|v|} \cdot Q_v}{C_u + \frac{C_v}{|v|}} \tag{8.2}$$

Note that Q_v remains unchanged and Q_u is updated by taking a weighted average between Q_v and Q_u.

Pruned OGA-UCT: Many applications need to deal with domains with a very high stochastic branching factor. In such cases, OGA-UCT will spend a significant amount of time in computing SAP abstractions since large number of transitions need to be considered. In order to ameliorate this problem, while constructing transition tables, we do not consider nodes with very low transition probabilities. Assume that we need to compute abstraction of an SAP node (s, a, d). Let $\mathcal{T}^*_s = max_{s'}\mathcal{T}(s, a, s')$ where the maximization is taken over the states s' such that $(s', d+1)$ is a node present in the UCT tree. Then, during abstraction computation, we only consider those nodes $(s', d + 1)$ in the tree whose transition probability $\mathcal{T}(s, a, s') \geq \alpha * \mathcal{T}^*_s$. Here α is a constant s.t $0 \leq \alpha \leq 1$. In addition, we achieve this without a complete linear

Algorithm 9 Compute-Abstraction

```
1:  procedure COMPUTE-SAP-ABSTRACTION(s, a, d)
2:      ∀x ∈ 𝒳 : 𝒯_𝒳[x] = 0
3:      for (s', d + 1) in Tree T do
4:          𝒯_𝒳[μ_ℰ^{d+1}(s')] += 𝒯(s, a, s')
5:      if [𝒯_𝒳, C(s, a)] exists in M_𝒳^d then
6:          w = M_𝒳^d[𝒯_𝒳, C(s, a)]
7:          return w
8:      u ← CREATE-NEW-SAP-ABSTRACTION(d)
9:      Insert [𝒯_𝒳, C(s, a)], u in M_𝒳^d
10:     return u
11:
12: procedure COMPUTE-STATE-ABSTRACTION(s, d)
13:     𝒥_𝒰 ← {}
14:     for a ∈ 𝒜 do
15:         𝒥_𝒰 ← 𝒥_𝒰 ∪ {μ_ℋ^d(s, a)}
16:     if 𝒥_𝒰 exists in M_𝒰^d then
17:         z = M_𝒰^d[𝒥_𝒰]
18:         return z
19:     x ← CREATE-NEW-STATE-ABSTRACTION(d)
20:     Insert 𝒥_𝒰, x in M_𝒰^d
21:     return x
```

scan over nodes at depth $d+1$ by a small optimization. We call the resulting algorithm Pruned OGA-UCT . Note that Pruned OGA-UCT defaults to OGA-UCT when $\alpha = 0$. As demonstrated by our experiments, Pruned OGA-UCT (for a suitably chosen value of α) is competitive with OGA-UCT while giving improved performance on domains with high branching factor.

Implementation Details: Whenever the abstraction of a state or SAP node changes, we might need to update the abstraction of its ancestors continuing all the way up to the root of the tree. Since UCT tree is a Directed Acyclic Graph, a single update of abstraction at a node may result in multiple such updates on an ancestor through different paths. In our implementation, we carefully avoid these multiple updates by performing them in a breadth first manner.

8.3 Characteristics of OGA-UCT

OGA-UCT has several desirable theoretical and algorithmic properties. We first prove that it converges to the optimal solution in the limit of infinite samples.

Theorem 8.3.1. *Given an MDP $M = (S, A, \mathcal{T}, C, H)$, the value function computed by OGA-UCT for the abstract node containing a state s at depth d converges to the value function computed by UCT for state s, as number of trajectories $N \to \infty$ i.e $\forall s \in S \ \ \forall d \leq H$*

$$\lim_{N\to\infty} V^N_{OGA}(\mu^d_{\mathcal{X}}(s), d) = \lim_{N\to\infty} V^N_{UCT}(s, d)$$

Here V^N_{OGA} and V^N_{UCT} denote the value functions computed by OGA-UCT and UCT, respectively.

Proof. The proof will proceed in two parts.

Part 1 (Sound abstractions lead to correct values): We say that abstractions computed by OGA-UCT are sound if two state (SAP) nodes that fall in the same abstraction under OGA-UCT, also fall in the same abstraction using the ASAP definition of abstractions in the ground finite-horizon MDP. Further, ASAP abstractions are guaranteed to have identical Q-values and V-values [Anand *et al.*, 2015a]. Therefore, applying UCT on such a sound abstract tree will result in simulation of ground UCT which will converge to the optimal values in the limit.

Part 2 (OGA-UCT leads to sound abstractions): Let $N_{(s,d)}$ denote the number of trajectories passing through the state node (s, d). We say that (s, d) is visited sufficiently if $N_{s,d} \to \infty$ when $N \to \infty$. We define sufficient visits for SAP nodes in a similar manner. We will inductively prove that the abstractions in a sub-tree rooted at the state node (s, d) are sound if (s, d) is visited sufficiently. Let the D be the maximum depth in the tree. We will prove the claim by using backward induction from D going all the way to 0. Clearly, the claim is true for $d = D$ (leaves of the tree). Let us assume that it holds for state nodes at depth $d + 1$. We will now prove it for depth d. Consider a state node (s, d). Since (s, d) is visited sufficiently, all its children SAP nodes must be in the tree (due to the exploration in UCT). Let (s, a, d) be one such child node. Let $u \in \mathcal{U}$ denote the abstract node corresponding to (s, a, d). Then, again due to exploration in UCT, $\exists$ at least one node (s', a', d) with abstraction u which is visited sufficiently (all the nodes in an abstraction can not be starved). Since an SAP node samples its child nodes based on the transition probabilities, all its children $(s'', d + 1)$ must be in the tree, must be visited sufficiently, and hence, should have sound abstractions using the inductive hypothesis. Combining the above two facts, we can say that (s', a', d) will also have a sound abstraction with any node in u. This implies that (s, a, d) (child of (s, a)) will have a sound abstraction. But since (s, a, d) was arbitrary, all the children of (s, d) must have sound abstractions. Combining this with the fact that all the children of (s, d) are already in the tree, (s, d) must also have sounds abstraction. Finally, since the root is visited sufficiently by the statement of the theorem, all the abstractions in the UCT tree must be sound in the limit. Hence, proved. □

□

We now place OGA-UCT into the context of our previous analysis of algorithmic design choices. OGA-UCT is *incremental* – it tightly integrates the abstraction computation routine

with tree construction and makes only local changes in abstractions. Its focus on where to recompute abstractions is *adaptive* – it recomputes abstractions for frequently visited nodes much more often than others, thereby effectively utilizing the abstraction computation time on important parts of the search space. OGA-UCT can both *split* and *merge* existing abstractions, allowing itself to maintain as accurate a domain abstraction as possible given current knowledge. Last but not the least, it abstracts both states and state-action pairs, and in the limit converges to a reduced search space.

8.4 Experiments

We compare the performance of OGA-UCT with ASAP-UCT, the state-of-the-art UCT-based algorithm that employs domain abstractions. Previous work [Anand *et al.*, 2015a] showed that it obtains better performance than AS-UCT and variants. We also compare OGA-UCT with vanilla UCT to assess the overall value of abstractions in UCT. In addition, we also do a sensitivity analysis for different values of K in OGA-UCT. We illustrate our experiments on six popular MDP planning domains from literature and International Probabilistic Planning Competition (IPPC).

8.4.1 Experimental Settings

We implement OGA-UCT on the top of MDP Engine,[2] the UCT implementation from Bonet & Geffner [2012] in C++. ASAP-UCT[3] is also implemented over the same codebase. This makes the runtime comparisons between the three algorithms meaningful.

In spite of having access to PARSS source-code [Hostetler *et al.*, 2015], we could not complete a meaningful comparison. PARSS is implemented in Java giving it a rather different execution profile. Its basis in sparse sampling and its unique style of abstractions makes PARSS very different in nature compared to UCT implementing AS, ASAP, or OGA. Because it starts with a tree pre-built up to the planning horizon, we do not expect PARSS performance to match up to UCT-based algorithms for large horizons – original PARSS experiments are on horizons of up to five. Their results are also on significantly modified versions of original benchmarks. We leave empirical comparison with PARSS to future work.

In our experiments, all algorithms were given equal time per decision, computed as total planning time divided by the execution horizon. UCT rollouts employed a random base policy. We set the exploration constant for UCB rule to be the absolute value of current Q-value of node, as per recommendations in [Bonet and Geffner, 2012]. The l value in ASAP-UCT, which

[2]Available at `https://code.google.com/p/mdp-engine/`
[3]Downloaded from `https://github.com/dair-iitd/asap-uct`

determines the number of abstraction phases per decision, was set to 1 as done in previous chapter. We tried various K values and found OGA-UCT performance to vary slightly in a few domains, with no clear winner. We choose K to be 3 in all experiments for its marginally better overall performance.

All our experiments are performed on Intel Quad core i-7 system. For all the domains, we use a planning horizon of 50 and execution horizon of 100, i.e, a total of 100 decisions are taken per problem, and each decision is taken with a maximum lookahead of 50.

8.4.2 Domain Descriptions

In this case, we employ three more domains of Sysadmin, Academic Advising and Race Track in addition to Sailing Wind, Game of Life and Navigation domains used in previous chapter of ASAP-UCT. We also experiment with 5 instances of each domain. We next describe the new domains briefly and also, describe the characteristics of the instances used for each domain.

Race Track: This traditional MDP [Barto *et al.*, 1995; Bonet and Geffner, 2012] consists of a race track with acceleration and deceleration in either direction as the available actions. We test on six different race-tracks as implemented in Bonet's MDP Engine.

SysAdmin: We use IPPC 2011 [Sanner and Yoon, 2011] version of this traditional domain [Guestrin *et al.*, 2003]. The agent is a network administrator, and can reboot machines. Machines can probabilistically crash based on the number of alive machines that are neighbors. We test on six problems – ring, hub, and line topologies with 10 and 15 machines each.

Academic Advising: This domain from IPPC 2014 requires an agent to pass various courses that have pre-requisite relations [Guerin *et al.*, 2012]. Good grades in pre-requisites makes it more likely to pass a course. We test on 4 different IPPC instances for this domain.

Navigation: This domain is similar to the domain description defined in Section 7.4.1. We test on five IPPC 2011 instances of varying sizes.

Sailing Wind: This domain was also used in experiments in Section 7.4.1. We use 5 different instances having grid sizes $\{10, 15, 20, 25, 30\}$ in our tests.

Game of Life: Similar to Navigation and Sailing Wind, this domain was also used in Section 7.4.1. The domain has a very high branching factor. We test on two instances for grid sizes 3×3, 4×4, and 5×5 with uniform noise probabilities in each cell.

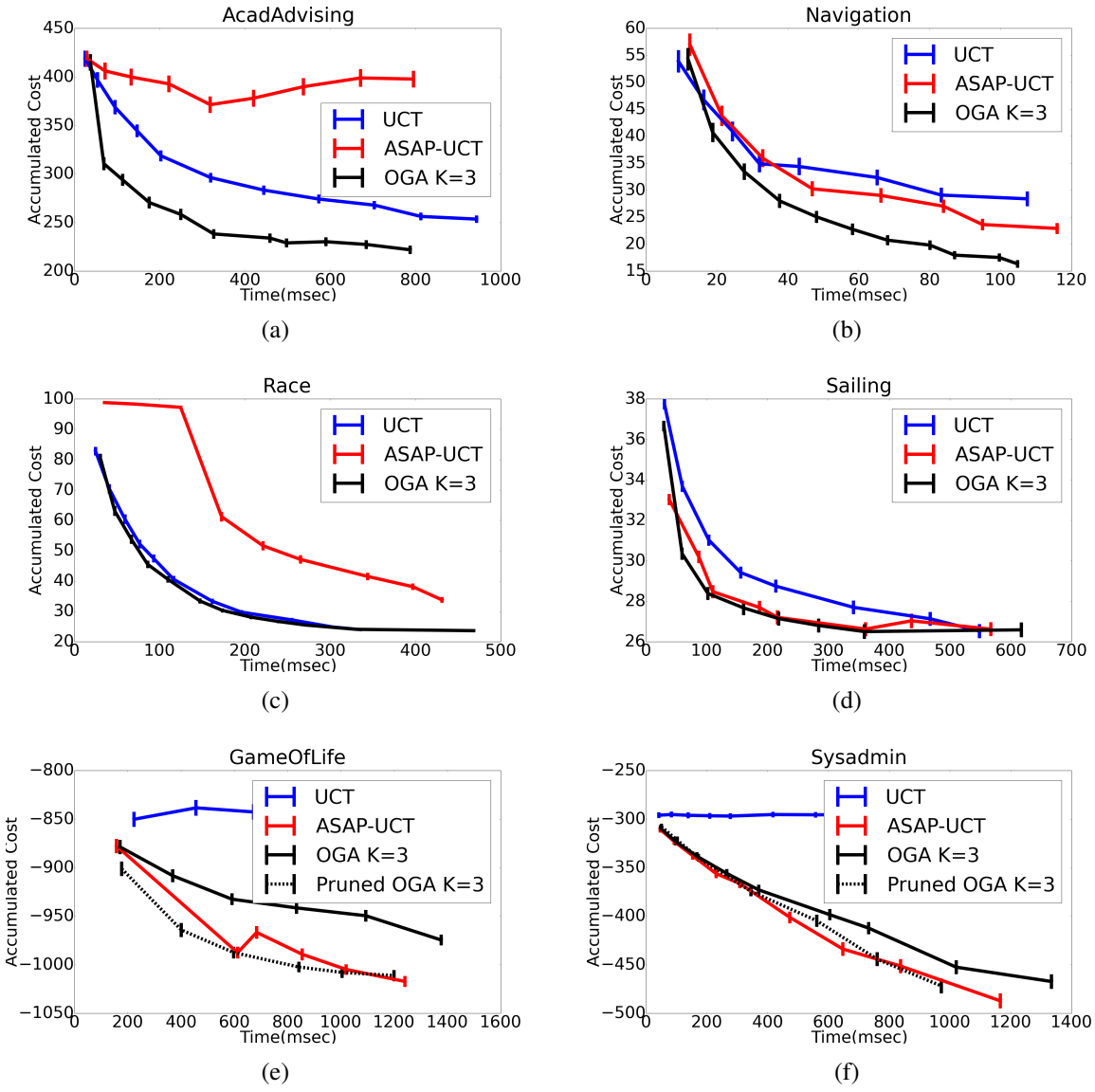


Figure 8.2: OGA-UCT performs better or at par with ASAP-UCT and UCT for most of the domains

Domains	UCT	ASAP-UCT	OGA (K=3)	Pruned ASAP	Pruned OGA (K=3)
Acadadvising	0.48 ± 0.07	0.23 ± 0.16	**0.89 ± 0.03**	0.23 ± 0.16	**0.88 ± 0.03**
Navigation	0.41 ± 0.16	0.45 ± 0.13	**0.57 ± 0.13**	0.45 ± 0.12	**0.54 ± 0.08**
RaceTrack	**0.84 ± 0.13**	0.38 ± 0.09	**0.84 ± 0.13**	0.46 ± 0.11	**0.82 ± 0.14**
Sailing Wind	0.61 ± 0.03	**0.80 ± 0.05**	**0.82 ± 0.05**	**0.82 ± 0.04**	**0.82 ± 0.03**
GameOfLife	0.14 ± 0.17	**0.81 ± 0.06**	0.64 ± 0.11	0.44 ± 0.21	**0.82 ± 0.04**
Sysadmin	0.02 ± 0.01	**0.66 ± 0.03**	0.54 ± 0.01	0.45 ± 0.05	0.57 ± 0.08

Table 8.2: Aggregate performance across different problems and planning times per domain normalized between 0 and 1. Pruned OGA-UCT is best or on par with the best on almost all domains, including those with high branching factors.

8.4.3 Observations

We compare OGA-UCT with ASAP-UCT and unabstracted UCT on these problems with different total planning times and draw cost vs. time curves. Representative runs on each domain are illustrated in Figure 8.2. Each curve is an average of 1,000 reruns and also draws 95% confidence interval bars.

In addition to the representative curves we also show aggregate performance of an algorithm on each domain across different planning times (Table 8.2). We choose six equi-spaced planning times for each problem and run the three algorithms along with all variants (pruned versions and different K values). Of all of these points, we give the least cost a score of 1, and the worst cost a score of zero. We normalize each cost value to a number between 0 and 1. This normalization is related to the metric used by IPPCs, with small differences. For example, IPPCs normalize only across algorithms and not across planning times.

We observe that OGA-UCT performs the best or on par with the best on four out of the six domains. In AcadAdvising and Navigation, OGA's performance is substantially better than both algorithms. It matches performance with UCT on RaceTrack and with ASAP-UCT on Sailing Wind. In Game of Life (GOL) and Sysadmin, OGA-UCT performs worse than ASAP-UCT. Both of these domains have exponential branching factors, which severely slow down abstraction computation routines. Since OGA recomputes abstractions more often than ASAP, it suffers significantly.

The pruned version of OGA-UCT helps with exactly this (we set probability threshold to 0.1). Pruned OGA-UCT makes abstraction computations approximate and only higher probability transitions are taken into account while computing abstractions. This improves performance on both domains, with pruned OGA-UCT becoming at par with ASAP-UCT on GOL. The performance in other domains remains mostly similar. For fair comparison, we also attempt pruning with ASAP-UCT and find that it hurts substantially in GOL and Sysadmin. Since ASAP-UCT computes abstractions only a handful of times per decision step, computing them accurately is likely more important for it than the computational savings due to approximation.

Overall, we find that performances of ASAP-UCT and UCT can depend heavily on the domain, but OGA-UCT admits least variance and is robustly good across several domains.

Sensitivity Analysis across different K-Values: We ran all our experiments for $K = 1, 3, 5, 7$. Table 8.3 shows the aggregate score for different Ks. While there is no single K, which is best across all domains, most of the performances are significantly close to each other. We choose $K = 3$, which gives an overall balanced performance across all domains.

Domains	OGA (K=1)	OGA (K=3)	OGA (K=5)	OGA (K=7)
AcadAdvising	0.86 ± 0.02	0.89 ± 0.03	0.91 ± 0.03	0.87 ± 0.08
Navigation	0.73 ± 0.07	0.57 ± 0.13	0.50 ± 0.11	0.45 ± 0.13
RaceTrack	0.79 ± 0.17	0.84 ± 0.13	0.85 ± 0.12	0.85 ± 0.13
SailingWind	0.76 ± 0.03	0.82 ± 0.05	0.84 ± 0.03	0.83 ± 0.03
GameOfLife	0.62 ± 0.09	0.64 ± 0.11	0.63 ± 0.11	0.61 ± 0.11
Sysadmin	0.42 ± 0.02	0.54 ± 0.01	0.59 ± 0.02	0.61 ± 0.02

Table 8.3: Comparison of OGA-UCT different K-values for all domains. Performances remain similar, and there is no clear winner.

8.5 Conclusions

We present OGA-UCT, an algorithm to compute domain abstractions on the go within the UCT framework for AI planning. It makes several desirable design choices such as it computes abstractions of state-action pairs, using an incremental and adaptive computation of abstractions, with a tight coupling between abstraction computation and tree construction. This allows OGA-UCT to efficiently recover from inaccurate abstractions as more information gets available. In the limit of infinite samples, OGA-UCT obtains a sound reduction of the original search tree and converges to the optimal solution.

Our experiments demonstrate that OGA-UCT is robust across domains. It compares favorably to the best of the algorithms in many domains. However, it can suffer when the branching factor is very high because that directly impacts the abstraction computation routine. An extension of OGA-UCT that prunes various low-probability transitions allows it to scale to such domains. Overall, Pruned OGA-UCT obtains best performance in almost all domains obtaining up to 28% quality gains.

We also contribute our analysis of algorithmic design choices applicable to MCTS with abstractions. We hope that this analysis will be useful in understanding existing algorithms and also for algorithm development in the future. This work comprehensively classifies the existing work of incorporating abstractions in MCTS algorithms. It makes some novel contributions in defining new abstractions as well as using these in modern AI planning algorithms.

In this work, ASAP abstractions have been applied in UCT which is a widely used MCTS algorithm. Apart from using abstractions, many works [Keller and Eyerich, 2012] have improved the performance of UCT using various other heuristics and optimizations. Prost [Keller and Eyerich, 2012] is one of these and have been a state-of-the-art solver in International Probabilistic Planning Competitions(IPPC)-2011 and IPPC-2014. Prost incorporates many more additional heuristics like bounding the search depth, identifying the reward locks and Q-Value initialization instead of roll outs to enhance the performance of vanilla UCT algorithm. There have been some recent attempts [Jain, 2017; Steindel, 2017] at incorporating ASAP abstrac-

tions in the Prost [Keller and Eyerich, 2012] but with limited success. Theoretically, ASAP framework must capture many more abstractions as compared to abstractions captured within Prost. The only symmetry based abstraction in Prost is that of removing equivalent actions within a state. On the other hand, OGA-UCT not only identifies state equivalences but also state-action pair equivalences among actions belonging to two different states.

However, ASAP abstractions have not shown that much benefit over Prost experimentally as demonstrated in [Jain, 2017]. In most cases, symmetries discovered by our algorithm are precisely the ones discovered by Prost using his optimizations/enhancements. There is not much additional benefit. (2) In some cases, symmetries lose out due to the time spent in trying to find them whereas in reality there are hardly any symmetries. Figuring out apriori which decision making problems may gain from exploiting symmetries (so that this overhead could be avoided in case of no or little symmetries) is a direction for future work.

We would also like to point out that some of the differences in performance may stem from different underlying implementations of Prost (which uses a Tree based data structure) vs MDP engine (which uses a DAG), and accounting for this differences may also result in discovery of additional symmetrical structures. Carrying out these experiments carefully is a direction for future work.

Part IV

Application of Symmetries to Computer Vision

Chapter 9

Symmetries for Structured Output Prediction in Computer Vision

This part of the thesis focuses on symmetry aware AI from the point of view of specific applications in computer vision. It differs from the previous parts in two major aspects. Firstly, both the previous parts applied symmetries in classic algorithms for those problems. This included MCMC algorithms for application of state symmetries in PGMs and MCTS algorithms in AI planning. This part investigates whether the proposed symmetry aware template is also useful in state-of-the-art algorithms which include many additional optimizations. It applies symmetry aware template to off the shelf near to state-of-the-art algorithms in stereovision and image segmentation. Secondly, both the previous parts used exact symmetries. This included multiple novel notions of state symmetries in PGMs and ASAP symmetries in planning. These exact symmetries may or may not be found in real world problems and domains. This part analyze the kind of symmetry which can be useful in real world problems specific to a particular domain. Then, it adapts the ideas of exact symmetries to propose novel notions of approximate symmetries which are really effective on real world datasets.

Specifically, we focus on the applications that use Probabilistic Graphical Models (PGMs) and apply the ideas proposed in lifted inference literature to speedup inference. Algorithms for NLP, computational biology, and computer vision (CV) problems make heavy use of PGM machinery (e.g., [Blei *et al.*, 2003; Friedman, 2004; Szeliski *et al.*, 2008a]). But, they also include significant problem-specific insights to get high performance. Barring a handful of exceptions as in [Jernite *et al.*, 2015; Nath and Domingos, 2016], lifted inference has not been applied directly to such algorithms.

Our work studies the potential value of lifting to CV problems such as image denoising, stereo vision, and image segmentation. Most CV problems can be modeled as structured output prediction tasks, typically assigning a label to each pixel. A large class of solutions are PGM-

based: they define a Markov Random Field (MRF) , where each pixel represents a node, with a unary potential that depends on the pixel value, and pairwise neighborhood potentials that favor similar labels to neighboring pixels. Inference on this MRF is computationally expensive since it is grid structured, and hence has high tree width [Koller and Friedman, 2009]. This work explores the use of symmetries to speedup the inference process.

We see three main challenges in applying existing lifted inference literature to these problems. First, most existing algorithms focus on computing marginals [Singla and Domingos, 2008; Kersting *et al.*, 2009; Gogate and Domingos, 2011; Niepert, 2012; Anand *et al.*, 2016b; Anand *et al.*, 2017] whereas most of CV problems are MAP inference problems. Second, among the algorithms performing lifted MAP [Noessner *et al.*, 2013; Mladenov *et al.*, 2014; Sarkhel *et al.*, 2014; Mittal *et al.*, 2014], most focus on exact lifting. Exact symmetries are not readily found in CV since most pixels may not have the *exact* same neighborhood. Third, the few algorithms that perform approximate lifting for MAP, e.g. [Sarkhel *et al.*, 2015], can not handle a distinct unary potential on every node. This is essential for our application since image pixels take ordinal values in three channels (RGB).

In response, we develop an approximate lifted MAP inference algorithm, which can effectively handle distinct unary potentials. We initialize our algorithm by merging together the pixels having the same order of top-k labels based on the unary potential values. It then adapts an existing symmetry finding algorithm [Kersting *et al.*, 2009] to discover groupings that also have similar neighborhoods. We refer to our groupings as *lifted pixels*. The algorithm, further, imposes the constraint that all pixels in a lifted pixel must be assigned the same label. Our approximate lifting reduces the model size drastically leading to significant time savings. Unfortunately, such approximate lifting could adversely impact the solution quality. However, we vary the degree of approximation in symmetry computation to output a *sequence* of coarse-to-fine models with varying quality-time trade-offs. By switching between such models, this work develops a coarse-to-fine (C2F) inference procedure applicable to many CV problems. Our C2F approach is an illustration of *joint symmetry aware inference* described in Part-I, where the symmetry computation module closely interacts with the inference procedure in an iterative fashion. Although the initial symmetries are precomputed but there is incremental symmetry improvement due to continuous refining of partitions. These ideas are formalized in a novel template for using lifted inference in CV.

This work tests C2F lifted inference on two problems: stereo matching and image segmentation. We start with one of the best MRF-based solvers each for both problems. Mozerov & Weijer [2015] use a *two-way* energy minimization to effectively handle occluded regions in stereo matching. Co-operative cuts [Kohli *et al.*, 2013] for image segmentation use concave functions over a predefined set of pixel pairs to correctly segment images with sharp edges. Our work implements C2F inference on top of both these algorithms and find that C2F versions

have a strong anytime behavior i.e., given any amount of inference time, they output a much higher quality (and are never worse) than their unlifted counterparts, and do not suffer any loss in the final quality.

Overall, this chapter makes the following three contributions. Firstly, we present an approximate lifted MAP algorithm that can efficiently handle a large number of distinct unary potentials. We develop a novel template for applying lifted inference in structured prediction tasks in CV. Secondly, we provide methods that output progressively finer approximate symmetries, leading to a C2F lifted inference procedure. Thirdly, we implement C2F inference over a near state-of-the-art stereo matching algorithm, TSGO and one of the popular MRF-based image segmentation algorithms, Cooperative Graph Cut. We release our implementation for wider use by the community.[1] We find that C2F has a much superior anytime behavior. In particular, for stereo matching it achieves 60% better quality on average in time-constrained settings while for image segmentation C2F reaches convergence in 33% less time.

This work was done jointly with Haroun Habeeb. While Haroun came up with the top-K heuristic, Ankit developed the framework for coarse-to-fine inference. Experiments were done jointly by Haroun and Ankit, with significant contributions from both of them. The part done by Haroun appeared in his bachelor's thesis.

9.1 Background

Most computer vision problems are structured output prediction problems and their PGM-based solutions often follow similar formulations. They cast the tasks into the problem of finding the lowest energy assignment over grid-structured MRFs (denoted by $G = (\mathcal{X}, \gamma)$). This representation of graphical model differs from our earlier representation of graphical models as a set of weighted features $\mathcal{G} = \{f_j, w_j\}_{j=1}^{m}$ in Part-II. Although, the two representations are mathematically equivalent, we introduce the new representation to align it with formulations popularly used in computer vision research.

9.1.1 Computer Vision Problems as MRFs

Given an MRF $G = (\mathcal{X}, \gamma)$ for an input image, the random variables in G are the set of pixels $\mathcal{X}$ in the image. Given a set of labels $L : \{1, 2, \ldots, |L|\}$, the task of structured output prediction is to label each pixel $X \in \mathcal{X}$ with a label from L. The MRFs have two kinds of potentials (γ) – unary and higher-order. Unary potentials are defined over each individual pixel, and usually incorporate pixel intensity, color, and other pixel features. Higher order potentials operate over cliques (pairs or more) of neighboring pixels and typically express some form of *spatial*

[1] Available at `https://github.com/dair-iitd/c2fi4cv/`

homophily – "neighboring pixels are more likely to have similar labels." While the general PGM structure of various tasks are similar, the specific potential tables and label spaces are task-dependent.

The goal is to find the MAP assignment over this MRF, which is equivalent to energy minimization (by defining energy as negative log of potentials) or probability maximization. We denote the negative log of unary potentials by ϕ, and that of higher-order potentials by ψ.[2] Thus, energy of a complete assignment $\mathbf{x} \in L^{|\mathcal{X}|}$ can be defined as:

$$E(\mathbf{x}) = \sum_{i \in 1..|\mathcal{X}|} \phi(x_i) + \sum_j \psi_j(\hat{x}_j) \tag{9.1}$$

Here $\hat{x}_j$ denotes the assignment $\mathbf{x}$ restricted to the set of variables in the potential ψ_j. And the output of the algorithm is the assignment $\mathbf{x}_{\text{MAP}}$:

$$\mathbf{x}_{\text{MAP}} = \arg\min_{\mathbf{x} \in L^{|\mathcal{X}|}} E(\mathbf{x}) \tag{9.2}$$

The problem is in general intractable. Efficient approximations exploit special characteristics of potentials like submodularity [Jegelka and Bilmes, 2011], or use variants of graph cut or loopy belief propagation [Boykov *et al.*, 2001; Freeman *et al.*, 2000].

Though there are various methods to solve the energy minimization, the methods based on graph cuts [Boykov and Jolly, 2001] have been fairly popular and effective. The main idea is to reduce each energy minimization problem to finding a min cut in the associated graph. For a binary pairwise labeling, the graph is constructed as follows: there is one vertex for each pixel. In addition, there are two additional vertices labeled as source and sink which are connected to each pixel vertex and the weight of that edge is the unary energy for that pixel. Also, there are edges between pixels whose weight is determined by pairwise energy on those pixels. Any labeling in this graph can be seen as defining a cut with the weight of that cut being the energy of the labeling. The problem of energy minimization, then, reduces to finding the minimum weight cut in the graph.

Many optimizations for this problem have been proposed for higher order potentials and multi-label case. Specifically, we use Alpha-Expansion fusion algorithm [Lempitsky *et al.*, 2010] which handles the problem of multiple discrete labels in these problem by combining two sub-optimal labelings using graph cut. The algorithms have been widely used and implemented in multiple MAP inference libraries.

[2] In the interest of readability, we say 'potential' to mean 'negative log of potential' in the rest of the paper.

9.1.2 Symmetries in Graphical Models

As described in Part-II, two popular methods for symmetry computation are color passing for computing symmetries of variables [Kersting *et al.*, 2009], and graph isomorphism for symmetries of states [Niepert, 2012; Bui *et al.*, 2013]. Part-II of thesis deals with graph isomorphism for computing state symmetries. In this chapter, we focus on color passing which is similar to message passing for bisimulations based symmetries used in MDPs as described in Part-III.

Color passing for an MRF operates over a colored bipartite graph containing nodes for all variables and potentials, and each node is assigned a color. The variable nodes form one side of the bipartite graph while potential nodes form the other side with the edges defined across these two sides. Each potential node has undirected edges to all the variables which participate in that potential. The colors in the graph are initialized as follows: all variables nodes get a common color; all potential nodes with exactly same potential tables are assigned a unique color. Now, in an iterative color passing scheme, in each iteration, each variable node sends a message containing its color to all neighboring potential nodes. The potential nodes store incoming color signatures (messages) in a vector, append their own color to it, and send the vector back to variable nodes. The variable nodes stack these incoming vectors. New colors are assigned to each node based on the set of incoming messages such that two nodes with same messages are assigned the same unique color. This process is repeated until convergence, i.e., no further change in colors.

A coloring of the bipartite graph defines a partition of variable nodes such that all nodes of the same color form a partition element. Each iteration of color passing creates successively finer partitions, since two variable nodes, once assigned different colors, can never get the same color.

9.2 Lifted Computer Vision Framework

In this section, we will describe our generic template which can be used to lift a large class of vision applications including those in stereo, and segmentation. Our template can be seen as transforming the original problem space to a reduced problem space over which the original inference algorithm can now be applied much more efficiently. Specifically, our description in this section is entirely *algorithm independent*.

We will focus on MAP inference which is the inference task of choice for most vision applications. Our formulation is based on the realization that pixels that are involved in the same (or similar) kinds of unary and higher order potentials, and have the same (or similar) neighborhoods, are likely to have the same MAP value. Therefore, if somehow we could discover such sets of pixels a priori, we could explicitly enforce these pixels to have the same value

while searching for the solution, substantially reducing the problem size and still preserving the optimal MAP assignment(s). Since in general doing this exactly may lead to a degenerate network, we do it approximately. Our approximations trade-off speed for marginal loss in solution quality. The loss in solution quality is offset by resorting to coarse-to-fine inference where we start with a crude approximation, and gradually make it finer, to guarantee optimality at the end while still obtaining significant gains. We next describe the details of our approach.

9.2.1 Obtaining a Reduced Problem

Consider an energy minimization problem over a PGM $G = (\mathcal{X}, \gamma)$. Let $L = \{1, 2, \cdots, |L|\}$ denote the set of labels over which variables in the set $\mathcal{X}$ can vary. We denote a partition of variables by $\mathcal{Y}^P$ in this chapter instead of Δ which is used for describing a partition in rest of the thesis. Specifically, let $\mathcal{Y}^P = \{Y_1^P, Y_2^P, \cdots, Y_r^P\}$ denote a partition of $\mathcal{X}$ into r disjoint subsets, i.e., $\forall k$, $Y_k^P \subseteq \mathcal{X}$, $Y_{k_1}^P \cap Y_{k_2}^P = \emptyset$ when $k_1 \neq k_2$, and $\bigcup_k Y_k^P = \mathcal{X}$. We refer to each Y_k^P as a partition element. Correspondingly, let us define $\mathcal{Y} = \{Y_1, Y_2, \cdots, Y_r\}$ as a set of partition variables, where there is a one to one correspondence between partition elements and the partition variables and each partition variable Y_k takes values in the set L. Let $part(X_i)$ denote the partition element to which X_i belongs. Let $\hat{X}_j \subseteq \mathcal{X}$ denote a subset of variables.

Definition 9.2.1. *We say that a partition element Y_k^P is* **represented** *in the set $\hat{X}_j$ if $\exists X_i \in \hat{X}_j$ s.t. $part(X_j) = Y_k^P$.*

Let $\gamma_j(\hat{X}_j)$ be a potential defined completely over the subset $\hat{X}_j$. Let $\hat{x}_j$ denote an assignment to variables in the set $\hat{X}_j$. Let $\hat{x}_j.elem(i)$ denote the value taken by a variable X_i in $\hat{X}_j$.

Definition 9.2.2. *We say that an assignment $\hat{X}_j = \hat{x}_j$* **respects** *a partition $\mathcal{Y}^P$ if the variables in $\hat{X}_j$ belonging to the same partition element have the same label in $\hat{x}_j$, i.e., $part(X_i) = part(X_{i'}) \Rightarrow \hat{x}_j.elem(i) = \hat{x}_j.elem(i'), \forall X_i, X_{i'} \in \hat{X}_j$.*

Next, we introduce the notion of a reduced potential.

Definition 9.2.3. *Let $\mathcal{X}$ be a set of variables and let $\mathcal{Y}^P$ denote its partition. Given the potential $\gamma_j(\hat{X}_j)$, the reduced potential Γ_j is defined to be the restriction of $\gamma_j(\hat{X}_j)$ to those labeling assignments of $\hat{X}_j$ which respect the partition $\mathcal{Y}^P$. Equivalently, we can define the reduced potential $\Gamma_j(\hat{Y}_j)$ over the set of partition variables $\hat{Y}_j$ which are represented in the set $\hat{X}_j$.*

For example, consider a potential $\gamma(X_1, X_2, X_3)$ defined over three Boolean variables. The table for γ would have 8 entries. Consider the partition $\mathcal{Y}^P = \{Y_1^P, Y_2^P\}$ where $Y_1^P = \{X_1, X_2\}$ and $Y_2^P = \{X_3\}$. Then, the reduced potential Γ is the restriction of γ to those rows in

the table where $X_1 = X_2$. Hence Γ has four rows in its table and equivalently can be thought of defining a potential over the 4 possible combinations of Y_1 and Y_2 variables. We are now ready to define a reduced graphical model.

Definition 9.2.4. *Let $G = (\mathcal{X}, \gamma)$ represent a PGM. Given a partition $\mathcal{Y}^P$ of $\mathcal{X}$, the reduced graphical model $\mathcal{G}(\mathcal{Y}, \Gamma)$ is the graphical model defined over the set of partition variables $\mathcal{Y}$ such that every potential $\gamma_j \in \gamma$ in G is replaced by the corresponding reduced potential $\Gamma_j \in \Gamma$ in $\mathcal{G}$.*

Let $E(\mathbf{x})$ and $\mathcal{E}(\mathbf{y})$ denote the energies of the states $\mathbf{x}$ and $\mathbf{y}$ in G and $\mathcal{G}$, respectively. The following theorem relates the energies of the states in the two graphical models.

Theorem 9.2.1. *For every assignment $\mathbf{y}$ of $\mathcal{Y}$ in $\mathcal{G}$, there is a corresponding assignment $\mathbf{x}$ of $\mathcal{X}$ such that $\mathcal{E}(\mathbf{y}) = E(\mathbf{x})$.*

Proof. The theorem can be proved by noting that each potential $\Gamma_j(\hat{Y}_j)$ in $\mathcal{G}$ was obtained by restricting the original potential $\gamma_j(\hat{X}_j)$ to those assignments where variables in X_j belonging to the same partition element took the same label. Since this correspondence is true for every potential in the reduced set, to obtain the desired state $\mathbf{x}$, for every variable $X_i \in \mathcal{X}$ we simply assign it the label of its partition element in $\mathbf{y}$. And since, the value of each potential is exactly same, the Energy value $E(\mathbf{x})$, which is sum of potentials, is exactly same as $\mathcal{E}(\mathbf{y})$. □

Let $\mathbf{x}_{\text{MAP}}$ and $\mathbf{y}_{\text{MAP}}$ be the MAP states (i.e. having the minimum energy) for G and $\mathcal{G}$, respectively. Then, $\mathcal{E}(\mathbf{y}_{\text{MAP}}) \geq E(\mathbf{x}_{\text{MAP}})$.

Proof. The process of reduction can be seen as curtailing the entire search space to those assignments where variables in the same partition take the same label. Let the set of states where the variables in the same partition element take the same label be denoted by $\mathcal{X}^{red}$. Let $\mathbf{y}_{\text{MAP}}$ be MAP-state in $\mathcal{G}$, then, as per Theorem 9.2.1, $\exists \mathbf{x}$ such that $E(\mathbf{x}) = \mathcal{E}(\mathbf{y}_{\text{MAP}})$. If MAP state exists in $\mathcal{X}^{red}$, then, this $\mathbf{x}$ corresponding to $\mathbf{y}_{\text{MAP}}$ will be MAP-state i.e $\mathbf{x}_{\text{MAP}} = \mathbf{x}$ and hence, $\mathcal{E}(\mathbf{y}_{\text{MAP}}) = E(\mathbf{x}_{\text{MAP}})$. Otherwise, let MAP state $\mathbf{x}_{\text{MAP}} \in \mathcal{X} - \mathcal{X}^{red}$, then, $E(\mathbf{x}_{\text{MAP}}) \leq E(\mathbf{x})$ where $\mathbf{x}$ is state corresponding to $\mathbf{y}_{\text{MAP}}$ as per theorem 9.2.1. Hence, $\mathcal{E}(\mathbf{y}_{\text{MAP}}) = E(\mathbf{x}) \geq E(\mathbf{x}_{\text{MAP}})$. □

A reduction in the problem space will lead to computational gains, but might result in loss of solution quality, where the solution quality can be captured by the difference between $\mathcal{E}(\mathbf{y}_{\text{MAP}})$ and $E(\mathbf{x}_{\text{MAP}})$. Therefore, we need to trade-off the balance between the two.

Intuitively, a good problem reduction will keep those variables in the same partition that are likely to have the same value in the optimal assignment for the original problem. How do we find such variables approximately without actually solving the inference task? We will describe one such technique in Section 9.2.3.

There is another perspective. Instead of solving one reduced problem, we can instead work with a series of reduced problems, which successively get closer and closer to the optimal solution. The initial reductions are coarser and far from optimal, but can be solved efficiently to quickly reach near the region where the solution lies. The successive iterations can then refine the solution iteratively getting closer to the optimal. This leads us to the coarse-to-fine inference described next.

9.2.2 Coarse-to-Fine Inference

We will define a framework for C2F (coarse-to-fine) inference so that we maintain the computational advantage while still preserving optimality. In the following, for ease of notation, we will drop the superscript P in $\mathcal{Y}^p$ to denote the partition of $\mathcal{X}$. Therefore, $\mathcal{Y}$ will refer to both the partition as well as the set of partition variables. Before we describe our algorithm, let us start with some definitions.

Definition 9.2.5. *Let $\mathcal{Y}$ and $\mathcal{Y}'$ be two partitions of $\mathcal{X}$. We say that $\mathcal{Y}$ is coarser than $\mathcal{Y}'$, denoted as $\mathcal{Y} \preceq \mathcal{Y}'$, if $\forall Y'_i \in \mathcal{Y}'$, $\exists Y_j \in \mathcal{Y}$ such that $Y'_i \subseteq Y_j$ where Y'_i and Y_j denote the partition elements of respective partition. We equivalently say that $\mathcal{Y}'$ is finer than $\mathcal{Y}$.*

It is easy to see that $\mathcal{X}$ defines a partition $\mathcal{Y}^*$ where we have a separate partition element corresponding to each variable. This partition is the finest among all partitions, i.e., $\forall \mathcal{Y}$ such that $\mathcal{Y}$ is a partition of $\mathcal{X}$, $\mathcal{Y} \preceq \mathcal{Y}^*$. We also refer it to as the degenerate partition. We will refer to the corresponding PGM as $\mathcal{G}^*$ (same as G). Next, we state a theorem which relates two partitions with each other.

Lemma 9.2.1. *Let $\mathcal{Y}$ and $\mathcal{Y}'$ be two partitions of $\mathcal{X}$ such that $\mathcal{Y} \preceq \mathcal{Y}'$. Then for every state $\mathbf{y}$ corresponding to reduced graphical model $\mathcal{G}$ associated with $\mathcal{Y}$, there is a state $\mathbf{y}'$ in the reduced graphical model $\mathcal{G}'$ associated with $\mathcal{Y}'$ such that $\mathcal{E}(\mathbf{y}) = \mathcal{E}(\mathbf{y}')$. Further, if $\mathbf{y}_{\text{MAP}}$ and $\mathbf{y'}_{\text{MAP}}$ are MAP states corresponding to $\mathcal{G}$ and $\mathcal{G}'$ respectively, then, $\mathcal{E}(\mathbf{y}_{\text{MAP}}) \geq \mathcal{E}(\mathbf{y'}_{\text{MAP}})$*

Proof. The proof of this lemma is similar to theorem 9.2.1. Given the state $\mathbf{y}$, we can construct the state $\mathbf{y}'$ in the following way. Since $\mathcal{Y}'$ is finer than $\mathcal{Y}$, $\forall Y'_i \in \mathcal{Y}'$, $\exists Y_j \in \mathcal{Y}$ s.t $Y'_i \subseteq Y_j$, we assign the same label to each partition element Y'_i in $\mathbf{y}'$ exactly the same label as taken by its counterpart Y_i in state $\mathbf{y}$. Then, we need to prove that the energy obtained for $\mathbf{y}'$ constructed in this way is same as energy of state $\mathbf{y}$. It should be noted that both $\mathbf{y}$ and $\mathbf{y}'$ will have the same state $\mathbf{x}$ in the ground model given by theorem 9.2.1. This is because, multiple partition elements of $\mathcal{Y}'$ each of which is a subset of a particular partition element $Y_j \in \mathcal{Y}$ will get the same label. Let X_k be a ground pixel which belongs to partition element Y'_i in $\mathcal{Y}'$. If $Y'_i \subseteq Y_j$ then, X_k, must belong to the Y_j in $\mathcal{Y}$. Hence, X_k will get the same label whether $\mathbf{x}$ is constructed through

$\mathcal{Y}$ or $\mathcal{Y}'$ (as both partition-elements have same label). Hence, $\mathcal{E}(\mathbf{y}) = E(\mathbf{x})$ and $\mathcal{E}(\mathbf{y}') = E(\mathbf{x})$. Therefore, $\mathcal{E}(\mathbf{y}) = \mathcal{E}(\mathbf{y}')$.

For the second part, it should be noted that there are more states in graphical model corresponding to $\mathcal{Y}'$ than in the graphical model associated with $\mathcal{Y}$. Also, from the first part, for every state in graphical model for $\mathcal{Y}$, there is a state in graphical model associated with $\mathcal{Y}'$ which has same energy. Hence, $\mathcal{E}(\mathbf{y}_{\text{MAP}}) \geq \mathcal{E}(\mathbf{y'}_{\text{MAP}})$.

□

Consider a set $\mathbf{Y}$ of coarse to fine partitions given as $\mathcal{Y}^0 \preceq \mathcal{Y}^1, \cdots, \preceq, \mathcal{Y}^t, \preceq, \cdots, \mathcal{Y}^*$. Let $\mathcal{G}^t, \mathcal{E}^t, \mathbf{y}^t_{MAP}$ respectively denote the reduced problem, energy function and MAP assignment for the partition $\mathcal{Y}^t$. Using Lemma 9.2.1, we have $\mathcal{E}^t(\mathbf{y^t}) = \mathcal{E}^{t+1}(\mathbf{y^{t+1}})$ for every state y^t. Also, we have we have $\forall t\ \mathcal{E}^t(\mathbf{y}^{\mathbf{t}}_{\text{MAP}}) \geq \mathcal{E}^{t+1}(\mathbf{y}^{\mathbf{t+1}}_{\text{MAP}})$. Together, these two statements imply that starting from the coarsest partition, we can gradually keep on improving the solution as we move to finer partitions.

Our C2F set-up assumes an iterative MAP inference algorithm A, which has the anytime property i.e., can produce solutions of increasing quality with time. C2F Function (see Algorithm 10) takes 3 inputs: a set of C2F partitions $\mathbf{Y}$, inference algorithm A, and a stopping criteria $\mathcal{C}$. The algorithm A in turn takes three inputs: PGM $\mathcal{G}^t$, starting_assignment $\mathbf{y}^t$, stopping criteria $\mathcal{C}$. A outputs an approximation to the MAP solution once the stopping criteria $\mathcal{C}$ is met. Starting with the coarsest partition ($t = 0$ in line 2), a start state is picked for the coarsest problem to be solved (line 3). In each iteration (line 4), C2F finds the MAP estimate for the current problem ($\mathcal{G}^t$) using algorithm A (line 5). This solution is then mapped to a same energy solution of the next finer partition (line 6) which becomes the starting state for the next run of A. The solution is thus successively refined in each iteration. The process is repeated until we reach the finest level of partition. In the end, A is run on the finest partition and the resultant solution is output (lines 8,9). Since the last partition in the set is the original problem $\mathcal{G}^*$, optimality with respect to A is guaranteed.

Next, we describe how to use the color passing algorithm (Section 9.1) to get a series of partitions which get successively finer. Our C2F algorithm can then be applied on this set of partitions to get anytime solutions of high quality while being computationally efficient.

Algorithm 10 Coarse-to-Fine Lifted MAP Algorithm

procedure C2F_Lifted_MAP(C2F Partitions $\mathbf{Y}$, Algo A, Criteria $\mathcal{C}$)
  $t = 0, T = |\mathbf{Y}|$
  $\mathbf{y}^t$ = getInitState($\mathcal{G}^t$)
  while $t < T$ **do**
    $\mathbf{y}^t_{\text{MAP}} = A(\mathcal{G}^t, y^t, \mathcal{C})$
    $\mathbf{y}^{t+1} = getEquivAssignment(\mathcal{Y}^t, \mathcal{Y}^{t+1}, \mathbf{y}^t_{\text{MAP}})$
    $t = t + 1$
  $\mathbf{y}^T_{\text{MAP}} = A(\mathcal{G}^T, y^T, \mathcal{C})$
  return y^T_{MAP}

9.2.3 C2F Partitioning for Computer Vision

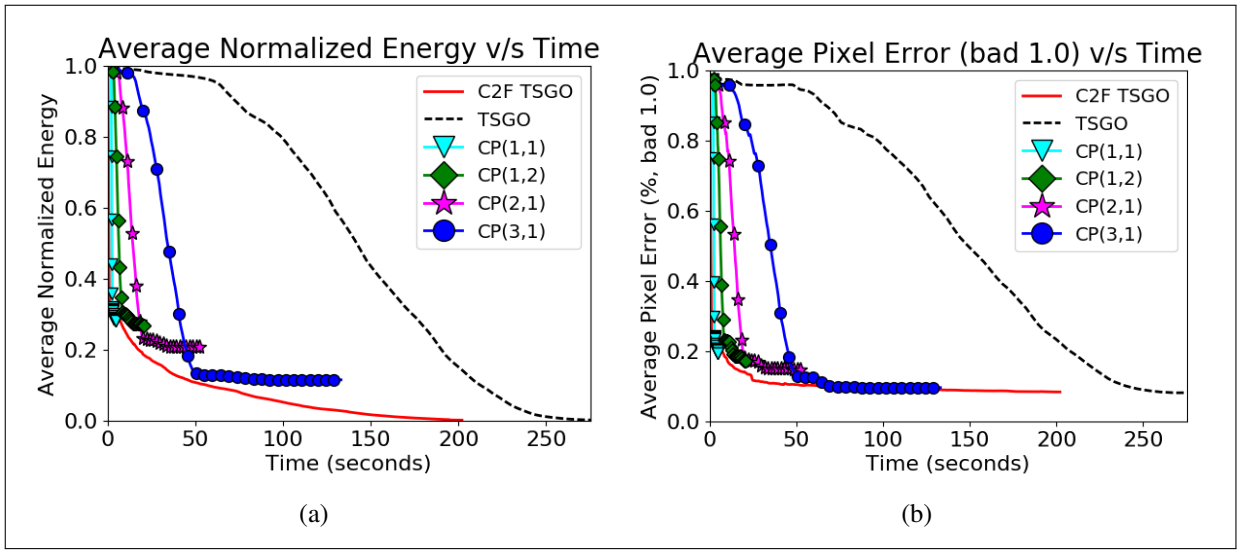
We now adapt the general color passing algorithm to MRFs for CV problems. Unfortunately, unary potentials make color passing highly ineffective. Different pixels have different RGB values and intensities, leading to almost every pixel getting a different unary potential. Naive application of color passing splits almost all variables into their own partitions, and lifting offers little value.

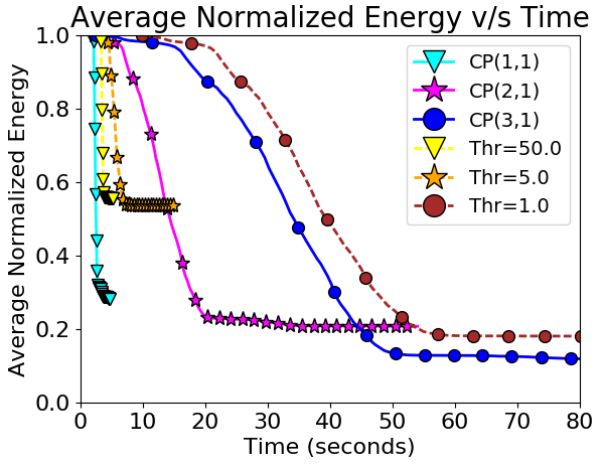
A natural approximation is to define a threshold, such that two unary potentials within that threshold be initialized with the same color. Our experiments show limited success with this scheme because two pixels may have the same label even when their actual unary potentials are very different. What is more important is relative importance given to each label than the actual potential value.

In response, we adapt color passing for CV by initializing it as before, but with one key change: we initialize two unary potential nodes with the same color if their lowest energy labels have the same order for the top N_L labels (we call this unary split threshold). Experiments reveal that this approximation leads to effective partitions for lifted inference.

Finally, we can easily construct a sequence of coarse-to-fine partitions in the natural course of color passing's execution – every iteration of color passing creates a finer partition. Moreover, as an alternative approach, we may also increase N_L. In our implementations, we intersperse the two, i.e., before every next step we pick one of two choices: either, we run another iteration of color passing; or, we increase N_L by one, and split each variable partition based on the N_L^{th} lowest energy labels of its constituent variables.

We parameterize $CP(N_L, N_{iter})$ to denote the partition from the current state of color passing, which has been run till N_{iter} iterations and unary split threshold is N_L. It is easy to prove that another iteration of color passing or splitting by increasing N_L as above leads to a finer partition. I.e., $CP(N_L, N_{iter}) \preceq CP(N_L + 1, N_{iter})$ and $CP(N_L, N_{iter}) \preceq CP(N_L, N_{iter} + 1)$. We refer to each element of a partition of variables as a *lifted pixel*, since it is a subset of pixels.

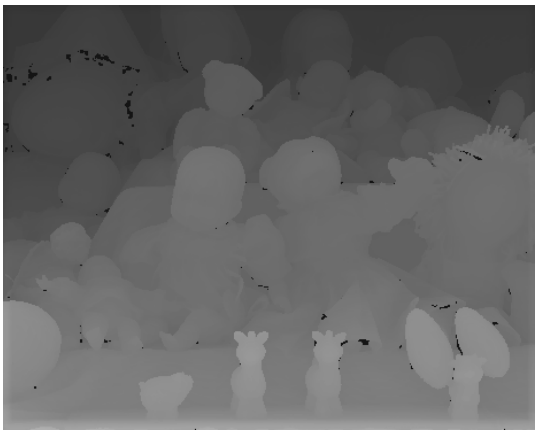




(c)

Figure 9.1: **(a)** Average (normalized) energy vs. inference time **(b)** Average pixel error vs. time. C2F TSGO achieves roughly 60% reduction in time for reaching the optima. It has best anytime performance compared to vanilla TSGO and static lifted versions. **(c)** Average (normalized) energy vs. time for different thresholding values and CP partitions. Plots with the same marker have MRFs of similar sizes.

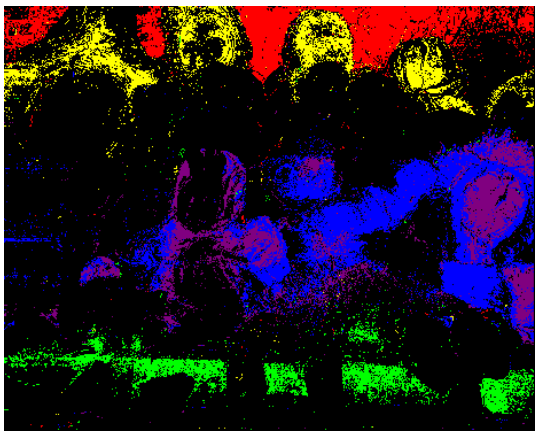
(a)



(b)

(c)

(d)



(e)

Figure 9.2: Qualitative results for Doll image at convergence. C2F-TSGO is similar to base TSGO.**(a)** Left and Right Images **(b)** Ground Truth **(c)** Disparity Map by TSGO **(d)** Disparity Map by C2F TSGO **(e)** Each colored region (other than black) is one among the 10 largest partition elements from CP(1,1). Each color represents one partition element. Partition elements form non-contiguous regions

9.3 Lifted Inference for Stereo Matching

We first demonstrate the value of C2F lifted inference in the context of stereo matching [Scharstein and Szeliski, 2002]. It aims to find pixel correspondences in a set of images of the same scene, which can be used to further estimate the 3D scene. Formally, two images I^l and I^r corresponding to images of the scene from a left camera and a right camera are taken such that both cameras are at same horizontal level. The goal is to compute a disparity labeling D^l for every pixel $X = (a, b)$ such that $I^l[a][b]$ corresponds to $I^r[a - D^l[a][b]][b]$. We build a lifted version of a popular MRF based stereomatching algorithm, Two Step Global Optimization (TSGO) [Mozerov and van de Weijer, 2015]. TSGO ranks 2^{nd} on the Middlebury Stereo Evaluation Version 2 leaderboard at the time of publication of this work.[3]

9.3.1 Background on TSGO Algorithm

Traditionally, stereomatching methods are classified in two categories: cost filtering methods and energy minimization methods. A neighborhood based filter is applied in cost filtering methods based on the fact that neighboring pixels have similar disparity while the energy minimization methods perform a global optimization over the complete image space. Though energy minimization methods have shown good results but cost filtering methods perform better in occluded regions. Our base algorithm, TSGO combines the virtue of both the methods to propose a novel two step global optimization algorithm.

The first step of the algorithm is a cost filtering method on a fully connected MRF with pairwise potentials while the second is a global energy minimization on a conventional locally connected pairwise MRF (4-connected model). At the high level, TSGO runs message passing on a fully connected MRF, computes marginals of each pixel X_i. The marginals computed by the first step act as unary potentials $\phi_i^2(l)$ for the MRF of second step. We use the superscript to denote the step (out of 2 steps) of energy minimization in TSGO.

The fully connected MRF with pairwise potentials has the energy given by

$$E^1(\mathbf{x}) = \sum_{i=1\rightarrow|\mathcal{X}|} \phi_{\mathbf{i}}^{\mathbf{1}}(x_i) + \sum_{\{i,j\}=1\rightarrow|\mathcal{X}|, i\neq j} w(i,j)(1-\delta(x_i,x_j)) \tag{9.3}$$

where the unary potential for the first stage ϕ_i^1 is computed as a linear combination of two terms: 1) Dissimilarity between the right and left images (u_i^I) 2) Dissimilarity between the gradient of two images (u_i^G).

The dissimilarity between left and right images (u^I) is given by popular Birchfield-Tomasi measure [Birchfield and Tomasi, 1999] given by

[3] Available at `http://vision.middlebury.edu/stereo/eval/`

$$u_i^I = \min(\min_{l-0.5 \leq d \leq l+0.5} (\sum_{c \in C} |I^l(x_i) - I^r(x_i - d)|), \tau) \tag{9.4}$$

where C is the set of channels and τ is a parameter. The gradient image is a 6-D vector function in the same space as image and computed as given in [Mozerov and van de Weijer, 2015].

The pairwise terms is a product of two terms where $w(i,j)$ is a weight of pairwise potential function which captures the influence between two pixels i and j in the position and color space of pixels i, j and $\delta(X_i, X_j)$ is the Kronecker delta function. The marginals of each pixel for the first stage can be computed by belief propagation or message passing method. The aim of the message passing algorithm is to compute the marginals of each pixel $\bar{\phi}_i^1$. This is accomplished by iterative message passing between connected nodes where a message ($m_{i \to j}^t$) is passed at iteration t between pixels i and j if an edge exists between them. The marginals at iteration t is given as $\bar{\phi}^{1,t}$

$$\bar{\phi}_i^{1,t}(l) = \phi_i^1(l) + \sum_{j=1 \to |\mathcal{X}|} m_{j \to i}^t(l) \tag{9.5}$$

where the message is given by

$$m_{i \to j}^t(l) = \min_{l' \in L} (\bar{\phi}_i^{1,t-1}(l) - m_{j \to i}^{t-1}(l) + \sum_{\{i,j\}=1 \to |\mathcal{X}|,\, i \neq j} w(i,j)(1 - \delta(l, l')) \tag{9.6}$$

Since increasing the number of iterations of message passing at this stage does not improve the final result, only one iteration of message passing is performed at this stage. Further, under appropriate constraints on unary potentials (as described in [Mozerov and van de Weijer, 2015]), this iteration of message passing is shown to be equivalent to a bilateral filter and can be performed efficiently. The computed marginals are used as unary potentials for the second layer.

The MRF used in second stage is a standard grid MRF which have pairwise potentials over neighbors in space. The pairwise potential ψ used in step two is $\psi(X_i, X_j) = w(X_i, X_j)\varphi(X_i, X_j)$, where $\varphi(X_i, X_j)$ is a truncated linear function of $\|X_i - X_j\|$, and $w(X_i, X_j)$ takes one of three distinct constant values ($\lambda_{1,2}$, λ_3) depending on color difference between pixels. The color difference captures the gradient and hence, edges are divided in three categories depending on low or high image gradient. The unary potentials are marginals computed by first stage. The MAP assignment $\mathbf{x}_{\mathrm{MAP}}$ computes the lowest energy assignment of disparities D^l for every pixel for this MRF. The energy minimization in this second space is Np-hard and hence, approximate inference algorithms have to be used to compute the MAP state. Specifically, TRW-S algorithm [Kolmogorov, 2006] based on belief propagation framework was used for computing the MAP state in TSGO. This step is computationally expensive in the whole pipeline.

The last stage in TSGO algorithm is post-processing of disparity map obtained in Step-2

of energy minimization. TSGO involves a complex multi-stage post processing which consists of a left-to-right disparity map cross checking, weighted median filtering the obtained disparity map and outlier suppression techniques (details in [Mozerov and van de Weijer, 2015]. Since these post processing techniques are orthogonal to applying symmetry in MAP inference, we use these techniques as used in base algorithms directly in our work.

9.3.2 Lifted TSGO

Since step two is costlier (as first step runs only single iteration), we build its lifted version as discussed in previous section. For color passing, two unary potential nodes are initialized with the same color if their lowest energy labels exactly match ($N_L = 1$). Other initializations are consistent with original color passing for general MRFs. A sequence of coarse-to-fine models is outputted as per Section 9.2.3. C2F TSGO uses outputs from the sequence $CP(1,1)$, $CP(2,1)$, $CP(3,1)$ and then refines to the original MRF. Model refinement is triggered whenever energy has not decreased in the last four iterations of alpha expansion (this becomes the stopping criteria $\mathcal{C}$ in Algorithm 1).

Experiments: Our experiments build on top of the existing TSGO implementation,[4] but we change the minimization algorithm in step two to alpha expansion fusion [Lempitsky *et al.*, 2010] from OpenGM2 library [Andres *et al.*, 2010; Kappes *et al.*, 2015], as it improves the speed of the base implementation. We use the benchmark Middlebury Stereo datasets of 2003, 2005 and 2006 [Scharstein and Szeliski, 2003; Hirschmuller and Scharstein, 2007]. For the 2003 dataset, quarter-size images are used and for others, third-size images are used. The label space is of size 85 (85 distinct disparity labels).

We compare our coarse-to-fine TSGO (using $CP(N_L, N_{iter})$ partitions) against vanilla TSGO. Figures 9.1(a,b) show the aggregate plots of energy (and error) vs. time. We observe that C2F TSGO reaches the same optima as TSGO, but in less than half the time. It has a much superior anytime performance – if inference time is given as a deadline, C2F TSGO obtains 59.59% less error on average over randomly sampled deadlines. We also eyeball the outputs of C2F TSGO and TSGO and find them to be visually similar. Figure 9.2 shows a sample qualitative comparison. Figure 9.2(e) shows five of the ten largest partition elements in the partition from $CP(1,1)$. Clearly, the partition elements formed are not contiguous, and seem to capture variables that are likely to get the same assignment. This underscores the value of our lifting framework for CV problems.

We also compare our $CP(N_L, N_{iter})$ partitioning strategy with threshold partitioning discussed in Section 9.2.3. We merge two pixels in thresholding scheme if the L1-norm distance of their unary potentials is less than a threshold. For each partition induced by our approach, we

[4] Available at `http://www.cvc.uab.es/$\sim$mozerov/Stereo/`

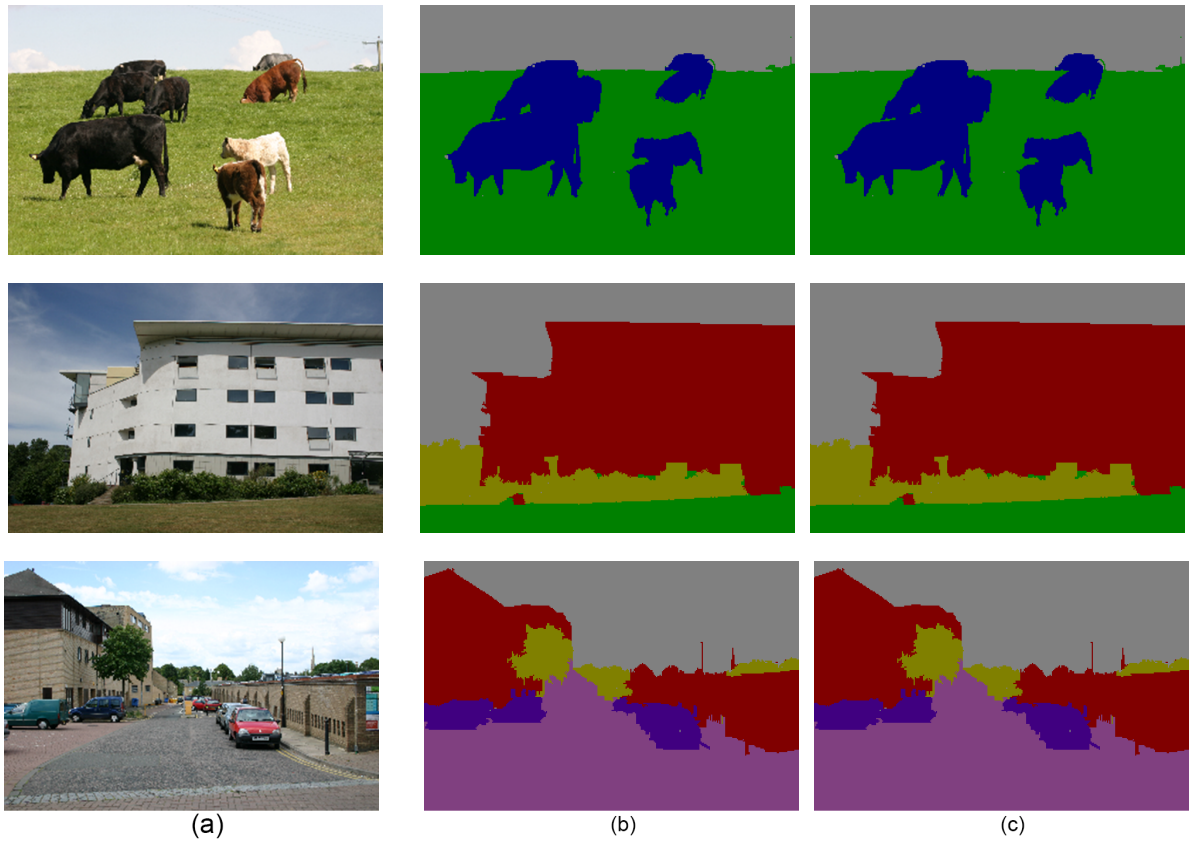


Figure 9.3: **(a-c)** Qualitative Results for Segmentation. C2F has quality similar to CoGC algorithm **(a)** Original Image **(b)** Segmentation by CoGC **(c)** Segmentation by C2F CoGC

find a value of threshold that has roughly the same number of lifted pixels. Figure 9.1(c) shows that partitions based on $CP(1,1)$ and $CP(3,1)$ converges to a much lower energy quickly compared to the corresponding threshold values ($Thr = 50$ and $Thr = 1$ respectively). For $CP(2,1)$, convergence is slower compared to corresponding threshold ($Thr = 5$) but eventually $CP(2,1)$ has significantly better quality.

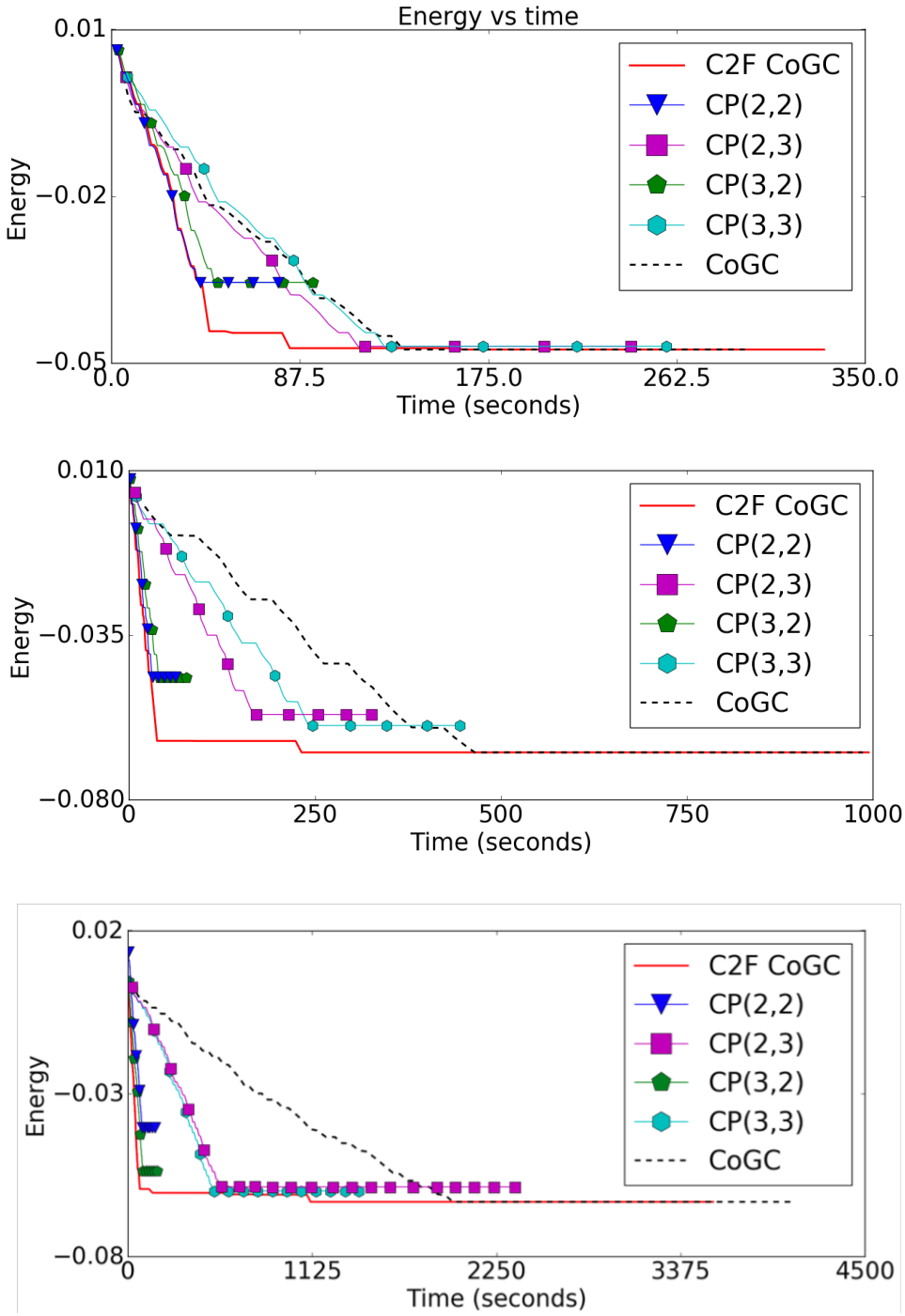
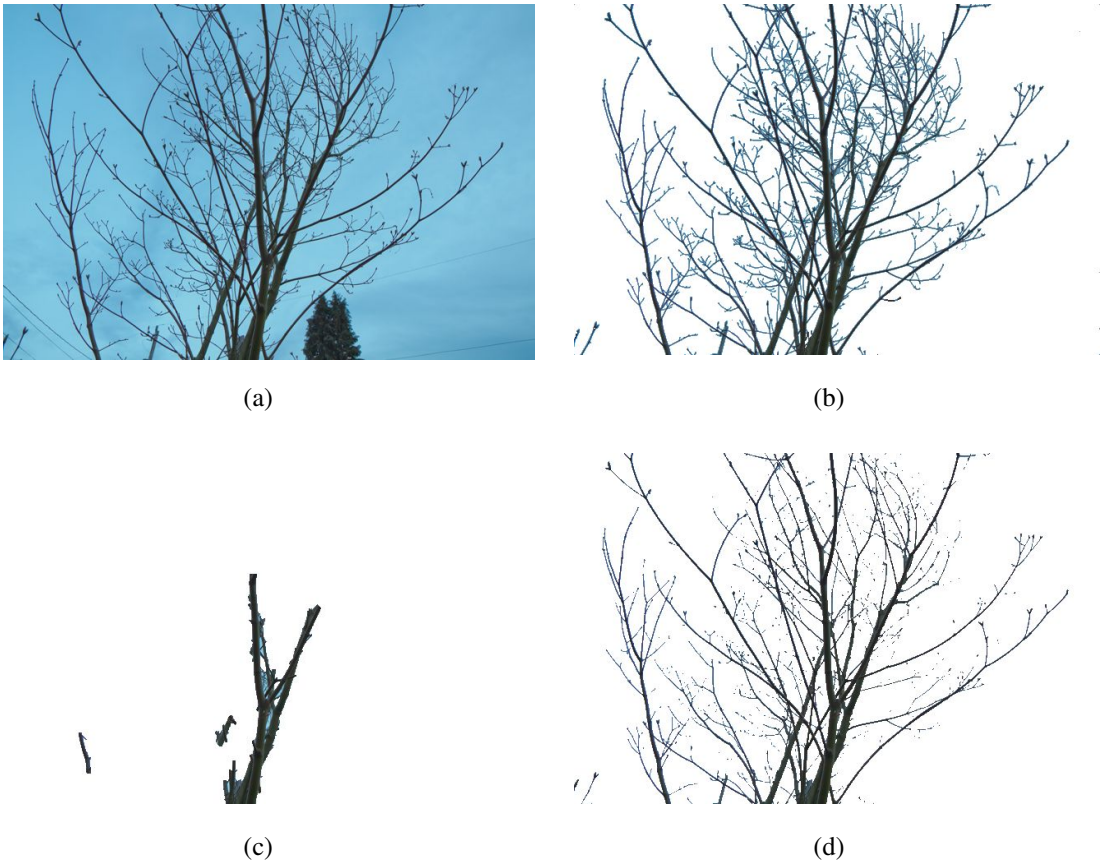


Figure 9.4: C2F CoGC has lower energy compared to CoGC and other lifted variant at all times on three random samples

9.4 Lifted Inference for Image Segmentation

We now demonstrate the general nature of our lifted CV framework by applying it to a second task. We choose multi-label interactive image segmentation, where the goal is to segment an image I based on a seed labeling (true labels for a few pixels) provided as input. Like many other CV problems, this also has an MRF-based solution, with the best label-assignment generally obtained by MAP inference using graph cuts or loopy belief propagation [Boykov *et al.*, 2001; Szeliski *et al.*, 2008a].

However, MRFs with only pairwise potentials are known to suffer from *short-boundary bias* – they prefer segmentations with shorter boundaries, because pairwise potentials penalize every pair of boundary pixels. This leads to incorrect labeling for sharp edge objects. For example, Figure 9.5 (b), shows the original image and ground truth for a tree while Figure 9.5(c) shows the segmentation obtained by a traditional pairwise model. Kohli *et al.* [2013] use CoGC, cooperative graph cuts [Jegelka and Bilmes, 2011], to develop one of the best MRF-based solvers that overcome this bias. Figure 9.5(d) shows the segmentation obtained by cooperative graph cut which clearly captures the sharp edges better.



(a) (b)

(c) (d)

Figure 9.5: Example of Short Boundary Bias on a sample tree image **(a)** Original Image **(b)** Ground Truth **(c)** Segmentation by Classical Pairwise MRF **(d)** Segmentation by Cooperative Cut

9.4.1 Background on Cooperative Graph Cut (CoGC)

Traditional MRFs define the energy function over a grid structured model where each state $\mathbf{x}$ has the following energy

$$E(\mathbf{x}) = \sum_{i=1 \rightarrow |\mathcal{X}|} \phi_i(x_i) + \sum_{\{i,j\} \in edges} \psi_{ij}(x_i, x_j) \tag{9.7}$$

Here, the function ϕ encode the unary potential of a particular pixel intensity taking a particular label while the ψ encode the contrast sensitive pairwise potential of two neighboring pixels taking the same label. Depending on contrast difference between two neighboring pixels, it penalizes each pair of neighboring pixels differently if they do not take the same label.

In the graphical structure, each pixel corresponds to a vertex and there is an edge corresponding to each pairwise potential. A particular labelling, then, can be thought of as a partition of vertices and as a cut in this grid graph. Label discontinuity on neighboring pixels can be seen as edges across the cut. The energy minimization, therefore, corresponds to finding the minimum cut in this graph. The problem with this model is that it linearly penalize the number of label discontinuities at edges (boundary pixel pairs). This helps in ensuring smoothness on neighboring pixels but at the same time restricts the model to always prefer a short boundary or perimeter of an object. This problem is particularly critical when segmenting objects with sharp boundaries like tree leaves, insect wings etc.

CoGC, on the other hand, overcomes this problem by not penalizing the number of label discontinuities but the diversity of different types of label discontinuities. This is helpful for sharp objects where the same label discontinuity is repeated again and again. The cooperative cut achieves this by dividing the set of edges in groups. A set of edge groups, $\mathbb{G}$, are created such that each element, $g, \in \mathbb{G}$ represents one type of label discontinuity. Associated with each element of edge group g is a set, ε_g, of edges or pixel pairs. The model obtains the partition of all edges into ε_g by clustering edges based on the color difference. The cooperative cut model has the following energy function:

$$E(\mathbf{x}) = \sum_{i=1}^{|\mathcal{X}|} \phi_i(x_i) + \sum_{g \in \mathbb{G}} \sum_{l \in L} F\left(\sum_{(x,x') \in \varepsilon_g} w(x, x').\mathbb{I}(x = l, x' \neq l) \right) \tag{9.8}$$

The key difference in this energy function is F which is a concave function. $\mathbb{I}$ is the indicator function, and $w(x, x')$ depends on the color difference between x, x'. Intuitively, F collects all edges with similar discontinuities and penalizes them sub-linearly, thus reducing the short-boundary bias in the model. The model is a strict generalization of classical energy function. If the function F is identity, the model reduced to classical pairwise energy function.

The usage of a concave function over the edge groups makes the MRF higher order with

complexity exponential to the number of vertices in a group. Though MAP inference over higher order potentials is expensive in general, it can be shown that the higher order potential over edge groups can be reduced to pairwise potentials through the addition of auxiliary variables [Jegelka and Bilmes, 2011]. However, the number of auxiliary variables required are exponential in the number of pixels taking part in higher order potentials. [Kohli *et al.*, 2013] showed that in the presence of structure as available for these cooperative cut potentials, the number of auxiliary variables required are linear in number of edges. They further optimize their algorithm using dynamic max-flow algorithms which compute a series of related max-flow problems efficiently.

9.4.2 Lifted CoGC

CoGC is lifted using the framework of Section 9.2, with some additional changes. The unary potentials are grouped using K-Label heuristic and specific number of iterations of color passing as earlier. It should be noted that the each edge group, which signifies a different higher order potential, is colored differently during initialization of color passing. This avoids merging two pixels together which have edges in different edge groups. This also results in having the resulting lifted model's energy function exactly same as the original CoGC. Hence, the lifted model can also use the same set of optimizations as proposed for CoGC.

Another change in Lifted CoGC from proposed CoGC is how we obtain edge groups. We cluster edge groups using color difference *and* the position of the edge. This is performed by doing k-means clustering over edges. Edge groups that are formed only on the basis of color difference make the error of grouping different segment's boundaries into a single group. For e.g., it erroneously cluster boundaries between white cow and grass, and sky and grass together in the top image in Figure 9.3.

Coarse-to-fine partitions are obtained by the method described in Section 9.2.3. C2F CoGC uses outputs from the sequence $CP(\lceil \frac{L}{2} \rceil, 2), CP(\lceil \frac{L}{2} \rceil, 3)$ before refining to the original MRF. Model refinement is triggered if energy has not reduced over the last $|L|$ iterations.

Experiments: Our experiments use the implementation of Cooperative Graph Cuts as provided by [Kohli *et al.*, 2013].[5] Energy minimization is performed using alpha expansion fusion [Boykov *et al.*, 2001]. The implementation of CoGC performs a greedy descent on auxiliary variables while performing alpha expansion fusion on the remaining variables, as described in Kohli et. al. [2013]. The dataset used is provided with the implementation. It is a part of the MSRC V2 dataset.[6]

[5] Available at `https://github.com/aosokin/coopCuts_CVPR2013`

[6] Available at `https://www.microsoft.com/en-us/research/project/image-understanding/?from=http%3A%2F%2Fresearch.microsoft.com%2Fvision%2Fcambridge%2Frecognition%2F`

Figure 9.3 shows the qualitative performance of C2F CoGC with respect to CoGC on three different images. C2F CoGC obtains the same quality as CoGC. Figure 9.4 shows three individual energy vs. time plots. Results on other images are similar. We find that C2F CoGC algorithm converges to the same energy as CoGC in about two-thirds the time on average. Overall, C2F CoGC achieves a much better anytime performance than other lifted and unlifted CoGC.

Similar to Section 9.3, refined partitions attain better quality than coarser ones at the expense of time. Since the implementation performs a greedy descent over auxiliary variables, refinement of current partition also resets the auxiliary variables to the last value that produced a change. Notice that energy minimization on output of $CP(2,3)$ attains a lower energy than on $CP(3,2)$. This observation drives our decision to refine by increasing N_{iter}. Qualitatively, C2F CoGC produces the same labeling as CoGC. Finally, similar to stereo matching, partitions based on thresholding scheme perform significantly worse compared to $CP(N_L, N_{iter})$ for image segmentation as well.

9.5 Related Work

There is a large body of work on exact lifting, both marginal [Kersting *et al.*, 2009; Gogate and Domingos, 2011; Niepert, 2012; Mittal *et al.*, 2015] and MAP [Kersting *et al.*, 2009; Gogate and Domingos, 2011; Niepert, 2012; Sarkhel *et al.*, 2014; Mittal *et al.*, 2014], which is not directly applicable to our setting. There is some recent work on approximate lifting [Van den Broeck and Darwiche, 2013; Venugopal and Gogate, 2014a; Singla *et al.*, 2014; Sarkhel *et al.*, 2015; Van den Broeck and Niepert, 2015] but its focus is on marginal inference whereas we are interested in lifted MAP. Further, this work cannot handle a distinct unary potential on every node. An exception is work by Bui et al. [2012] which explicitly deals with lifting in presence of distinct unary potentials. Unfortunately, they make a very strong assumption of exchangeability in the absence of unaries which does not hold true in our setting since each pixel has its own unique neighborhood.

Work by Sarkhel et al. [2015] is probably the closest to our work. They design a C2F hierarchy to cluster constants for approximate lifted MAP inference in Markov logic. In contrast, we partition ground atoms in a PGM. Like other work on approximate lifting, they cannot handle distinct unary potentials. Furthermore, they assume that their theory is provided in a normal form, i.e., without evidence, which can be a severe restriction for most practical applications. Kiddon & Domingos [2011] also propose C2F inference for an underlying Markov logic theory. They use a hierarchy of partitions based on a pre-specified ontology. CV does not have any such ontology available, and needs to discover partitions using the PGM directly.

Nath & Domingos [2010] exploit (approximate) lifted inference for video segmentation. They experiment on a specific video problem (different from ours), and they only compare against vanilla BP. Their initial partitioning scheme is similar to our thresholding approach, which does not work well in our experiments.

In computer vision, a popular approach to reduce the complexity of inference is to use superpixels [Achanta *et al.*, 2012; Van den Bergh *et al.*, 2012]. Superpixels are obtained by merging neighboring nodes that have similar characteristics. All pixel nodes in the same superpixel are assigned the same value during MAP inference. SLIC [Achanta *et al.*, 2012] is one of the most popular algorithms for discovering superpixels. Our approach differs from SLIC in some significant ways. First, their superpixels are local in nature whereas our algorithm can merge pixels that are far apart as can be seen in Figure 9.2(e). This can help in merging two disconnected regions of the same object in a single lifted pixel. Second, they obtain superpixels independent of the inference algorithm, whereas we tightly integrate our lifting with the underlying inference algorithm. This can potentially lead to discovery of better partitions; indeed, this helped us tremendously in image segmentation. Third, they do not provide a C2F version of their algorithm and we did not find it straightforward to extend their approach to discover successively finer partitions. There is some recent work [Wei *et al.*, 2016] which addresses last two of these challenges by introducing a hierarchy of superpixels. In our preliminary experiments, we found that SLIC and superpixel hierarchy perform worse than our lifting approach.

9.6 Conclusion and Future Work

We develop a generic template for applying lifted inference to structured output prediction tasks in computer vision. We show that MRF-based CV algorithms can be lifted at different levels of abstraction, leading to methods for coarse to fine inference over a sequence of lifted models. We test our ideas on two different CV tasks of stereo matching and interactive image segmentation. We find that C2F lifting is vastly more efficient than unlifted algorithms on both tasks obtaining a superior anytime performance, and without any loss in final solution quality. To the best of our knowledge, this is the first demonstration of lifted inference in conjunction with top of the line task-specific algorithms. Although we restrict to CV in this work, we believe that our ideas are general and can be adapted to other domains such as NLP, and computational biology. We plan to explore this in the future.

Part V

Epilogue

Chapter 10

Conclusion and Future Directions

" A conclusion is the place where you get tired of thinking."

—Arthur Bloch

This thesis discusses the importance of exploiting symmetries to improve the efficiency and performance of AI and ML algorithms. Specifically, it discusses *symmetry aware algorithms* in the context of probabilistic inference and sequential decision making under uncertainty. In probabilistic inference, it characterizes different types of state symmetries that can be captured by graph automorphism. It also proposes novel notion of Contextual Symmetries, Variable-Value Symmetries, Non-Equicardinal Symmetries and Block-Value Symmetries. We also illustrates the effectiveness of these symmetries in end-to-end systems that use MCMC algorithms. Finally, we also study the hierarchy of state symmetries and relations among different types of state symmetries.

In the context of decision making, we study different notions of abstractions which uses symmetry. We find that existing notions of abstractions in MDPs only capture state abstractions by the ideas of bisimulations and homomorphisms. We extend these ideas to propose a novel notion of state-action pair abstractions which can provide gains even when there is no state abstraction present in the domain. In the line of recent work, we apply these abstractions in Monte Carlo Tree Search Algorithms. We develop a novel algorithm ASAP-UCT which applies abstraction of state-action pairs in UCT. We, further, improve the ASAP-UCT algorithm by applying abstractions on-the-go in UCT in our novel algorithm, OGA-UCT. Our experimental results on several benchmark domains illustrate the usefulness of applying ASAP abstractions in MCTS algorithms.

In addition, our work on applying symmetries in computer vision is one of the few works which have looked at symmetries from the application's point of view. We propose a novel top-k label heuristic which captures approximate symmetries for MAP inference in computer vision applications. Our coarse-to-fine Lifted MAP framework provided end-to-end time gains on near

state-of-the-art algorithms for the problems of Stereovision and Semantic Image Segmentation, thereby leveraging the theoretical study on lifted inference.

In addition, there are multiple research directions emanating from this thesis and contemporary research which can be pursued in future:

- **End-to-End Lifted Inference for Deeply Learned PGMs:** Most of the earlier works applying PGM inference to various tasks used hand tuned potentials. But recent works have shown that neural network learned representations can be used as potentials in the graphical models. A critical challenge in end to end learning these networks is that each learning step involves a computationally expensive PGM inference as a sub-step making learning not scale beyond a few iterations. We will like to address this challenge by the application of lifted inference which interestingly opens up following new directions:
  a) **Lifted Inference for end-to-end learning**: Since lifted inference provides speedup upto 10x, application of lifted inference in training of PGM augmented neural network models will make training significantly faster. The gain in time allows training algorithms to run many more iterations thereby improving the performance. Initial investigations by [Nandwani *et al.*, 2018] illustrate the benefit of using lifted inference to reduce the training time of PGM augmented neural networks significantly.
  b) **Learning to Learn Symmetries:** Most of the approximate lifted inference algorithms work on manually defined heuristics for calculating symmetries. The end to end system cannot leverage its power fully for these manually defined symmetries. In such a scenario, learning to automatically learn approximate symmetries is a critical component for an end-to-end system. How to devise an end-to-end differentiable symmetry learning module is a key question to be addressed here. With the advent of neural networks, Graph Convolutional Networks[Kipf and Welling, 2017] have become popular recently to find good representations for graph problems where the kernel parameters are shared across the complete graph. GCNs are able to embed graph nodes with similar neighbourhood structure by applying a message passing algorithm within a neural network. These parameters determine the size of neighborhood used for determining which nodes are symmetric to each other. It would be interesting to compute state symmetries in this way. One interesting insight could be to use these embeddings to compute symmetries instead of our graph automorphism / bisimulation based approach. This may help in computing approximate symmetric nodes as well. This could be clearly helpful in identifying states and/or actions which are symmetric.

- **Approximate Symmetries:** An important and forthcoming question for the lifted inference community is to adapt the ideas of theoretically sound symmetries into appropriate practical notions of approximate symmetries. There has been some work in this direc-

tion exemplified by the works of [Broeck and Niepert 2015], [Habeeeb et.al 2017]. A first step in this direction is to define different notions of relaxation and identify metrics to define and precisely quantify different types of approximations for real world symmetries. For e.g, defining bisimulation metric distance between two symmetric states could be one such metric. This should be motivated by the fact that certainalgorithmic guarantees should be preserved under these approximations like the convergence to stationary distributions. The second step is to compute these approximate symmetries fast. I believe representation learning by neural networks could play critical role in learning good approximate symmetries. Thirdly, some algorithmic innovations are required that our current lifted inference algorithms are applicable and flexible enough to identify symmetries early and handle wrongly identified approximate symmetries . We believe coarse-to-fine approach and On-the-Go abstractions have provided some of the first steps in this direction but we certainly believe more work is needed to push the ideas of lifted decision making and inference to real world domains

- **Symmetries for Resource Constrained ML:** Most of IoT devices and smart phones need to make some prediction for their usual operations. Though the capability of these devices are no less than a small computer, these devices operate in resource constrained settings of memory, battery power and CPUs. To avoid the communication delays and amidst privacy concerns, there has been a lot of push to do predictions on device in limited memory and power scenario([Kumar *et al.*, 2017]). Some works([McMahan *et al.*, 2017]) have pushed to do training updates on the device giving rise to paradigm of Federated Learning. In such scenarios, abstractions play a critical role in not only reducing the prediction time but also reducing the model size. I would like to explore how abstractions discussed above can be computed efficiently in such scenarios of limited resources. I will look forward to collaborations with people having experience in embedded devices for this work.

Bibliography

[Achanta *et al.*, 2012] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Ssstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *In PAMI*, Nov 2012.

[Anand *et al.*, 2015a] Ankit Anand, Aditya Grover, Mausam, and Parag Singla. ASAP-UCT: Abstraction of State-Action Pairs in UCT. In *IJCAI*, pages 1509–1515, 2015.

[Anand *et al.*, 2015b] Ankit Anand, Aditya Grover, Mausam, and Parag Singla. A Novel Abstraction Framework for Online Planning. In *AAMAS*, 2015.

[Anand *et al.*, 2016a] A. Anand, A. Grover, Mausam, and P. Singla. Contextual Symmetries in Probabilistic Graphical Models. In *IJCAI*, 2016.

[Anand *et al.*, 2016b] Ankit Anand, Ritesh Noothigattu, Mausam, and Parag Singla. OGA-UCT: On-the-Go Abstractions in UCT. In *ICAPS*, 2016.

[Anand *et al.*, 2017] A. Anand, R. Noothigattu, P. Singla, and Mausam. Non-Count Symmetries in Boolean & Multi-Valued Prob. Graphical Models. In *AISTATS*, 2017.

[Andres *et al.*, 2010] B. Andres, J. H. Kappes, U. Köthe, C. Schnörr, and F. A. Hamprecht. An Empirical Comparison of Inference Algorithms for Graphical Models with Higher Order Factors Using OpenGM. In *Pattern Recognition*. 2010.

[Baier and Winands, 2012] Hendrik Baier and Mark HM Winands. Time Management for Monte-Carlo Tree Search in Go. In *Advances in Computer Games*. Springer, 2012.

[Balla and Fern, 2009] Radha-Krishna Balla and Alan Fern. UCT for Tactical Assault Planning in Real-Time Strategy Games. In *IJCAI*, pages 40–45, 2009.

[Barnsley, 2014] Michael F Barnsley. *Fractals Everywhere*. Academic press, 2014.

[Barto *et al.*, 1995] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72(1-2):81–138, January 1995.

[Baudiš and Gailly, 2012] Petr Baudiš and Jean-loup Gailly. Pachi: State of the art open source Go program. In *Advances in Computer Games*. Springer, 2012.

[Bellman, 1957] Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 1957.

[Bertsekas, 1995] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1st edition, 1995.

[Birchfield and Tomasi, 1999] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3):269–293, 1999.

[Blei *et al.*, 2003] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *JMLR*, 3, March 2003.

[Bonet and Geffner, 2012] Blai Bonet and Hector Geffner. Action Selection for MDPs: Anytime AO* Versus UCT. In *AAAI*, 2012.

[Boutilier *et al.*, 1996] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific Independence in Bayesian Networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, UAI'96, pages 115–123, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[Boykov and Jolly, 2001] Yuri Y Boykov and M-P Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.

[Boykov *et al.*, 2001] Y. Boykov, O. Veksler, and R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *In PAMI*, 23(11), November 2001.

[Braz *et al.*, 2005] R. Braz, E. Amir, and D. Roth. Lifted First-Order Probabilistic Inference. In *IJCAI*, 2005.

[Browne *et al.*, 2012] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[Bui *et al.*, 2012] H. Bui, T. Huynh, and R. De Salvo Braz. Exact Lifted Inference with Distinct Soft Evidence on Every Object. In *AAAI*, 2012.

[Bui *et al.*, 2013] H. Bui, T. Huynh, and S. Riedel. Automorphism Groups of Graphical Models and Lifted Variational Inference. In *UAI*, 2013.

[Byers, 1998] N. Byers. E. Noether's Discovery of the Deep Connection Between Symmetries and Conservation Laws. *ArXiv Physics e-prints*, July 1998.

[Campbell *et al.*, 2002] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[Cohen *et al.*, 2006] David Cohen, Peter Jeavons, Christopher Jefferson, Karen E Petrie, and Barbara M Smith. Symmetry Definitions for Constraint Satisfaction Problems. *Constraints*, (2-3):115–137, 2006.

[Crawford *et al.*, 1996] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, pages 148–159, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[Darga *et al.*, 2008] Paul T Darga, Karem A Sakallah, and Igor L Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th annual Design Automation Conference*, pages 149–154. ACM, 2008.

[Domingos and Lowd, 2009] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

[Ferns *et al.*, 2004] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for Finite Markov Decision Processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 162–169, Arlington, Virginia, United States, 2004. AUAI Press.

[Freeman *et al.*, 2000] W. Freeman, E. Pasztor, and O. Carmichael. Learning Low-Level Vision. *In IJCV*, 40, 2000.

[Friedman, 2004] N. Friedman. Inferring Cellular Networks using Probabilistic Graphical Models. *Science*, 303, 2004.

[GAP, 2015] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.7.9*, 2015.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[Gent *et al.*, 2005] Ian P Gent, Tom Kelsey, Steve A Linton, Iain McDonald, Ian Miguel, and Barbara M Smith. Conditional Symmetry Breaking. In *Principles and Practice of Constraint Programming*. 2005.

[Gent *et al.*, 2007] Ian P Gent, Tom Kelsey, Stephen A Linton, Justin Pearson, and Colva M Roney-Dougal. Groupoids and Conditional Symmetry. In *Principles and Practice of Constraint Programming*. 2007.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[Givan *et al.*, 2003] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(12):163 – 223, 2003.

[Gogate and Domingos, 2011] V. Gogate and P. Domingos. Probabilisitic Theorem Proving. In *UAI*, 2011.

[Gogate *et al.*, 2012] V. Gogate, A. Jha, and D. Venugopal. Advances in lifted importance sampling. In *Proc. of AAAI-12*, pages 1910–1916, 2012.

[Grzes *et al.*, 2014] Marek Grzes, Jesse Hoey, and Scott Sanner. International Probabilistic Planning Competition (IPPC) 2014. In *ICAPS*, 2014.

[Guerin *et al.*, 2012] Joshua T Guerin, Josiah P Hanna, Libby Ferland, Nicholas Mattei, and Judy Goldsmith. The academic advising planning domain. *WS-IPC 2012*, page 1, 2012.

[Guestrin *et al.*, 2003] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *J. Artif. Intell. Res.(JAIR)*, 19:399–468, 2003.

[Gupta *et al.*, 2010] Rahul Gupta, Sunita Sarawagi, and Ajit A. Diwan. Collective inference for extraction mrfs coupled with symmetric clique potentials. *J. Mach. Learn. Res.*, 11:3097–3135, December 2010.

[Habeeb *et al.*, 2017] Haroun Habeeb, Ankit Anand, Mausam, and Parag Singla. Coarse-to-fine Lifted MAP Inference in Computer Vision. In *IJCAI*, 2017.

[Hartmanis, 1966] Juris Hartmanis. *Algebraic Structure Theory of Sequential Machines (Prentice-Hall International Series in Applied Mathematics)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1966.

[Hirschmuller and Scharstein, 2007] H. Hirschmuller and D. Scharstein. Evaluation of Cost Functions for Stereo Matching. In *CVPR*, 2007.

[Hostetler *et al.*, 2014] Jesse Hostetler, Alan Fern, and Tom Dietterich. State Aggregation in Monte Carlo Tree Search. In *AAAI*, 2014.

[Hostetler *et al.*, 2015] Jesse Hostetler, Alan Fern, and Thomas Dietterich. Progressive abstraction refinement for sparse sampling. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2015.

[Howard, 1960] Ronald A. Howard. Dynamic programming and markov processes, 1960.

[Jain, 2017] Tapas Jain. Prost vs. OGA-UCT for Probabilistic Planning. 2017.

[Jegelka and Bilmes, 2011] S. Jegelka and J. Bilmes. Submodularity Beyond Submodular Energies: Coupling Edges in Graph Cuts. In *CVPR*, 2011.

[Jernite *et al.*, 2015] Y. Jernite, A. Rush, and D. Sontag. A Fast Variational Approach for Learning Markov Random Field Language Models. In *ICML*, 2015.

[Jha *et al.*, 2010] A. Jha, V. Gogate, A. Meliou, and D. Suciu. Lifted Inference Seen from the Other Side : The Tractable Features. In *NIPS*, 2010.

[Jiang *et al.*, 2014] Nan Jiang, Satinder Singh, and Richard Lewis. Improving UCT Planning via Approximate Homomorphisms. In *AAMAS*, 2014.

[Jiang *et al.*, 2015] Nan Jiang, Alex Kulesza, and Satinder Singh. Abstraction selection in model-based reinforcement learning. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 179–188, 2015.

[Kappes *et al.*, 2015] J. Kappes, B. Andres, A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. A Comparative Study of Modern Inference Techniques for Structured Discrete Energy Minimization Problems. *In IJCV*, 2015.

[Kearns *et al.*, 2002] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Mach. Learn.*, 49(2-3):193–208, November 2002.

[Keller and Eyerich, 2012] Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *ICAPS*, 2012.

[Kersting *et al.*, 2009] K. Kersting, B. Ahmadi, and S. Natarajan. Counting Belief Propagation. In *UAI*, 2009.

[Kiddon and Domingos, 2011] Chloé Kiddon and Pedro M Domingos. Coarse-to-Fine Inference and Learning for First-Order Probabilistic Models. In *AAAI*, 2011.

[Kimmig *et al.*, 2015] A. Kimmig, L. Mihalkova, and L. Getoor. Lifted Graphical Models: A Survey. *Machine Learning*, 2015.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML*. Springer, 2006.

[Kohli *et al.*, 2013] P. Kohli, A. Osokin, and S. Jegelka. A Principled Deep Random Field Model for Image Segmentation. In *CVPR*, 2013.

[Koller and Friedman, 2009] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[Kolmogorov, 2006] V. Kolmogorov. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, Oct 2006.

[Kolobov *et al.*, 2012] Andrey Kolobov, Mausam, and Daniel S. Weld. LRTDP versus UCT for online probabilistic planning. In *AAAI*, 2012.

[Kopp *et al.*, 2015] Timothy Kopp, Parag Singla, and Henry Kautz. Lifted symmetry detection and breaking for map inference. In *Advances in Neural Information Processing Systems*, pages 1315–1323, 2015.

[Koriche *et al.*, 2017] Frdric Koriche, Sylvain Lagrue, ric Piette, and Sbastien Tabary. Constraint-Based Symmetry Detection in General Game Playing. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 280–287, 2017.

[Kumar *et al.*, 2017] A. Kumar, S Goyal, and M. Varma. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In *ICML*, August 2017.

[Lempitsky *et al.*, 2010] V. Lempitsky, C. Rother, S. Roth, and A. Blake. Fusion Moves for Markov Random Field Optimization. *In PAMI*, Aug 2010.

[Li *et al.*, 2006] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a Unified Theory of State Abstraction for MDPs. In *ISAIM*, 2006.

[Madan *et al.*, 2018] Gagan Madan, Ankit Anand, Mausam, and Parag Singla. Block Value Symmetries in Probabilistic Graphical Models. In *UAI*, 2018.

[Mausam and Kolobov, 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers, 2012.

[McMahan *et al.*, 2017] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, pages 1273–1282, 2017.

[Milch *et al.*, 2008] B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaebling. Lifted probabilistic inference with counting formulas. In *Proc. of AAAI-08*, 2008.

[Miller, 1973] Willard Miller. *Symmetry Groups and Their Applications*, volume 50. Academic Press, 1973.

[Mittal *et al.*, 2014] H. Mittal, P. Goyal, V. Gogate, and P. Singla. New Rules for Domain Independent Lifted MAP Inference. In *NIPS*, 2014.

[Mittal *et al.*, 2015] H. Mittal, A. Mahajan, V. Gogate, and P. Singla. Lifted Inference Rules With Constraints. In *NIPS*, 2015.

[Mladenov *et al.*, 2014] M. Mladenov, K. Kersting, and A. Globerson. Efficient Lifting of MAP LP Relaxations Using k-Locality. In *AISTATS*, 2014.

[Mozerov and van de Weijer, 2015] M. G. Mozerov and J. van de Weijer. Accurate Stereo Matching by Two-Step Energy Minimization. *IEEE Transactions on Image Processing*, March 2015.

[Nandwani *et al.*, 2018] Yatin Nandwani, Ankit Anand, Mausam, and Parag Singla. Lifted Inference for Faster Training (LIFT) in End-to-End Neural-CRF Models. In *Proceedings of the Workshop on Relational Representation Learning (R2L), NeurIPS*, 2018.

[Nath and Domingos, 2010] A. Nath and P. Domingos. Efficient Lifting for Online Probabilistic Inference. In *AAAIWS*, 2010.

[Nath and Domingos, 2016] A. Nath and P. Domingos. Learning Tractable Probabilistic Models for Fault Localization. In *AAAI*, 2016.

[Niepert and Domingos, 2014] Mathias Niepert and Pedro Domingos. Exchangeable variable models. In *International Conference on Machine Learning*, pages 271–279, 2014.

[Niepert and Van den Broeck, 2014] Mathias Niepert and Guy Van den Broeck. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[Niepert, 2012] M. Niepert. Markov Chains on Orbits of Permutation Groups. In *UAI*, 2012.

[Noessner *et al.*, 2013] J. Noessner, M. Niepert, and H. Stuckenschmidt. RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models. In *AAAI*, 2013.

[Noether, 1971] E. Noether. Invariant Variation Problems. *Transport Theory and Statistical Physics*, 1:186–207, January 1971.

[Pak, 2000] I. Pak. The product replacement algorithm is polynomial. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 476–485, 2000.

[Park, 1981] David Park. Concurrency and Automata on Infinite Sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, UK, 1981. Springer-Verlag.

[Poole, 2003] D. Poole. First-Order Probabilistic Inference. In *IJCAI*, 2003.

[Puterman, 1994] M.L. Puterman. *Markov Decision Processes*. John Wiley & Sons, Inc., 1994.

[Ravindran and Barto, 2004] Balaraman Ravindran and Andrew Barto. Approximate homomorphisms: A framework for nonexact minimization in Markov decision processes. In *Int. Conf. Knowledge-Based Computer Systems*, 2004.

[Ravindran, 2004] Balaraman Ravindran. *An Algebraic Approach to Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2004.

[Richardson and Domingos, 2006] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62, 2006.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[Sanner and Yoon, 2011] Scott Sanner and Sungwook Yoon. International Probabilistic Planning Competition (IPPC) 2011. In *ICAPS*, 2011.

[Sanner, 2010] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. 2010.

[Sarkhel *et al.*, 2014] S. Sarkhel, D. Venugopal, P. Singla, and V. Gogate. Lifted MAP inference for Markov logic networks. In *AISTATS*, 2014.

[Sarkhel *et al.*, 2015] S. Sarkhel, P. Singla, and V. Gogate. Fast Lifted MAP Inference via Partitioning. In *NIPS*, 2015.

[Scharstein and Szeliski, 2002] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *In IJCV*, 2002.

[Scharstein and Szeliski, 2003] D. Scharstein and R. Szeliski. High-accuracy Stereo Depth Maps Using Structured Light. In *CVPR*, pages 195–202, 2003.

[Schiffel, 2010] Stephan Schiffel. Symmetry detection in general game playing. In *AAAI*, 2010.

[Sharma *et al.*, 2018] Vishal Sharma, Noman Ahmed Sheikh, Happy Mittal, Vibhav Gogate, and Parag Singla. Lifted Marginal MAP Inference. In *UAI*, 2018.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484 EP –, Jan 2016. Article.

[Singla and Domingos, 2008] P. Singla and P. Domingos. Lifted First-Order Belief Propagation. In *AAAI*, 2008.

[Singla *et al.*, 2014] P. Singla, A. Nath, and P. Domingos. Approximate Lifting Techniques for Belief Propagation. In *AAAI*, 2014.

[Srinivasan *et al.*, 2015] Sriram Srinivasan, Erik Talvitie, and Michael Bowling. Improving Exploration in UCT Using Local Manifolds. In *AAAI Conference on Artificial Intelligence*, 2015.

[Steindel, 2017] Tim Steindel. Enhancing Prost with Abstraction of State-Action Pairs. 2017.

[Sutton and Barto, 1998] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 1999] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[Szeliski *et al.*, 2008a] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors. *In PAMI*, June 2008.

[Szeliski *et al.*, 2008b] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE transactions on pattern analysis and machine intelligence*, 30(6):1068–1080, 2008.

[Tarlow *et al.*, 2010] Daniel Tarlow, Inmar Givoni, and Richard Zemel. Hop-map: Efficient message passing with high order potentials. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 812–819, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[Van den Bergh *et al.*, 2012] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. Van Gool. *SEEDS: Superpixels Extracted via Energy-Driven Sampling*. 2012.

[Van den Broeck and Darwiche, 2013] G. Van den Broeck and A. Darwiche. On the Complexity and Approximation of Binary Evidence in Lifted Inference. In *NIPS*, 2013.

[Van den Broeck and Niepert, 2015] G. Van den Broeck and M. Niepert. Lifted Probabilistic Inference for Asymmetric Graphical Models. In *AAAI*, 2015.

[Van den Broeck *et al.*, 2011] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proc. of IJCAI-11*, 2011.

[Venugopal and Gogate, 2012] D. Venugopal and V. Gogate. On lifting the Gibbs sampling algorithm. In *Proc. of NIPS-12*, pages 1664–1672, 2012.

[Venugopal and Gogate, 2014a] D. Venugopal and V. Gogate. Evidence-Based Clustering for Scalable Inference in Markov Logic. In *Joint ECML-KDD*, 2014.

[Venugopal and Gogate, 2014b] D. Venugopal and V. Gogate. Scaling-up importance sampling for Markov logic networks. In *NIPS*, pages 2978–2986, 2014.

[Walsh, 2006] Toby Walsh. General symmetry breaking constraints. In *Principles and Practice of Constraint Programming*. 2006.

[Wei *et al.*, 2016] X. Wei, Q. Yang, Y. Gong, M. Yang, and N. Ahuja. Superpixel Hierarchy. *CoRR*, abs/1605.06325, 2016.

[Wielandt, 2014] Helmut Wielandt. *Finite permutation groups*. Academic Press, 2014.

Appendix

Theorem 2.3.1: PGM-Num_Orbit is NP-Hard.

Proof. We show that given any 3-SAT formula, we can construct a PGM, such that the formula is not satisfiable if and only if the number of orbits in the coarsest probability preserving partitioning is ≤ 2. The construction of this PGM should be polynomial in the size of the input for 3-SAT.

Consider an instance of 3-SAT over variables $Q_1, Q_2, \cdots, Q_n$ and clauses $C_1, C_2, \cdots, C_m$. **Let us assume for now that the given problem instance is not a tautology.** We will handle this case later. We construct a graphical model as shown in Figure 10.1. Every clause C_i is connected to the variables that appear in the clause. We define a feature over each clause and the variables that appear in the clause. Consider the feature for the i^{th} clause, defined over variables Q_j, Q_k, Q_l. The feature has weight 1 and is true if the assignment to the Q_j, Q_k, Q_l "agrees" with the assignment to C_i, and false otherwise. Note that it is not necessary for value of C_i to be 1 for feature to be true. It could also be possible that C_i has an assignment 0 and the feature is still true, as long as the value "agrees" with the assignment to Q_j, Q_k, Q_l.

Further we have some additional variables A_i's and we define a feature associated with each A_i in a similar way. This feature having weight 1 is true if $A_i = C_{i+1} \wedge A_{i-1}$ and false otherwise. Finally we define a feature with weight 3 associated with the node X as true if $X = 1$ and $X = C_m \wedge A_{m-2}$. In addition, we define a feature with weight 1 which is true if $X = 0$ and $X = C_m \wedge A_{m-2}$ and false otherwise.

Note that in case there is some satisfying assignment to the variables $Q_1, Q_2, \cdots, Q_n$, the unnormalized probability for the state corresponding to this assignment and $\{C_1, C_2, \cdots, C_m, A_1, A_2, \cdots, A_{m-2}, X\} = \{1, 1, \cdots, 1\}$ is 3. Since this is not a tautology, there would be some assignment to $Q_1, Q_2, \cdots, Q_n$, such that some clause is violated. In this case, the unnormalized probability for the complete assignment to the state is 1. Also, there would always be some states with unnormalized probability 0, which would be the states in which the values for some C_i or some A_i or X do not "agree" with the required condition.

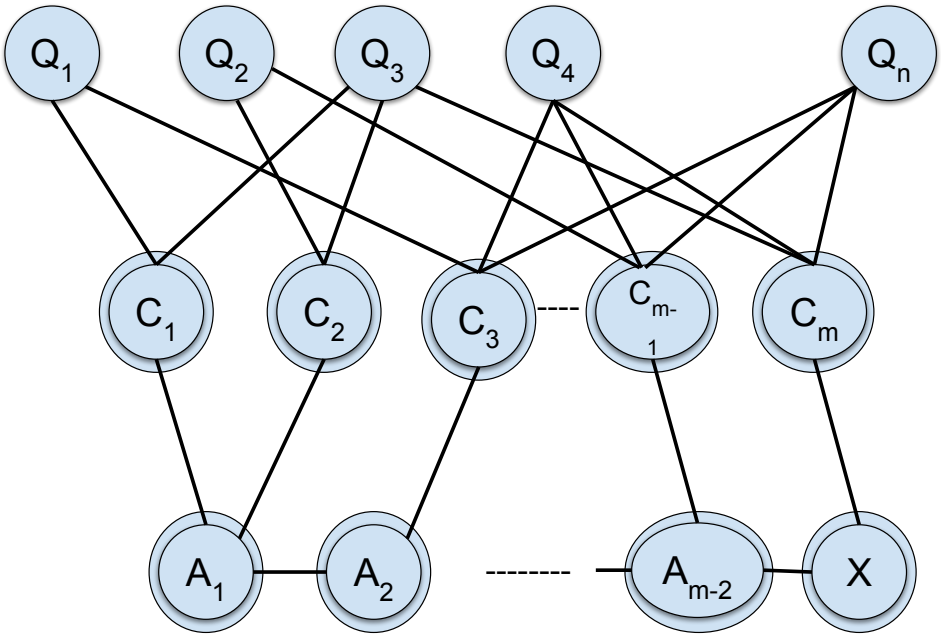


Figure 10.1: Outline of Network Structure for Reduction. Source: Adapted from [Koller and Friedman, 2009]

Therefore, in case there is a satisfying assignment, the number of orbits in the **coarsest valid** partitioning would be 3, and in case there is no satisfying assignment, it would be 2.

Now consider the case in which the given problem instance is a tautology. Since the problem is in CNF form, it can be checked in linear time that each clause is a tautology. In case it is a tautology, we specify a fixed graphical model that has number of orbits in the **coarsest valid** partitioning to be 3.

Therefore, the graphical model we construct would have the number of orbits in coarsest probability preserving state partitioning to be 3 (for unnormalized probabilities - 0,1 and 3) if and only if there is a satisfying assignment to the variables, and the number of orbits in coarsest probability partitioning to be 2 if and only if there is no satisfying assignment to the variables. Giving this as an input to **PGM-NumSym** with the graphical model defined as above, and setting k=2, if the solver for **PGM-NumSym** outputs a Yes, then the given instance for **3-SAT** is not satisfiable and if it outputs a No, then the given instance for **3-SAT** is satisfiable.

□

List of Publications

This thesis is based on the following publications:

1. "Block-Value Symmetries in Probabilistic Graphical Models". Gagan Madan, Ankit Anand, Mausam, Parag Singla. Conference on Uncertainty in Artificial Intelligence (UAI). Monterey, CA, USA. August 2018.

2. "Coarse-to-Fine Lifted MAP Inference in Computer Vision". Haroun Habeeb, Ankit Anand, Mausam, Parag Singla. International Joint Conference on Artificial Intelligence (IJCAI). Melbourne, Australia. August 2017.

3. "Non-Count Symmetries in Boolean & Multi-Valued Prob. Graphical Models". Ankit Anand[1], Ritesh Noothigattu[1], Parag Singla, Mausam. International Conference on Artificial Intelligence and Statistics (AISTATS). Fort Lauderdale, Florida. April 2017.

4. "Contextual Symmetries in Probabilistic Graphical Models". Ankit Anand, Aditya Grover, Mausam, Parag Singla. International Joint Conference on Artificial Intelligence (IJCAI). New York, NY. July 2016.

5. "OGA-UCT: On the Go Abstractions in UCT". Ankit Anand[1], Ritesh Noothigattu[1], Mausam, Parag Singla. International Conference on Automated Planning and Scheduling (ICAPS). London, United Kingdom. June 2016.

6. "ASAP-UCT: Abstraction of State-Action Pairs in UCT". Ankit Anand, Aditya Grover, Mausam, Parag Singla. International Joint Conference on Artificial Intelligence (IJCAI).

[1] Shows Equal Contribution

Buenos Aires, Argentina. July 2015.

7. “A Novel Abstraction Framework for Online Planning”. Ankit Anand, Aditya Grover, Mausam, Parag Singla. Extended Abstract at International Conference on Autonomous Agents and Multi-agent Systems (AAMAS). Istanbul, Turkey. May 2015.

Biography

Ankit Anand pursued Ph.D at the Department of Computer Science and Engineering, Indian Institute of Technology Delhi from 2013-2018. He obtained a Masters degree in Computational Science from the Department of Computational and Data Sciences (formerly Super-computer Education and Research Centre), Indian Institute of Science, Bangalore in 2013. He has a Bachelors degree in Computer Science & Engineering from Punjab Engineering College(PEC), Chandigarh. His research interests include Probabilistic Graphical Models, Reinforcement Learning and Artificial Intelligence Planning.