

Thèse de Doctorat

Rajani CHULYADYO

*Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans*

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27

Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 19 octobre 2016

A new horizon for the recommendation
Integration of spatial dimensions to aid decision making

JURY

Rapporteurs : **M. Christophe GONZALES**, Professeur des universités, Université Pierre-et-Marie-Curie (Paris VI)
M. Nicolas LACHICHE, Maître de conférences, Université de Strasbourg

Examineurs : **M. Colin DE LA HIGUERA**, Professeur, Université de Nantes
M^{me} Armelle BRUN, Maître de conférences, Université de Lorraine

Invité : **M. Cédric HOUSSIN**, Directeur, DataForPeople

Directeur de thèse : **M. Philippe LERAY**, Professeur des universités, Université de Nantes

Acknowledgments

This thesis would not have been possible without the inspiration and continuous support of a number of people.

I would like to express my deep gratitude to Prof. Philippe Leray for his supervision of my thesis. His continuous guidance, suggestions, and optimism over the last years have been invaluable to advance in my research. I feel very fortunate to be advised by him and to benefit from his expertise on this interesting area of research.

I am very grateful to DataForPeople for supporting this research. My special thanks go to Mr. Romain Perruchon, former CTO of DataForPeople, and Mr. Cédric Houssin, CEO of DataForPeople, for their guidance on the professional aspect of this thesis. Their expertise in the projects of DataForPeople has been a great help during the thesis. I am thankful to both of them as well as Mr. Pierre-Yves Huan, COO of DataForPeople, for creating a cordial working environment and helping me to integrate to the world of start-ups.

I would like to express my sincere gratitude to my colleagues, Anthony Coutant, Mouna Ben Ishak, and Thomas Vincent, for always sharing their expertise and experiences, and making the development of PILGRIM fun. Their domain expertise has helped me get good insights of PILGRIM.

Additionally, I am very thankful to my friends, Prajol and Trija for making me feel home away from home, Amandine and Mickaël for helping me learn French, and Amit, Niroj and Parbati for their company and emotional support.

I would also like to thank all those who have helped me directly or indirectly in all stages of this thesis.

More than anything else, I am deeply grateful to my mom, dad, brother, and husband for their love and continuous moral support.

Thanks!

Contents

1	Introduction	1
1.1	Context	2
1.2	Motivation and problem statement	4
1.3	Contributions	6
1.4	Organization of the dissertation	7
I	State-of-the-art	9
2	Probabilistic Relational Models for Relational Learning	11
2.1	Introduction	12
2.2	Background	13
2.2.1	Relational data representation	13
2.2.2	Bayesian Networks	16
2.3	Probabilistic Relational Models (PRMs)	20
2.4	Extensions	24
2.4.1	PRM with structural uncertainty	24
2.4.2	Other extensions	24
2.5	Inference in PRMs	24
2.6	Learning PRMs	25
2.6.1	Learning parameters	25
2.6.2	Learning structure	26
2.7	Evaluating PRM learning algorithms	28
2.7.1	Evaluation strategy and metrics	29
2.7.2	Generating PRM benchmarks	30
2.7.3	Limitations	31
2.7.4	Proposals for improvement	32
2.8	Conclusion	40
3	Recommender Systems: A Common Application of Relational Data	41
3.1	Introduction	42
3.2	Recommendation models and techniques	43
3.2.1	Recommendation data	44
3.2.2	Recommendation techniques	45
3.2.3	New developments	51
3.3	Evaluation of recommender systems	51
3.3.1	Evaluation approaches	52
3.3.2	Accuracy metrics	52
3.3.3	Other evaluation metrics	54
3.3.4	Benchmark datasets and evaluation tools	55

3.4	Challenges	57
3.5	Conclusion	59
4	Using Probabilistic Relational Models for Recommendation	61
4.1	Introduction	62
4.2	Existing approaches	62
4.2.1	Collaborative Filtering using PRMs (Getoor and Sahami [1999])	62
4.2.2	A unified recommendation framework based on PRMs (Huang et al. [2004])	63
4.2.3	Hierarchical Probabilistic Relational Models (hPRM) (Newton and Greiner [2004])	64
4.2.4	Combining User Grade-based Collaborative Filtering and PRMs (UGCF-PRM) (Gao et al. [2007])	65
4.2.5	A RBN-based recommender system architecture (Ben Ishak et al. [2013])	65
4.3	Comparison and discussion	66
4.4	Conclusion	69
5	Spatial Data	71
5.1	Introduction	72
5.2	Spatial data representation	72
5.2.1	Tessellation data representation	73
5.2.2	Vector data representation	73
5.2.3	Network data type	74
5.3	Characteristics of spatial data	75
5.3.1	Spatial heterogeneity	75
5.3.2	Spatial autocorrelation	75
5.4	Spatial operators	76
5.4.1	Metric operators	76
5.4.2	Topological operators	76
5.5	Conclusion	78
6	Recommender Systems with Spatial Data	79
6.1	Introduction	80
6.2	Review of some spatial recommender systems	80
6.3	Discussion	82
6.4	Conclusion	84
II	Contributions	85
7	A Personalized Recommender System	87
7.1	Introduction	88
7.2	The proposed approach	90
7.2.1	PRM for preference-based recommendation (PRM-PrefReco)	90
7.2.2	Personalization	93
7.2.3	Relational attributes and types of model	96
7.2.4	Examples	98
7.3	Experiments	101
7.3.1	Dataset	101

7.3.2	Experiment methodology	102
7.3.3	Evaluation metrics	102
7.3.4	Results and discussion	103
7.4	Conclusion	104
8	PRM with Spatial Attributes (PRM-SA)	105
8.1	Introduction	106
8.2	Definitions	106
8.3	Learning PRM-SA	112
8.4	Evaluation of PRM-SA learning algorithms	116
8.4.1	Evaluation strategy and metrics	117
8.4.2	Generation of PRM-SA benchmarks	117
8.5	Experimental study	122
8.5.1	Methodology	123
8.5.2	Results and discussion	126
8.6	PRMs-SA in recommender systems	131
8.7	Conclusion	132
9	Implementations in PILGRIM	135
9.1	An Introduction to PILGRIM	136
9.2	Technical aspects	137
9.3	PILGRIM-Relational	137
9.3.1	Modules	139
9.3.2	Implementation of PRM-SA	144
9.3.3	Implementation of PRM benchmark generation	145
9.4	PILGRIM-Applications	148
9.5	Conclusion	150
10	Conclusion	151
III	Appendices	155
A	Empirical Study of PRM Sampling Algorithms	157
A.1	Empirical study of relational block Gibbs sampling algorithm	157
A.1.1	Methodology	157
A.1.2	Results and discussion	160
A.2	Comparison of sampling algorithms	162
A.2.1	Methodology	162
A.2.2	Results and discussion	163
A.3	Conclusion	165
B	Using PILGRIM	167
B.1	Defining a PRM(-SA)	167
B.1.1	Defining a relational schema	167
B.1.2	Defining a dependency structure	170
B.1.3	Defining parameters	174
B.2	Instantiating a PRM for making inference	175
B.3	Utility methods	176
B.3.1	Exporting/Importing a PRM	176

B.3.2	Exporting a relational schema into a database	176
B.4	Generating datasets from a PRM	176
B.4.1	Generating a random skeleton	177
B.4.2	Sampling a PRM	177
B.5	Working with PILGRIM-Recommendier	179
B.5.1	Defining a recommendation model	179
B.5.2	Making recommendations	181
C	Detailed Results of PRM-SA Learning Algorithm Evaluation	183

List of Tables

2.1	Notations and their meaning	14
3.1	Some datasets used for recommendation systems research purposes . . .	56
4.1	Comparison of some PRM-based recommendation approaches	68
5.1	Examples of spatial operators	77
6.1	Comparison of some recommender systems that exploit spatial data . . .	83
7.1	The scale of absolute numbers for expressing the relative importance of a pair of decision factors (Saaty [2008])	95
7.2	CPD of $P(\text{Tx.DF_furnished} \mid \text{Search.furnished}, \text{Tx.Property.furnished})$. .	99
7.3	Evaluation result	104
8.1	Average \pm standard deviation of metrics for four PRM-SA structure learning algorithms in the experiment.	128
8.2	Average \pm standard deviation of metrics to measure spatial influence in the learned models	129
9.1	PILGRIM modules, and contributions made by the team members . . .	138
9.2	Summary of major functionalities implemented in PILGRIM-Relational	140
9.3	Summary of major functionalities implemented in PILGRIM-Recommender	149
C.1	Average \pm standard deviation of Hard Precision for PRM-SA structure learning algorithms for models of Figure 8.6.	183
C.2	Average \pm standard deviation of Hard Recall for PRM-SA structure learning algorithms for models of Figure 8.6.	184
C.3	Average \pm standard deviation of hard F-score for PRM-SA structure learning algorithms for models of Figure 8.6.	184
C.4	Average \pm standard deviation of Soft Precision for PRM-SA structure learning algorithms for models of Figure 8.6.	184
C.5	Average \pm standard deviation of Soft Recall for PRM-SA structure learning algorithms for models of Figure 8.6.	185
C.6	Average \pm standard deviation of Soft F-score for PRM-SA structure learning algorithms for models of Figure 8.6.	185
C.7	Average \pm standard deviation of Soft Precision _{skeleton} of models of Figure 8.6.	186
C.8	Average \pm standard deviation of Soft Recall _{skeleton} of models of Figure 8.6	186
C.9	Average \pm standard deviation of Soft F-score _{skeleton} of models of Figure 8.6	186
C.10	Average \pm standard deviation of Hard Precision _{spatial} of models of Figure 8.6	187

C.11 Average \pm standard deviation of Hard Recall _{spatial} of models of Figure 8.6	187
C.12 Average \pm standard deviation of hard F-score _{spatial} of models of Figure 8.6	187
C.13 Average \pm standard deviation of Soft Precision _{spatial} of models of Figure 8.6	188
C.14 Average \pm standard deviation of Soft Recall _{spatial} of models of Figure 8.6	188
C.15 Average \pm standard deviation of Soft F-score _{spatial} of models of Figure 8.6	189
C.16 Average \pm standard deviation of normalized mutual information(NMI) between the spatial partition learned during the experiment, and the original spatial partition for the spatial attribute (<i>User.location</i>) of each model of Figure 8.6	189
C.17 Absolute difference between Bayesian Dirichlet score of the gold models and that of the learned models.	190

List of Figures

2.1	(a) An example of a relational schema, (b) Crow’s foot notations	15
2.2	Instantiations of the relational schema of Figure 2.1a	17
2.3	An example of a Bayesian network (Pearl [1988])	18
2.4	An example of a PRM corresponding to the relational schema of Figure 2.1a	22
2.5	Ground Bayesian network obtained by unrolling the PRM of Figure 2.4 over the relational skeleton of Figure 2.2	23
2.6	Class dependency graph of the PRM in Figure 2.4	23
2.7	Overview of PRM benchmark generation process proposed by Ben Ishak [2015].	31
2.8	(a) Relational schema DAG of Figure 2.1a (considering only three classes), (b) Relational skeleton of Figure 2.2a as a k -partite graph	33
2.9	A dummy relational schema: (a) Entity-Relationship diagram, (b) as a directed graph	37
2.10	Generating objects while performing Depth First Search (DFS) on the relational schema. This shows one iteration of a DFS performed on the schema.	38
2.11	Next two iterations of DFS on the relational schema of Figure 2.9 following the first iteration of Figure 2.10 to generate relational skeleton graph.	38
3.1	Data representations commonly used in recommender systems	45
3.2	Recommendation approaches	47
3.3	Hybridization techniques (Burke [2007], Jannach and Friedrich [2013])	49
4.1	A Bayesian network for two-sided clustering (Getoor and Sahami [1999])	63
4.2	(a) Hierarchy of Movie class and (b) Hierarchical Probabilistic Relational Model (hPRM) proposed by Newton and Greiner	64
4.3	Overview of the recommender approach proposed by Gao et al. [2007]	65
4.4	The overall architecture of the recommender system proposed by Ben Ishak et al. [2013]	66
4.5	Sample hierarchies an hPRM cannot address	67
5.1	Examples of spatial data	74
6.1	Partial pyramid structure proposed by Levandoski et al. [2012] for partitioning users based on their location.	81
6.2	An example of a location-based social network graph (Wang et al. [2013])	82
7.1	Relational schema of our proposed preference-based recommender system	91

7.2	The proposed PRM (a) before introducing decision factors, and (b) after introducing decision factors.	93
7.3	Three types of proposed recommendation model	97
7.4	(a) A screenshot of Kyzia, (b) Relational schema of Kyzia, (c) PRM-PrefReco for Kyzia.	99
7.5	Re-formulation of Delcroix and Ben Mrad [2016]’s approach of modeling decision criteria by V-structures in a Bayesian network into a PRM-PrefReco	101
8.1	(a) An example of a relational schema with a spatial attribute <code>Restaurant</code> . (b) The relational schema adapted for the spatial attribute <code>Restaurant.location</code> . (c) A PRM-SA as proposed in Definition 23	108
8.2	The class dependency graph for the PRM-SA in Figure 8.1c	111
8.3	An example of a class dependency graph with a cycle	111
8.4	An example of a dependency structure that models the dependency of an attribute with the aggregated value of the same attribute of spatial objects in the same cluster.	112
8.5	(a) A PRM-SA with multiple spatial attributes. (b) The corresponding moralized graph used to identify the set of partition functions to be optimized.	115
8.6	PRMs-SA used in the experiments as gold standard models.	124
8.7	Comparison of overall performance of PRM-SA learning algorithms with Nemenyi test	129
8.8	Comparison of overall hard $\text{precision}_{\text{spatial}}$, soft $\text{precision}_{\text{spatial}}$ and soft $\text{recall}_{\text{spatial}}$ of PRM-SA learning algorithms with Nemenyi test	130
8.9	Result of Nemenyi test for comparison of overall performance of PRM-SA learning algorithms in terms of the difference between the score of learned models and that of gold models.	130
9.1	Technological stack diagram	139
9.2	Class diagram showing how <code>RelationalSchema</code> , <code>Class</code> , <code>Attribute</code> , and <code>Domain</code> are related to each other	141
9.3	Class diagram showing how <code>RBN</code> is related to other classes	142
9.4	Types of <code>RBNDistribution</code>	143
9.5	Class diagram of <code>RGS</code>	146
9.6	Types of <code>GraphOperation</code> available in <code>PILGRIM-Relational</code>	146
9.7	Class diagram of relational skeleton generation strategies implemented in <code>PILGRIM</code>	147
9.8	Class diagram of sampling strategies implemented in <code>PILGRIM</code>	148
9.9	Class diagram of <code>RecoModel</code>	149
9.10	Class diagram of <code>NaiveBayesianClassifier</code>	150
A.1	(a) The PRM used in the experiments, and (b) the underlying relational schema as a DAG	158
A.2	Distribution of objects in the relational skeletons used in the experiments	159
A.3	Max in-degree of entity objects in the relational skeletons.	160
A.4	Distribution of in-degree in naïve and k -partite skeletons	160
A.5	Burn-in vs time taken by RBG sampling algorithm on naïve and k -partite graph-based skeletons of different size.	161

A.6	Skeleton size vs time taken by RBG sampling algorithm for different values of burn-in.	162
A.7	Skeleton size vs number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test for different values of burn-in. Lower values are better here.	163
A.8	Burn-in vs number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test on (a) k -partite graph-based skeletons, and (b) naïve skeletons of different size. Lower values are better here.	164
A.9	Time taken by relational forward sampling, RBG sampling, and GBN-based sampling algorithms on naïve and k -partite graph-based skeletons of different size.	164
A.10	Number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test on naïve and k -partite graph-based skeletons of different size. Lower values are better here.	165
C.1	Comparison of PRM-SA learning algorithms with Nemenyi test for Model A1	191
C.2	Comparison of PRM-SA learning algorithms with Nemenyi test for Model A2	191
C.3	Comparison of PRM-SA learning algorithms with Nemenyi test for Model C2	192
C.4	Comparison of PRM-SA learning algorithms with Nemenyi test for Model D1	193

List of Algorithms

1	Generate_Neighbors (Bayesian Networks)	20
2	Lazy Aggregation Block Gibbs (LABG)	25
3	Relational Greedy Search	26
4	Generate_Neighbors (PRM)	27
5	Generate_Random_PRM-DB (Ben Ishak [2015])	32
6	Generate_Relational_Skeleton	34
7	Generate_SubSkeleton	36
8	Relational forward sampling	39
9	Relational Block Gibbs sampling (based on Kaelin [2011]’s LABG)	40
10	Generate_Neighbors (Naïve Approach)	113
11	Increase_k Operation	113
12	Decrease_k Operation	114
13	Find_Structure_With_Best_k	116
14	Adaptative_Structure_Learning (Version 1)	116
15	Adaptative_Structure_Learning (Version 2)	117
16	Generate_Neighbors (Adaptative Structure Learning Version 3)	118
17	Generate_PRM-SA_Benchmark	119
18	Generate_Spatial_Schema	120
19	Generate_Spatial_Relational_Dataset	121
20	Generate_Spatial_Relational_Skeleton	121

Abbreviations

- AHP** Analytical Hierarchical Process. 81, 83, 95
- AIC** Akaike Information Criterion. 19, 136
- BD** Bayesian Dirichlet. 19
- BIC** Bayesian Information Criterion. 19, 136
- BN** Bayesian Network. 3, 12, 13, 16, 18–20, 24, 26, 28, 38, 66–68, 80, 81, 83, 100, 101, 106, 121, 154
- CARS** Context-aware Recommender System. 44, 45
- CDG** Class Dependency Graph. 22
- CF** Collaborative Filtering. 45, 46, 57, 65, 67–69, 80, 83
- CI** Conditional Independence. 19, 20
- CPD** Conditional Probability Distribution. 16, 18, 21, 24, 25, 30–32, 39, 92–94, 98, 139, 141, 145, 147, 167, 174
- CRP** Chinese Restaurant Process. 33, 35
- DAG** Directed Acyclic Graph. 3, 19, 29–31, 33–36, 118, 124, 125, 147, 157–160, 169
- DAPER** Directed Acyclic Probabilistic Entity Relationship. 30, 117, 124, 133, 154
- DF** Decision Factor. 92
- DFS** Depth-First Search. 34–36
- DMMHC** Dynamic Max-Min Hill Climbing. 136
- DMMPC** Dynamic Max-Min Parents and Children. 136
- DSS** Decision Support System. 42
- DTL** Database Template Library. 137
- DUKe** Data, User and KnowledgE. 136
- EAP** Expectation a Posteriori. 19, 143
- ERD** Entity-Relationship Diagram. 15
- GBN** Ground Bayesian Network. 21, 24, 25, 30–32, 35, 37, 39, 40, 94, 109, 110, 118, 121, 122, 128, 137, 139, 140, 143, 148, 153, 154, 157, 162–165, 175
- GIS** Geographic Information System. 80
- GPS** Global Positioning System. 72
- HCI** Human-Computer Interaction. 50
- hPRM** Hierarchical Probabilistic Relational Model. 24, 64, 67, 69

- IDG** Instance Dependency Graph. 22, 32
- IR** Information Retrieval. 42, 52, 54
- KNN** K-Nearest Neighbors. 81, 83
- LABG** Lazy Aggregation Block Gibbs. 25, 32, 39, 40, 118, 122, 153, 154
- LINA** Laboratoire d’Informatique de Nantes Atlantique. 136
- LSI** Latent Semantic Indexing. 57
- MAE** Mean Absolute Error. 53, 67–69
- MAP** Maximum a Posteriori. 16, 19, 143
- MAUT** Multi-Attribute Utility Theory. 50
- MBR** Minimum Bounding Rectangle. 76
- MCDM** Multi-criteria Decision Making. 50, 94
- MDL** Minimum Description Length. 19, 136
- ML** Machine Learning. 42
- MLE** Maximum Likelihood Estimation. 19, 25, 143
- MMHC** Max-Min Hill Climbing. 20, 26, 136
- MML** Minimum Message Length. 19
- MMPC** Max-Min Parents and Children. 20, 136
- NMAE** Normalized Mean Absolute Error. 53
- NMI** Normalized Mutual Information. 126, 130, 183
- OGC** Open Geospatial Consortium. 76
- OBN** Object-Oriented Bayesian Network. 3, 80
- PCA** Principle Component Analysis. 57
- PGM** Probabilistic Graphical Model. 2, 3
- POI** Point-of-interest. 56, 81
- PPR** Personalized PageRank. 82
- PRM** Probabilistic Relational Model. 3–8, 12, 13, 20, 24, 28, 62–69, 89, 90, 92–94, 104, 106, 111, 112, 117, 125, 132, 133, 136, 137, 139, 141, 143–145, 148, 151–154, 157, 158, 167, 170, 171, 174–178
- PRM-CH** Probabilistic Relational Model with Class Hierarchy. 24, 64, 67
- PRM-CU** Probabilistic Relational Model with Clustering Uncertainty. 24, 136, 138, 141, 144, 172
- PRM-EU** Probabilistic Relational Model with Existence Uncertainty. 3, 24, 63, 69, 89–91
- PRM-PrefReco** Probabilistic Relational Model for Preference-based Recommenders. 90, 92–94, 96, 98–102, 151
- PRM-RU** Probabilistic Relational Model with Reference Uncertainty. 24, 26, 136, 138, 141, 144, 172

- PRM-SA** Probabilistic Relational Model with Spatial Attributes. 5–8, 106, 123, 124, 127–130, 132, 133, 136–138, 143–145, 148, 152–154, 167, 170, 172, 174, 175, 177, 178, 183, 191–193
- RBG** Relational Block Gibbs. 38, 39, 122, 123, 138, 148, 153, 157, 158, 161–165
- RBN** Relational Bayesian Network. 3, 29, 117
- RDN** Relational Dependency Network. 3
- RGS** Relational Greedy Search. 26, 124, 132, 133, 143, 145, 153
- RMMHC** Relational Max-Min Hill Climbing. 26, 31, 133, 138, 143, 153
- RMMPC** Relational Max-Min Parents and Children. 133, 153
- RMN** Relational Markov Network. 3, 112
- RMSE** Root Mean Squared Error. 53
- RS** Recommender System. 3, 42–45, 48, 51, 52, 54, 58, 59, 62, 88–91, 106
- RSHD** Relational Structural Hamming Distance. 30, 117, 124
- SRL** Statistical Relational Learning. 2–4, 12, 13, 153
- ST-DBN** State-and-transition Dynamic Bayesian Network. 80
- SVD** Singular Value Decomposition. 57
- UGCF-PRM** User Grade-based Collaborative Filtering - PRM. 65, 67
- VGI** Volunteered Graphical Information. 72
- WSM** Weighted Sum Method. 94, 102, 103
- XML** EXtensible Markup Language. 176

Special Terms

k -partite graph A graph whose vertices can be partitioned into k disjoint sets so that there is no edge between any two vertices within the same set. [33](#), [118](#), [123](#), [153](#), [157](#), [177](#)

Markov blanket The Markov blanket of a node in a Bayesian network is the set of parents, children, and spouses of the node. [63](#), [69](#), [125](#)

Moral Graph A moral graph is the equivalent undirected graph of a directed acyclic graph and is obtained by adding an edge between the nodes that have a common child and changing directed edges to undirected ones. [114](#)

Skeleton of a DAG The skeleton of a directed acyclic graph is the undirected graph that results from ignoring the directionality of every edge. [125](#)

V-structure A v-structure in a directed acyclic graph G is an ordered triplet of nodes, (x, y, z) , such that G contains the arcs $x \rightarrow y$ and $y \leftarrow z$, and the nodes x and z are not adjacent in G . [125](#)



Introduction

Contents

1.1	Context	2
1.2	Motivation and problem statement	4
1.3	Contributions	6
1.4	Organization of the dissertation	7

1.1 Context

The increase in Internet users has become a global phenomenon. Almost all countries in the world have witnessed the incredible growth of Internet users in the past two decades¹. These users not only consume the data available in the Internet but also produce a huge amount of data through various activities, such as browsing, searching, providing personal information, blogging, sharing digital items (e.g., photos, videos etc.), volunteering (e.g., Wikipedia, OpenStreetMap), participating in surveys etc. Along with Internet users, the number of websites on the world wide web is also growing drastically². Simultaneously, technologies for collecting, disseminating, processing, and analyzing data have been improving at a fast pace. The combined effect of all these is that a tremendous volume of data is available day by day. The source of data is not limited to the Internet only. Various domains, such as environmental, biological, economical studies, and many applications contribute to the ever increasing data. With this abundance of data, extracting knowledge or useful information from data has become a widely researched topic.

Data analysis and knowledge discovery involve methods at the intersection of *machine learning*, statistics and databases. Machine learning is concerned with the extraction of knowledge in the form of functions, mathematical or statistical models that automatically evolve with experience. It is largely dominated by *inductive learning* (also known as *learning from examples* or *learning by induction*) techniques, which aim at inducing patterns (or hypotheses) from the given input data for predicting new data (*predictive modeling*) or for describing the input data (*descriptive modeling*). Traditional machine learning techniques are designed to work with *attribute-value representation* of data (also known as *single-table single-tuple* (Raedt [2008]) format or *propositional data*), where each row represents a data instance, and each column represents an attribute. These techniques also assume that the data instances are *independent and identically distributed* (IID), i.e., the data instances are drawn independently from each other from an identical distribution. Examples of such dataset include well-known Iris dataset³, mushrooms dataset⁴ etc., which have been used in many studies. However, in general, real-world data do not come in single-table format. It has been a common practice to conceptualize a real-world system in terms of objects and relationships between those objects. That means real-world data usually come as *multi-table multiple-tuple* (Raedt [2008]) data, also called *relational data*. This kind of data consists of multiple tables, each corresponding to an *entity* type or a *relationship* type. Each row in an entity table represents an entity/object, and each row in a relationship table denotes the relationship between objects. As objects are related to each other, the IID assumption is often violated in relational data. This limits the application of traditional machine learning algorithms on relational data. Consequently, *Statistical Relational Learning (SRL)* has been emerged as a branch of machine learning that is concerned with statistical analysis on domains with complex relations and uncertainty (Getoor and Taskar [2007]).

SRL aims at learning statistical models exploiting relational information present in the data. It is primarily dominated by methods that are based on *Probabilistic Graphical Models (PGMs)*. PGMs use graph theory for expressing complex probabilistic

1. <http://data.worldbank.org/indicator/IT.NET.USER.P2>

2. <http://www.internetlivestats.com/total-number-of-websites/>

3. <https://archive.ics.uci.edu/ml/datasets/Iris>

4. <http://www.cs.toronto.edu/~delve/data/mushrooms/desc.html>

dependencies between random variables (Wainwright and Jordan [2008]). Directed as well as undirected versions of PGMs have been extended for relational settings. *Probabilistic Relational Models (PRMs)* (Friedman et al. [1999], Getoor [2001]), *Relational Bayesian Networks (RBNs)* (Jaeger [1997]), and *Object-Oriented Bayesian Network (OOBN)* (Koller and Pfeffer [1997], Bangsø and Wuillemin [2000]) are relational extensions of *Bayesian Networks (BNs)*, which are PGMs that use *Directed Acyclic Graphs (DAGs)*. Markov networks, and dependency networks, which are respectively undirected and bi-directed counterparts of BNs, have been extended by *Relational Markov Networks (RMNs)* (Taskar et al. [2002]), and *Relational Dependency Networks (RDNs)* (Neville and Jensen [2007]) respectively with the concept of objects to deal with relational data. This thesis is concerned with the directed version, particularly PRMs. A PRM models the uncertainty over the attributes of objects in the domain and the uncertainty over the relations between objects (Friedman et al. [1999]).

SRL has found its application in special tasks that originate from the relational nature of data. Most popular among these tasks are *link prediction*, *entity resolution*, *collective classification*, and *link-based clustering*. The central idea to all of them is to exploit links (or relationships) between objects for problem solving. Links can be, for example, the action of rating/buying an item in an e-commerce website, friendships in social networks, the action of reading news articles/listening to music, co-author links between author references in bibliographical data, links between spatial references in geo-spatial data and so on. The objective of link prediction is to determine whether a link or relationship exists between objects. Entity resolution aims at identifying references that denote the same entity. Practical uses of entity resolution include finding duplicates in data, integrating data from multiple sources, disambiguating user queries etc. The goal of collective classification is to classify entities given their attributes and links in presence of autocorrelation. Link-based clustering groups together objects based on the similarity of their attributes as well as the attributes of linked objects.

In this thesis, we are particularly interested in the task of link prediction. One of the applications where link prediction is very useful is *Recommender Systems (RSs)*, which is also a very common source of relational data. Recommender systems are basically built around the interaction between users and items, and help discover interesting items for users from a big collection of items. Such systems can be observed in many different websites that we use everyday (e.g., Amazon, YouTube, IMDb, Facebook etc.). In general, users are described by some properties such as their demographics information. Items are domain-specific objects that users are looking for, such as music, movies, books, news, cities, restaurants, friends, products etc. Depending on the domain, users might interact with items in different ways. For example, in Amazon, users purchase items, and provide feedback in the form of ratings, and comments; in social networking sites, users become friends. The goal of a recommender system is to predict the items that users might find interesting to interact with. In other words, recommender systems predict whether a link could exist between a user and an item. *Probabilistic Relational Model with Existence Uncertainty (PRM-EU)*, an extension of PRM for modeling the uncertainty over the existence of relationships between objects, can be suitable in such scenario.

As recommender systems have the potential to increase revenue generation, they are being applied in many different systems. Consequently, various recommendation techniques have been devised to fit in different scenarios. A growing trend in recommender systems is to improve recommendations by exploiting locational or spatial information about users and/or items. For example, YouTube recommends videos that are popular

in users' geographic region, LinkedIn suggests jobs in nearby areas etc. The presence of a spatial dimension enhances users' experience. At the same time, it restrains the application of conventional algorithms because of the geographical representation of objects and the interactions it adds between objects. Such interactions make multi-relational settings suitable for analyzing spatial data (Malerba [2008]). Though PRMs are interesting SRL models that can extract statistical patterns from relational data, they cannot treat spatial objects in the same way as non-spatial ones due to the presence of spatial information, which are commonly represented through their geometry. Malerba [2008] has mentioned the possibility of using PRMs with spatial relational databases. However, not much progress has been made in this direction.

This thesis deals with the (not much explored) intersection of three related fields – PRMs, spatial data, and recommender systems. The thesis makes two main contributions. Our first contribution is concerned with the intersection of PRMs and recommender systems. We propose a novel approach for using PRMs for making personalized recommendations in preference-based systems. Our second contribution addresses the problem of integrating spatial information into PRMs. In the following section, we will describe the motivating scenario for this thesis and state the problems we try to address.

1.2 Motivation and problem statement

As we have discussed in the previous section, PRMs can be used to realize recommender systems through the task of link prediction. Using PRMs in recommender systems has, in fact, been a topic of research from the beginning of PRM formalism. However, most of the studies (e.g., Getoor and Sahami [1999], Newton and Greiner [2004]) are focused on implementing collaborative filtering approach, which demands a good amount of historical data about users' interactions in the system. In the absence of such historical data, PRMs are used with users' demographics information or items' basic features (Gao et al. [2007], Huang et al. [2004]). At DataForPeople, a young startup based in Nantes, there was a need for a recommender system for recommending real estate properties, which are far less frequently purchased than books, movies or similar inexpensive items. In this system, called Kyzia⁵, users provide their preferences (or criteria for search) about various features of real estate properties. The goal is to recommend most relevant properties to the users from their search criteria. The system does not oblige users to provide their personal details because they usually have short-term preferences. So, users have very less interactions with this young system. Because of the lack of user profiles, and infrequent interactions of users, commonly used techniques, such as collaborative filtering or demographics-based filtering, are not appropriate for this system. Besides being short-term, an interesting but challenging property of users' preferences in our system is that users may have different flexibility towards those preferences. In other words, users generally have preferences not only about items' features but also about search criteria. For example, a user who does not have good income may be strict about his preferred price of properties but flexible about other features of properties whereas a user with children may not be willing to compromise the surface area of properties and so on. Such situations can arise not only in real estate search systems but also in other systems, such as flight/product search systems, hotel reservation system etc., where items are described by a set of features

5. www.kyzia.fr

and users state their preferences about those features to find their preferred items. For similar domains, [Viappiani et al. \[2007\]](#), and [Shearin and Lieberman \[2001\]](#) have proposed critiquing-based systems, where users are first presented with a small set of recommended items and the recommendations are iteratively improved after receiving feedback (or critiques) from the users. They take into account users' preferences about feature values but not about search criteria. Moreover, such iterative solutions demand users' patience to receive good recommendations. Therefore, the first contribution of this thesis deals with the problem of providing personalized recommendations, even without user profiles, to users from their preferences about items' features and search criteria for recommending less frequently purchased (or interacted) items. We propose a [PRM](#)-based solution for this problem. Our solution is generic and is not restricted to the domain of real estates.

In the context of real estate recommendation, adding a spatial dimension is always interesting because real estate properties always involve geographical information, and users tend to have preferences about location. For example, users may prefer to live not too far from some points of interest such as workplace, children's schools etc., and they may have different preferences for the geographical position and various other features of the property. Thus, adding a spatial dimension in a recommender system might improve the quality of recommendations. We identify two approaches to make use of spatial information in our recommendation system – (1) by integrating spatial information directly into our personalized recommendation model, and (2) by enhancing [PRMs](#) to support spatial objects and then use the enhanced [PRMs](#) instead of standard [PRMs](#) in our recommendation model. The second approach is more general, and can enable [PRMs](#) to be applied on spatial data not necessarily from recommender systems only. Therefore, the second contribution of this thesis addresses the problem of supporting spatial objects in [PRMs](#). Our decision for taking the second approach is also driven by the change in business model of DataForPeople. Our concentration was shifted from real-estate data to collaborative data collected from city-dwellers through our mobile application FixMaVille⁶ to help improve their cities. Therefore, we now have a different kind of spatial data. To analyze general spatial data, we propose to extend standard [PRMs](#) to support spatial objects. We formalize the new extension of [PRMs](#) as *Probabilistic Relational Models with Spatial Attributes (PRMs-SA)*, and propose more than one algorithm for learning the structure of such models from spatial data. To evaluate our proposed algorithms, we follow [Ben Ishak \[2015\]](#)'s methodology, which involves the comparison of the model learned from a synthetic data with the model from which the synthetic data is generated. [Ben Ishak \[2015\]](#) has proposed an algorithm for generating random datasets for benchmarking [PRM](#) learning algorithms. However, it only generates non-spatial datasets. Thus, we propose an algorithm for generating spatial datasets by sampling [PRMs-SA](#).

To summarize, following are the main research challenges that this thesis attempts to answer:

1. How to make personalized recommendations from users' short-term preferences about items when :
 - Items are less frequently purchased, i.e. low user-item interactions,
 - Users have preferences not only for items but also for items' various characteristics,

6. www.fixmaville.fr

- The system lacks user profiles, i.e. no possibility of making recommendations based on demographics information, and
 - The system is new with very few users?
2. How to integrate spatial information into a [PRM](#)? How to learn such probabilistic models from spatial data?
 3. How to generate spatial datasets randomly?

1.3 Contributions

This thesis has made the following contributions:

A [PRM](#)-based, personalized recommender system

We have proposed a novel approach to build a personalized [PRM](#)-based recommendation model taking into consideration users' preferences for items as well as for items' characteristics ([Chulyadyo and Leray \[2014\]](#)). Using our generic approach, content-based, collaborative filtering as well as hybrid models can be achieved from the same [PRM](#). Our preliminary experiment on a real-world dataset has shown that our model is actually capable of personalizing recommendations in cold-start situation. This work was presented at KES '2014, and will be explained in detail in Chapter 7.

[PRM](#) with spatial attributes ([PRM-SA](#))

The second contribution of this thesis is the formalism of [PRMs-SA](#), an extension of [PRMs](#) to add a support for spatial data. We have also proposed some algorithms to learn our model from data. The theoretical aspects of our model has been presented at DSAA '2016 ([Chulyadyo and Leray \[2015\]](#)). This thesis adds experiments to evaluate our proposed algorithms for learning the structure of a [PRM-SA](#). Chapter 8 will provide an in-depth elaboration of our model, and present experimental findings.

Improvement of [PRM](#) benchmark generation process

As algorithms for learning [PRM-SA](#) structure can be evaluated in the same way as those for [PRMs](#), we tried to follow [Ben Ishak \[2015\]](#)'s approach. However, we encountered some limitations in their approach, which prevented us from utilizing their algorithms for simulating spatial datasets to evaluate our [PRM-SA](#) learning algorithms. We have discussed these limitations in Section 2.7.3, and proposed some improvements, notably algorithms for generating relational skeletons and sampling [PRMs](#), in Section 2.7.4. Our proposed algorithm for relational skeleton generation has been illustrated in the technical report [Ben Ishak et al. \[2016\]](#). We extended the improved benchmark generation process to generate random spatial datasets, which are needed for evaluating our [PRM-SA](#) learning algorithms. This will be explained in Section 8.4.2.

Development of [PILGRIM](#) software

One of the main objectives of this thesis is also to contribute in the development of a software tool to work with probabilistic graphical models. Our research lab, LINA, has been actively developing a tool called [PILGRIM](#) (**P**robab**IL**istic **G**Raph**I**cal

Model). Originally developed for modeling, learning and reasoning upon Bayesian networks, this project was extended to support [PRMs](#). This thesis has made significant contributions in the implementation and improvement of core functionalities as well as the development of some modules of PILGRIM. Our proposed model, [PRM-SA](#), and algorithms for learning such model have also been implemented in PILGRIM. To demonstrate the usage of [PRM](#) in practical applications, [Huang et al. \[2004\]](#)'s recommendation model has been implemented. Chapter 9 will explain PILGRIM in detail. Appendix B is provided as a user guide to illustrate the usage of PILGRIM along with code snippets.

Other contributions

Our state-of-the-art about [PRM](#)-based recommender systems has been presented at [Ben Ishak et al. \[2014\]](#), and published in a technical report [Chulyadyo and Leray \[2013\]](#).

An empirical study about [PRM](#) sampling algorithms was also performed during this thesis. Details on this study and its findings are presented in Appendix A.

This thesis has also contributed to professional projects of DataForPeople, notably Kyzia and FixMaVille.

1.4 Organization of the dissertation

Chapter 2 will provide an introduction to relational data, and Bayesian networks. We will define basic concepts used in the representation of relational data with some examples. We will present a general overview of Bayesian networks, and standard approaches for learning such models. Then, we will define Probabilistic Relational Models ([PRMs](#)), and related concepts. We will also present methods for learning a [PRM](#) from data. We will then concentrate on evaluation of [PRM](#) learning algorithms, particularly on the state-of-the-art methodology proposed by [Ben Ishak \[2015\]](#). We will discuss on some limitations of their approach, and propose some improvements.

Chapter 3 will introduce recommenders systems, and will provide a review of standard recommendation techniques. We will then focus our discussion on the evaluation of recommender systems. We will also discuss on some major challenges imposed on recommender systems.

In Chapter 4, we will review some recommender systems that use [PRMs](#). Through this review, we show that [PRMs](#) have the potential to be used for recommendations.

Chapter 5 will give a brief introduction to spatial data. We will explain basic concepts related to spatial data, such as commonly used ways for representing spatial data, special characteristics of spatial data, and various operations that can be performed on this kind of data.

We will review some spatial recommender systems in Chapter 6 to show the trend of using spatial information in recommender systems.

Chapter 7 will present our first major contribution of this thesis. We will present our approach to build a personalized [PRM](#)-based recommender system. We will explain

our model in detail, and illustrate, through examples, that our approach is applicable in different domains. We will then present the findings of a preliminary experiment that we performed on our target application, which is a very new system with very few users.

Our second contribution, PRMs with Spatial Attributes ([PRMs-SA](#)), will be presented in Chapter 8. We will start with the definitions of [PRMs-SA](#), and related concepts. We will also propose four algorithms for learning the structure of a [PRM-SA](#). We will discuss on the methodology for evaluating those algorithms. We will then present the results of the experiments performed to evaluate the four algorithms for learning a [PRM-SA](#).

Chapter 9 will provide a detailed insight into PILGRIM. It consists of four sub-projects. But since this thesis has contributed in the sub-projects PILGRIM-Relational, and PILGRIM-Applications, we will discuss on the implementation of various modules of these two projects only.

In Chapter 10, we will present our conclusions, and point towards some prospects of future research in the area addressed by this thesis.

Appendix A will present the experiments we had performed to study three [PRM](#) sampling algorithms. The findings of this study were crucial for setting up the experiments of Section 8.5 for evaluating [PRM-SA](#) learning algorithms.

Appendix B has been provided as a user guide to serve as a quick guide for those who use our PILGRIM-Relational and PILGRIM-Applications libraries.

Appendix C will provide the detailed results of our experiment for evaluating [PRM-SA](#) structure learning algorithms presented in Chapter 8.



State-of-the-art



Probabilistic Relational Models for Relational Learning

Contents

2.1	Introduction	12
2.2	Background	13
2.2.1	Relational data representation	13
2.2.2	Bayesian Networks	16
2.3	Probabilistic Relational Models (PRMs)	20
2.4	Extensions	24
2.4.1	PRM with structural uncertainty	24
2.4.2	Other extensions	24
2.5	Inference in PRMs	24
2.6	Learning PRMs	25
2.6.1	Learning parameters	25
2.6.2	Learning structure	26
2.7	Evaluating PRM learning algorithms	28
2.7.1	Evaluation strategy and metrics	29
2.7.2	Generating PRM benchmarks	30
2.7.3	Limitations	31
2.7.4	Proposals for improvement	32
2.8	Conclusion	40

2.1 Introduction

Nowadays it is very common to represent a system in terms of relationships between objects that exist in the system. It is, in fact, an intuitive approach to conceptualize systems because we often experience such relationships in our daily life. For example, a person is related to another person in a family, a cat is related to a person because he owns the cat, a webpage is often linked to other webpages, a movie may be related to a book because it is based on that book, and so on. Such relationships between different types of objects, referred to as *relational data*, have been the subject of analysis in various domains, such as bioinformatics (Salwinski et al. [2004]), recommender systems (Ricci et al. [2011], Bobadilla et al. [2013], Huang et al. [2005]), communication analysis (Rossi and Neville [2010]), network analysis (Tang and Liu [2009], Chau and Chen [2008]), scientific citation (Martin et al. [2013], Shibata et al. [2012], Popescul and Ungar [2003]), natural language processing (Califf and Mooney [1999]) etc.

With the increased use and availability of relational data, the interest in extracting useful patterns from such data has been steadily growing in the machine learning community. In traditional machine learning settings, data is usually assumed to consist of a single type of object, and the objects in the data are assumed to be independent and identically distributed (IID). The IID assumption, however, is often violated in a relational setting because of the presence of relationships between objects. This limits the application of traditional machine learning algorithms on relational data. Consequently, Statistical Relational Learning (SRL) has been emerged as a branch of machine learning that is concerned with statistical analysis on domains with complex relations and uncertainty (Getoor and Taskar [2007]). SRL aims at learning statistical models exploiting relational information present in the data. Application of SRL methods can be found in the tasks of link prediction (Popescul and Ungar [2003], Huang et al. [2005]), entity resolution (Nickel et al. [2011], Bhattacharya and Getoor [2007]), collective classification (Neville and Jensen [2003], Macskassy and Provost [2007]), and link-based clustering in relational context.

In the past two decades, significant advances have been made in the area of SRL. A variety of SRL models has been proposed. Most of these models are based on *probabilistic graphical models (PGMs)*, a formalism for multivariate statistical modeling that uses graph theory for expressing complex probabilistic dependencies among random variables (Wainwright and Jordan [2008]). *Probabilistic relational models (PRMs)* (Friedman et al. [1999], Getoor et al. [2001]), and *relational Bayesian networks (RBNs)* (Jaeger [1997]) are some of the earliest approaches to SRL¹, and extend *Bayesian networks (BNs)*, a commonly used directed graphical model. Taskar et al. [2007] proposed *relational Markov networks (RMNs)* based on *Markov networks*, an undirected counterpart of BNs. Other important SRL models based on graphical models are *relational dependency networks* (Neville and Jensen [2003]), which are based on dependency networks, *Directed Acyclic Probabilistic Entity-Relation (DAPER)* models (Heckerman et al. [2004]), *Markov logic networks* (Richardson and Domingos [2006]), Bayesian Logic Programs (Kersting and De Raedt [2007]) etc.

In this thesis, we focus on PRMs only. Since this model is an extension of BNs for relational data, we will begin with a brief overview of relational data representation and BNs in the following section. We will then introduce PRMs in Section 2.3.

1. PRMs are also referred to as RBNs in recent articles (Neville and Jensen [2003]). However, in this thesis, we preserve the original context and use PRM to refer to the model proposed by Friedman et al. [1999] and Getoor et al. [2001].

2.2 Background

As mentioned in Section 2.1, several models exist for SRL. These models are often characterized by the representation of relational data and the tasks they address. As we are concerned with PRMs only, we will present a review of the relational data representation used by PRMs in Section 2.2.1, and the formalism of BNs, the basis of PRMs, in Section 2.2.2.

2.2.1 Relational data representation

In relational learning, relational data are often defined in a variety of ways. These definitions are mostly based on entries in relational databases (Getoor [2001], Heckerman et al. [2007], Rossi et al. [2012], Nickel et al. [2016]) or ground predicates in first-order logic (De Raedt and Kersting [2004], Richardson and Domingos [2006]). In this thesis, we adopt the former one, which is based on database theory (Codd [1970]).

The relational data model is an approach to representing (and managing) systems involving multiple types of entities interacting with each other. An *entity* corresponds to a thing or object that can be stored in a database. An interaction between the entities are referred to as a *relationship*. We call entities and relationships as *classes*. Properties of a class are described by *attributes*². Notations used in this dissertation to denote these and the following concepts are listed in Table 2.1.

Definition 1 *Relational schema*

A relational schema describes the entity and relationship classes, \mathcal{X} , and the attributes, \mathcal{A} , in a domain, and specifies the constraints over the number of objects involved in a relationship. ■

Each class $X \in \mathcal{X}$ is described by a set of *descriptive attributes* $\mathcal{A}(X)$ and a set of *reference slots* $\mathcal{R}(X)$.

Definition 2 *Reference slot and inverse slot*

A reference slot $X.\rho$ relates an object of class X to an object of class Y and has $\text{Domain}[\rho] = X$ and $\text{Range}[\rho] = Y$. The inverse of a reference slot ρ is called inverse slot and is denoted by ρ^{-1} . ■

In the context of relational databases, a class refers to a single database table, descriptive attributes refer to the standard attributes of tables, and reference slots are equivalent to foreign keys. While a reference slot gives a direct reference of an object with another, objects of one class can be related to objects of another class indirectly through other objects. Such relations are represented with the help of a *slot chain*.

Definition 3 *Slot chain*

A slot chain is a sequence of slots (reference slots and inverse slots) $\rho_1, \rho_2, \dots, \rho_n$ such that for all i , $\text{Range}[\rho_i] = \text{Domain}[\rho_{i+1}]$. ■

A slot chain can be single-valued or multi-valued. When it is multi-valued, we need a function to summarize them. We call such function an *aggregator*. Examples include mode, average, cardinality etc.

2. Throughout this dissertation, we assume that each class has an identifier attribute, which helps to identify *instances* (aka *objects*) of the class.

Table 2.1 – Notations and their meaning

Notation	Meaning
\mathcal{X}	The set of classes in a relational schema
X or X	A class in \mathcal{X}
\mathcal{A}	The set of attributes in a relational schema
$\mathcal{A}(X)$	The set of attributes of the class X
$X.A$ or $X.A$	Attribute A of the class X
$\mathcal{V}(X.A)$	The possible values (domain) of the attribute $X.A$
\mathcal{R}	The set of reference slots in a relational schema
ρ	A reference slot in \mathcal{R}
$\mathcal{R}(X)$	The set of reference slots in the class X . This set will be empty for entity classes.
γ	An aggregator
Π	A PRM
\mathcal{I}	An instance of a relational schema
σ_r	A relational skeleton
$\sigma_r(X)$	The set of objects in skeleton σ_r whose class is X
x	An object of class X
$x.A$	The attribute A of the object x whose class is X
$Pa(X.A)$	Parents of $X.A$
$Ch(X.A)$	Children of $X.A$

Definition 4 Aggregator

An aggregator γ is a function that takes a multi-set of values and produces a single value as a summary. ■

Sometimes, it may be interesting to aggregate the results of more than one slot chain, in which case we can apply *multi-set operators*. Union, intersection, difference etc. are some examples of multi-set operators.

Definition 5 Multi-set operator

A multi-set operator ϕ_k on k multi-valued attributes A_1, \dots, A_k that share the same range $\mathcal{V}(A_1)$ is a function from $\mathcal{V}(A_1)^k$ to $\mathcal{V}(A_1)$. ■

Instantiation of a relational schema

Here, we consider two types of instantiation of a relational schema – 1) a complete instantiation without any missing or unknown values, and 2) a partial instantiation of a relational schema where only objects and relations between the objects are specified but attribute values are not. The former is referred to as a *database instance* and the latter as a *relational skeleton*. Though it is possible to store both kind of instantiation physically into different types of databases, such as relational databases, graph

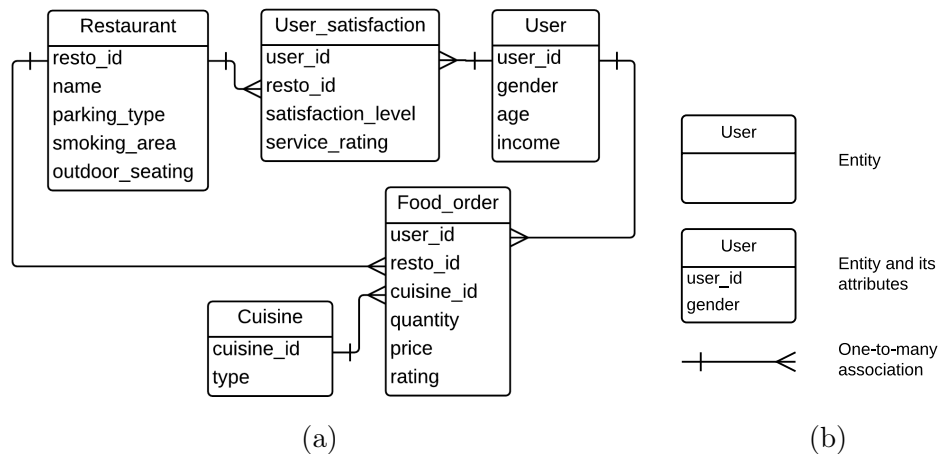


Figure 2.1 – (a) An example of a relational schema, (b) Crow's foot notations

databases etc., this thesis is concerned with relational databases only, where schemas are well-defined.

Definition 6 Instance of a relational schema (Getoor et al. [2007])

An instance \mathcal{I} of a schema specifies:

- for each class X , the set of objects in the class, $\mathcal{I}(X)$.
- a value for each attribute $x.A$ (in the appropriate domain) for each object x .
- a value y for each reference slot $x.\rho$, which is an object in the appropriate range type, i.e., $y \in \text{Range}[\rho]$. Conversely, $y.\rho^{-1} = x$ such that $x.\rho = y$. ■

Definition 7 Relational skeleton (Getoor et al. [2007])

A relational skeleton σ_r of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects $\sigma_r(X_i)$ for each class and the relations that hold between the objects. However, it leaves the values of the attributes unspecified. ■

A relational schema is usually depicted with an *Entity-Relationship Diagram (ERD)* (Chen [1976]). Several variants of ERD notations are available. Throughout this dissertation, we use “Crow's foot” notation (see Figure 2.1b) in the logical data model of relational schemas.

Example 2.1 Restaurant-User-Cuisine schema

To illustrate these concepts, we use a relational schema of a system where users order foods in restaurants and rate the service of the restaurants. The schema is shown in Figure 2.1a.

Here, *Restaurant*, *User* and *Cuisine* are entity classes, and *User_satisfaction* and *Food_order*, which represent the relationships *Restaurant–User* and *User–Cuisine* respectively, are relationship classes. The attributes *User_satisfaction.service_rating*, *User.age*, *User.gender* etc. are descriptive attributes whereas *User_satisfaction.user_id*, which refers to *User.user_id*, is a reference slot whose domain and range are the objects of the classes *User_satisfaction* and *User* respectively. *Restaurant* and *User_satisfaction* objects are directly linked through the reference slot *User_satisfaction.resto_id*. Note that $\text{Restaurant.resto_id}^{-1}$ is the inverse of *User_satisfaction.resto_id* and gives all *User_satisfaction* objects corresponding to *Restaurant* objects. *Restaurant* objects can also be indirectly related to *User* objects through slot chains. For example, the slot chain $\text{User_satisfaction.resto_id}^{-1}.\text{user_id}$ relates restaurants with all the users whose

satisfaction level and/or service rating about the restaurants are available. As there is a many-to-many relationship between the classes *Restaurant* and *User*, this slot chain may result into more than one users for a single restaurant. In such case, we need an aggregator (such as average) to summarize (or aggregate) the resulting set. For instance, $AVERAGE(User_satisfaction.resto_id^{-1}.user_id.age)$ gives the average age of the users who have rated the restaurants. Use of multi-set operators on slot chains can provide interesting results. Let's take an example of intersection operation between two slot chains $User_satisfaction.resto_id^{-1}.user_id$ and $Food_order.resto_id^{-1}.user_id$. This operation gives those users who have made orders in some restaurants and rated them too. To achieve the same result without the use of multi-set operators, we would need a long slot chain. ❖

Example 2.2 A relational skeleton for Restaurant-User-Cuisine schema

An example of a relational skeleton corresponding to this relational schema is shown in Figure 2.2. The skeleton has two users, three cuisine, and three restaurants. Here, the relationships between users, restaurants, and foods are specified but not the descriptive attributes. Corresponding to this skeleton, Figure 2.2c depicts a complete instantiation of the relational schema. ❖

2.2.2 Bayesian Networks

A Bayesian network (BN) (Pearl [1988]) is a directed acyclic graph where nodes correspond to random variables and arcs between nodes represent conditional dependencies; lack of an arc between nodes indicates that the variables are conditionally independent. A BN associates with each random variable X_i a conditional probability $P(X_i | Pa_i)$, where $Pa_i \in X$ is the set of variables that are called the parents of X_i . BNs achieve compact representation of probability distribution by exploiting conditional independence properties of random variables. Every node in a BN is conditionally independent of its non-descendants given its parent. This conditional independence assumption enables BNs to simplify the joint probability distribution given by the Chain rule as follows:

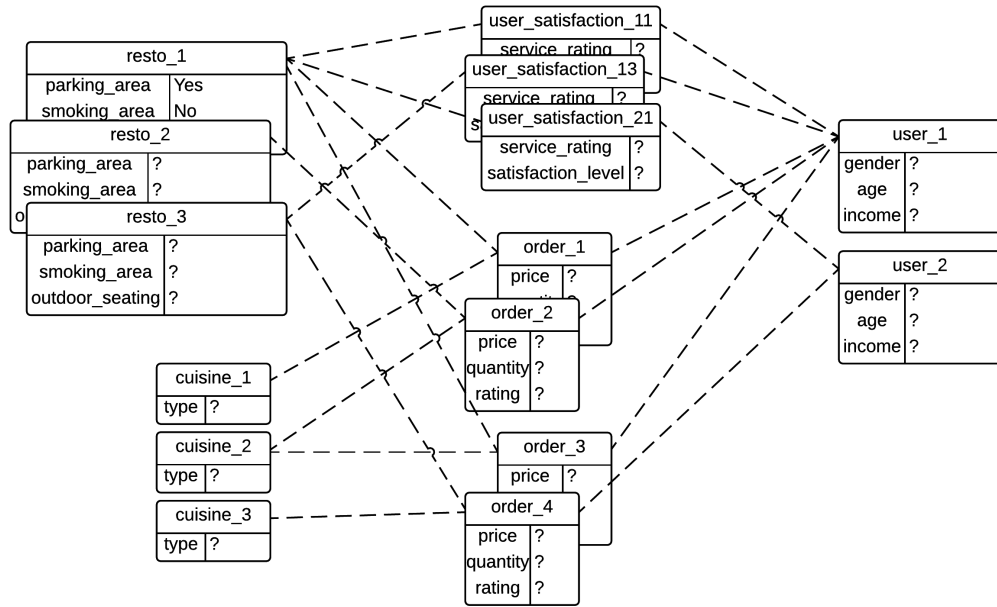
$$P(X_1, X_2, \dots, X_n) = \prod_i P(X_i | Pa_i) \quad (2.1)$$

Example 2.3 Burglary and Earthquake network from Pearl [1988]

Figure 2.3 shows an example of a Bayesian Network adapted from Pearl [1988]. All random variables in this example take binary states. Each node is associated with a *Conditional Probability Distribution (CPD)*. The edge between the nodes *Earthquake* and *Alarm* indicates that earthquakes can make the alarm go off. Similarly, if there is an earthquake, it is likely that there will be an announcement on the radio. ❖

Inference in Bayesian networks

Inference in Bayesian networks generally refers to: finding the probability of a variable being in a certain state, given that some of the other variables are set to certain values; or finding the values of a given set of variables that best explain (in the sense of the highest *Maximum a Posteriori (MAP)* probability) why a set of other variables are set to certain values (Daly et al. [2011]). A plethora of inference algorithms can



(a)

Restaurant					Food_order					
resto_id	name	parking_type	smoking_area	outdoor_seating	user_id	resto_id	cuisine_id	quantity	price	rating
resto_1	ABC	?	?	?	user_1	resto_1	cuisine_1	?	?	?
resto_2	DEF	?	?	?	user_1	resto_2	cuisine_2	?	?	?
resto_3	XYZ	?	?	?	user_1	resto_1	cuisine_2	?	?	?
					user_2	resto_3	cuisine_3	?	?	?

User				Cuisine		User_satisfaction			
user_id	gender	age	income	cuisine_id	type	resto_id	user_id	service_rating	satisfaction_level
user_1	?	?	?	cuisine_1	?	resto_1	user_1	?	?
user_2	?	?	?	cuisine_2	?	resto_1	user_2	?	?
user_3	?	?	?	cuisine_3	?	resto_3	user_1	?	?

(b)

Restaurant					Food_order					
resto_id	name	parking_type	smoking_area	outdoor_seating	user_id	resto_id	cuisine_id	quantity	price	rating
resto_1	ABC	Free	Yes	Yes	user_1	resto_1	cuisine_1	A	H	2
resto_2	DEF	Free	No	Yes	user_1	resto_2	cuisine_2	B	L	3
resto_3	XYZ	Paid	No	No	user_1	resto_1	cuisine_2	A	L	2
					user_2	resto_3	cuisine_3	A	L	1

User				Cuisine		User_satisfaction			
user_id	gender	age	income	cuisine_id	type	resto_id	user_id	service_rating	satisfaction_level
user_1	Male	20-30	Low	cuisine_1	A	resto_1	user_1	1	2
user_2	Female	45-60	High	cuisine_2	B	resto_1	user_2	2	2
user_3	Male	30-45	High	cuisine_3	A	resto_3	user_1	3	2

(c)

Figure 2.2 – Instantiations of the relational schema of Figure 2.1a. (a) depicts a relational skeleton as objects and relations between the objects. (b) presents the same skeleton in a relational database. (c) A complete instantiation of the relational schema of Figure 2.1a stored as a relational database

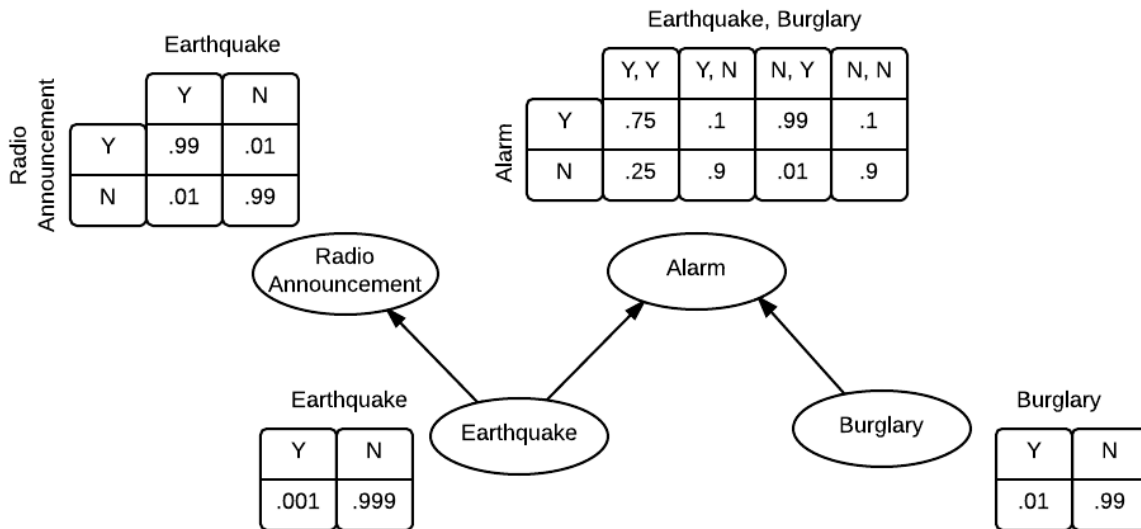


Figure 2.3 – An example of a Bayesian network (Pearl [1988])

be found in the literature. Pearl [1982]’s message passing algorithm is the first one for performing inference on BNs, and is a basis for many other algorithms (Shafer and Shenoy [1990]). It calculates marginal distribution for each unobserved node conditioned to any observed nodes. Originally formulated for trees, this algorithm was later extended to perform inference on polytrees (Kim and Pearl [1983]). These algorithms are applicable to singly-connected networks only, i.e. the networks that contain only a single path between nodes. Several algorithms exist for probabilistic inference on multiply connected networks (Lauritzen and Spiegelhalter [1988], Jensen et al. [1990], Shachter [1986], Sang et al. [2005]). These *exact inference* algorithms, which analytically compute the CPD over the variables of interest, are computationally expensive on large, complex BNs. As a solution, several *approximate inference* algorithms have been proposed. These algorithms involve heuristic and stochastic techniques which are not guaranteed to give the correct answer for a given query, but often return values that are close to the true values (Russell and Norvig [2003]). Henrion [1988]’s probabilistic logic sampling based on Monte Carlo methods, Jensen et al. [1995]’s block Gibbs sampling, and Weiss [1997]’s loopy belief propagation are some examples of approximate inference algorithms. In general, BN inference is found to be NP-hard in both the exact (Cooper [1990]) and approximate (Dagum and Luby [1993]) case.

Learning Bayesian networks

One way to construct a BN is to collect expert knowledge and create the model based on it. However, this is not feasible all the time as it may be difficult to find experts in the domain of interest. Also, data in the hand may not always be in accordance with experts’ opinion. Besides, it is very important to be able to construct a BN from data in today’s world, where data are constantly evolving and steadily being modified because such ever-changing data may show different behaviors at different time and hence the model will need to be updated with the changing data. Many algorithms have been developed in order to learn BNs from observed data. Learning a BN involves two tasks: (a) learning parameters (i.e. conditional probabilities), and (b) learning structure (i.e. the DAG structure).

Parameter learning Various statistical and Bayesian methods are available for learning conditional probabilities in BNs from different kinds of data. *Maximum Likelihood Estimation (MLE)*, Expectation Maximization with MLE, MAP and *Expectation a Posteriori (EAP)* are some Bayesian approach to parameter learning.

Structure learning Learning a BN structure is the task of finding a DAG structure that best represents the probabilistic dependencies existing in the given data. Exhaustive search to find the exact structure of a BN is impossible because the number of possible DAG grows exponentially with the number of nodes. Even for 10 nodes, there are 4.2×10^{18} possible DAGs. Many techniques have been proposed to learn BN structure. Those techniques can be broadly classified into three families – constraint-based approach, score and search approach, and hybrid approach.

Constraint-based approach In constraint-based approach, a BN is seen as an independence model and its structure is learned by testing *Conditional Independence (CI)* between the variables. Verma and Pearl [1991]’s Inductive Causation (IC), Spirtes et al. [2000]’s SGS, PC and PC* algorithms fall under this family. They use statistical tests, such as χ^2 and G tests for triples (X, Y, S) , where X and Y are variables and S is a subset of variables, to determine whether X and Y are conditionally independent given S , i.e. $X \perp Y \mid S$.

Score-and-search approach Also known as *score-based methods*, this approach looks for the DAG space in order to maximize a scoring/fitness function that assigns a score to a state in the search space to see how good a match is made with the sample data. Score-based algorithms begin with a search space of candidate BN structures, score each of them, and select the one with the best score. Because the size of the search space grows exponentially with the number of nodes, heuristics are generally applied to explore the search space. Greedy search (GS) is a commonly used heuristic search method, which performs in the following way: starting from a BN structure (which can be empty or not), GS moves to the best-scoring neighbor graph until convergence. GS is said to have converged to the solution when no neighbor graph has better score than the current graph. Intuitively, a graph and its neighbors are almost identical except for small local modifications. Such small local modifications are commonly obtained by adding an edge, deleting an edge or reverting an edge in the graph (Algorithm 1). A score-and-search method requires a scoring criterion that gives a good score when a structure matches the data well. Several scoring functions (such as *Bayesian Dirichlet (BD)* (Cooper and Herskovits [1992]), *Bayesian Information Criterion (BIC)* (Schwarz [1978]), *Akaike Information Criterion (AIC)* (Akaike [1970]), *Minimum Description Length (MDL)* (Bouckaert [1993]) and *Minimum Message Length (MML)* (Wallace et al. [1996]) etc.) have been devised for this purpose. Most scoring functions reward a better match of the data to the structure and prefer simpler structures. Another desirable property of a scoring function is decomposability, whereby the score of a particular structure can be obtained from the score for each node given its parents.

Hybrid approach The main limitation of score-and-search approach is scalability. For large number of variables, the search space is extremely big, and hence, a good amount of time is needed for examining candidate structures. On the other hand, constraint-based methods offer a fast approach even when there are large number of variables. The outcome of these methods, however, can be adversely affected

Algorithm 1 Generate_Neighbors (Bayesian Networks)**Input:** A DAG, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ **Output:** Neighbors of G , \mathcal{N}

```

1:  $\mathcal{N} \leftarrow \{\}$ 
2: for each  $n \in \mathcal{V}$  do
3:   for each  $n' \in \mathcal{V} \setminus n$  do
4:     if  $(n, n') \notin \mathcal{E}$  then ▷ If the edge  $(n, n')$  does not exist in  $\mathcal{G}$ , add it
5:        $\mathcal{G}' \leftarrow (\mathcal{V}, \mathcal{E} \cup \{(n, n')\})$ 
6:       if  $\mathcal{G}'$  has no cycle then
7:          $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{G}'$  ▷ Add_edge( $\mathcal{G}, n, n'$ )
8:       end if
9:     else ▷ If the edge  $(n, n')$  exists in  $\mathcal{G}$ , remove it or revert it
10:       $\mathcal{G}' \leftarrow (\mathcal{V}, \mathcal{E} \setminus \{(n, n')\} \cup \{(n, n')\})$  ▷ Revert_edge( $\mathcal{G}, n, n'$ )
11:      if  $\mathcal{G}'$  has no cycle then
12:         $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{G}'$ 
13:      end if
14:       $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathcal{V}, \mathcal{E} \setminus \{(n, n')\})$  ▷ Delete_edge( $\mathcal{G}, n, n'$ )
15:    end if
16:  end for
17: end for

```

by hidden variables and/or weak **CI** tests with larger conditioning sets. Hybrid methods exploit positive aspects of constraint-based and score-based methods by combining local search with global learning procedure. These methods typically search one local neighborhood for each node and learn global structure using this local information. Tsamardinos et al. [2006]’s *Max-Min Hill Climbing (MMHC)* algorithm is an example of hybrid structure learning method that uses *Max-Min Parents and Children (MMPC)* (Aliferis and Tsamardinos [2002]) for local information along with greedy search for global structure search.

2.3 Probabilistic Relational Models (PRMs)

Bayesian networks have been one of the main models for reasoning under uncertainty. The simplicity of their specification is one of the reasons for their success. However, one of the difficulties in Bayesian networks is to create and maintain the model of very large domains, which are usually conceived with relational settings. **BNs** are not sufficient to model this construct as they lack the concept of objects and their relations. They are designed for modeling attribute-based domains, where we have a single table of independent and identically distributed instances (Getoor et al. [2001]). They require a propositional data set whereas real world data are often stored and managed using relational representation. Converting relational data into flat data representation for statistical learning may introduce statistical skew and lose useful information that might help us understand the data. Thus, in order to learn a statistical model from relational data, *Probabilistic Relational Models (PRMs)* were emerged which specify a probability model for classes of objects rather than simple attributes. A **PRM** models the uncertainty over the attributes of objects in the domain and the uncertainty over the relations between the objects (Friedman et al. [1999]). It basically defines a template for probabilistic dependencies in typed relational domains which can be later

instantiated with a particular set of objects and relations between them to obtain a Bayesian network.

A PRM defines a *probabilistic model* for a *relational schema* of the domain. The probabilistic model of a PRM is specified for classes of objects, and represents generic probabilistic dependencies between the attributes of classes in the relational schema. The dependencies can be between the attributes of the same class or between the attributes of difference classes. Like in Bayesian networks, the dependency structure is associated with the conditional probability distribution of each node (attribute) given its parents.

Definition 8 Probabilistic Relational Model (PRM)

A PRM $\Pi = (\mathcal{S}, \Theta)$ for a relational schema \mathcal{R} is composed of a dependency structure \mathcal{S} , and a set of parameters Θ (Getoor [2001]). The dependency structure \mathcal{S} consists of a set of random variables and a set of probabilistic dependencies among the random variables. Each random variable $X.A$ in \mathcal{S} is a descriptive attribute $A \in \mathcal{A}(X)$ of a class $X \in \mathcal{X}$, and has a set of parents $Pa(X.A) = \{U_1, \dots, U_l\}$, which describes probabilistic dependencies. Each U_i has the form $X.B$ or $\gamma(X.\tau.B)$, where B is an attribute of any class, τ is a slot chain and γ is an aggregator of $X.\tau.B$. Finally, the parameters Θ is a set of conditional probability distributions (CPDs), representing $P(X.A | Pa(X.A))$. ■

Figure 2.4 depicts a PRM that corresponds to the relational schema in Figure 2.1a. The dashed lines here indicate that the classes are linked through reference slots.

PRMs define a distribution over instantiations of the database that are consistent with the relational skeleton. Instantiating a PRM for a relational skeleton results in a Bayesian network, also known as a *Ground Bayesian Network (GBN)*. The process of generating a GBN involves copying the associated PRM for every object in skeleton σ_r . Thus, a GBN will have a node for every attribute of every object in σ_r and probabilistic dependencies and CPDs as defined in the PRM.

Definition 9 Ground Bayesian Network (GBN)

A ground Bayesian network (GBN) defined for a PRM Π and a relational skeleton σ_r is as follows Getoor et al. [2007]:

- There is a node $x.A$ for every attribute of every object $x \in \sigma_r(X)$.
- Each $x.A$ depends probabilistically on parents of the form $x.B$ or $x.K.B$. If K is not single-valued, then the parent is the aggregate computed from the set of random variables $\{y | y \in x.K\}, \gamma(x.K.B)$.
- The CPD for $x.A$ is $P(X.A | Pa(X.A))$. ■

Figure 2.5 shows an example of a GBN (structure only), which is obtained by instantiating the PRM of Figure 2.4 for the relational skeleton of Figure 2.2.

The joint distribution over the instantiations of a PRM, Π , for a relational skeleton, σ_r is very similar to the chain rule for standard Bayesian networks.

$$P(\mathcal{I} | \sigma_r, \Pi) = \prod_{X \in \mathcal{X}} \prod_{A \in \mathcal{A}(X)} \prod_{x \in \sigma_r(X)} P(x.A | Pa(x.A)) \quad (2.2)$$

Here, we need to ensure that the probability distributions are coherent, i.e. the sum of probability of all instances is 1. In Bayesian networks, this requirement is

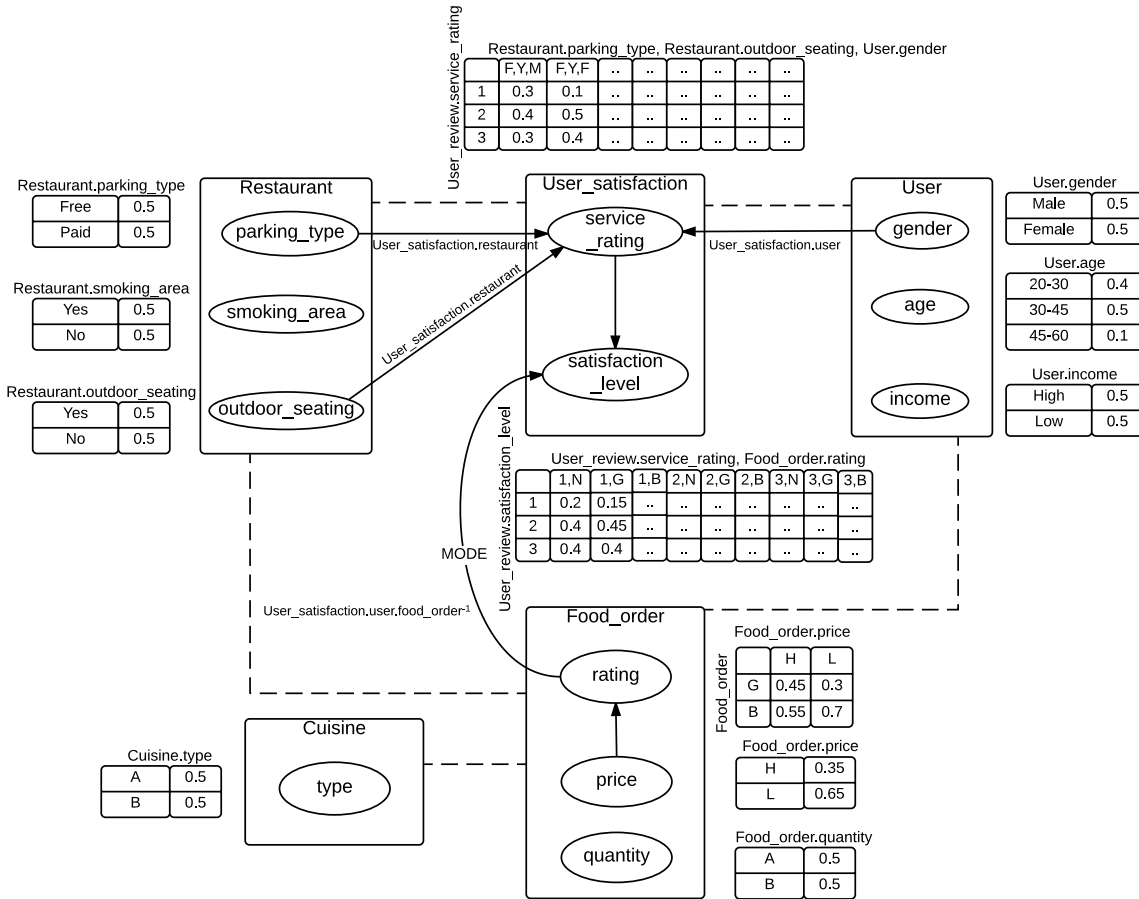


Figure 2.4 – An example of a PRM corresponding to the relational schema of Figure 2.1a

satisfied if the dependency graph is acyclic (Getoor et al. [2007]). To check whether the dependency structure \mathcal{S} of a PRM is acyclic relative to a given relational skeleton, we can inspect the graph of dependencies among attributes of objects in the skeleton. Such graph is termed as an *Instance Dependency Graph (IDG)*.

Definition 10 Instance dependency graph (IDG) Getoor et al. [2007]

The instance dependency graph G_{σ_r} for a PRM Π and a relational skeleton σ_r has a node for each descriptive attribute of each object $x \in \sigma_r(X)$ in each class $X \in X$. Each $x.A$ has the following edges:

1. Type I edges: For each formal parent of $x.A$, $X.B$, we introduce an edge from $x.B$ to $x.A$.
2. Type II edges: For each formal parent $X.K.B$, and for each $y \in x.K$, we define an edge from $y.B$ to $x.A$. ■

The dependency structure for a PRM is guaranteed to be acyclic for the given relational skeleton if the corresponding instance dependency graph is acyclic. However, if we want to check whether the dependency structure is acyclic for *any* relational skeleton, we can examine *Class Dependency Graph (CDG)*.

Definition 11 Class dependency graph (CDG) Getoor et al. [2007]

The class dependency graph G_{Π} for a PRM Π has a node for each descriptive attribute $X.A$, and the following edges:

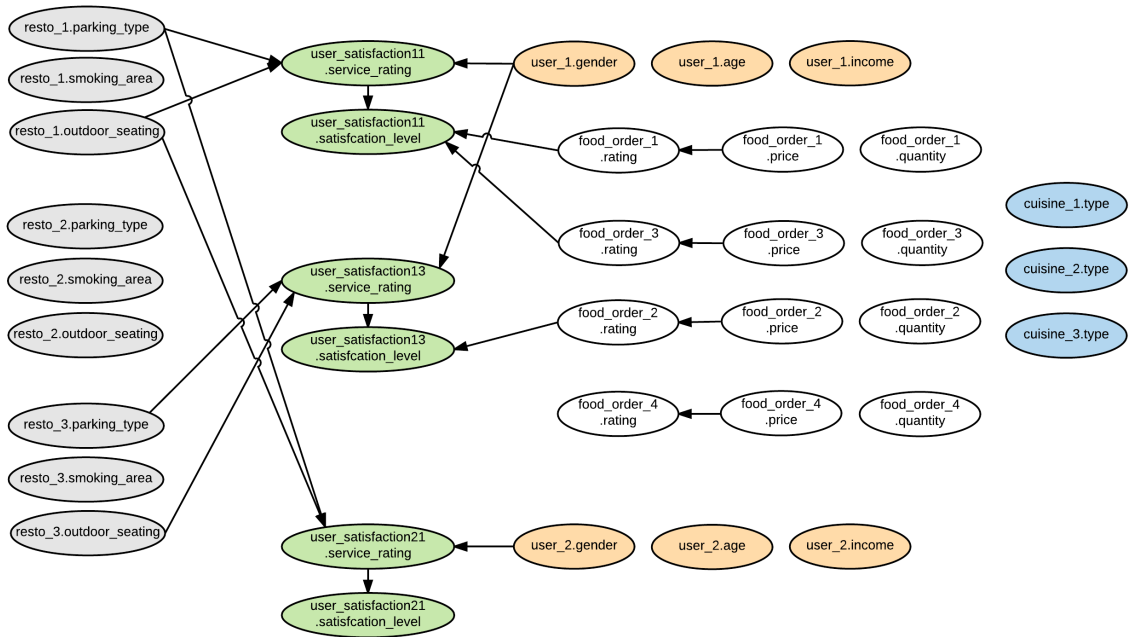


Figure 2.5 – Ground Bayesian network obtained by unrolling the PRM of Figure 2.4 over the relational skeleton of Figure 2.2. Colors are used to distinguish the class of the nodes and do not carry any significant meaning here. CPDs are not shown here to avoid cluttering.

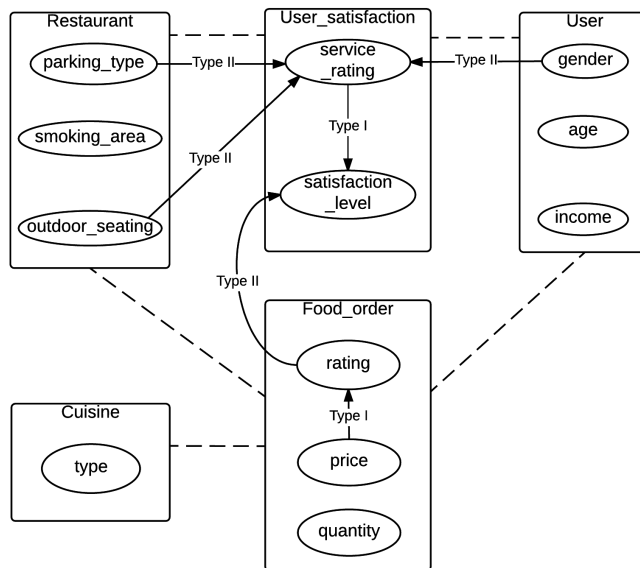


Figure 2.6 – Class dependency graph of the PRM in Figure 2.4

1. *Type I edges:* For any attribute $X.A$ and any of its parents $X.B$, we introduce an edge from $X.B$ to $X.A$.
2. *Type II edges:* For any attribute $X.A$ and any of its parents $X.K.B$ we introduce an edge from $Y.B$ to $X.A$, where $Y = Range[X.K]$. ■

The probabilistic model of a PRM is guaranteed to be coherent regardless of *any* relational skeleton if the corresponding class dependency graph is acyclic.

2.4 Extensions

Regular PRMs (cf. Definition 8) provide a model for domains where attribute values are uncertain. In these models, all relations between attributes are determined by the relational skeleton; uncertainty exists in the descriptive attributes only. [Getoor \[2001\]](#) have extended regular PRMs to deal with the cases where both attributes and link structure are uncertain.

2.4.1 PRM with structural uncertainty

A PRM with structural uncertainty is an extension to a regular PRM and provides probabilistic models for relational or link structure too. [Getoor \[2001\]](#) have proposed the following two mechanisms to model link uncertainty.

PRM with Reference Uncertainty (PRM-RU) It models uncertainty over the value of reference slots. The probability distribution for this uncertainty is defined over the set of all objects of the domain class of the reference slot. In general, this distribution would be very large. To achieve a compact representation, a *partition function* is used to partition the domain class into subsets, and smaller probability distribution is defined over these subsets. [Coutant \[2015\]](#) has presented a detailed study of *Probabilistic Relational Model with Reference Uncertainty (PRM-RU)*.

PRM with Existence Uncertainty (PRM-EU) It provides probabilistic models for existence of relations between objects too. Under this extension, we assume that we are given entity classes only, and the existence of the objects of relationship classes is uncertain. In other words, this model deals with the task of predicting links between objects. [Getoor \[2001\]](#) proposed to add a relationship class with a binary existence attribute $X.Exists$ which indicates whether the relationship object actually exists or not. Then, the existence attribute $X.Exists$ would be a descriptive attribute in a PRM-EU. [Huang et al. \[2004\]](#) have shown that PRM with existence uncertainty can be interesting for recommendation applications.

2.4.2 Other extensions

[Getoor \[2001\]](#) have proposed another extension of regular PRMs, called *Probabilistic Relational Models with Class Hierarchy (PRMs-CH)*, which provide refined probabilistic models using class hierarchies. With the introduction of subclasses, a PRM-CH allows to specialize probabilistic dependencies and CPDs within particular subclasses. [Newton and Greiner \[2004\]](#) have proposed *Hierarchical Probabilistic Relational Model (hPRM)*, an adaptation of PRM-CH, for collaborative filtering. More recent extensions include *Probabilistic Relational Model with Clustering Uncertainty (PRM-CU)* ([Coutant et al. \[2015\]](#)), PRM with relational uncertainty ([Fersini et al. \[2009\]](#)), and Hybrid Probabilistic Relational Models ([Närman et al. \[2010\]](#)).

2.5 Inference in PRMs

The traditional approach to inference in PRMs is to apply BN inference algorithms on the GBN obtained by unrolling a PRM for the given relational skeleton σ_r . Theoretically, standard inference algorithms for Bayesian networks (cf. Section 2.2.2) can

be used to query the GBN but it may be impractical because GBNs tend to be very big for real datasets. When GBNs are small, exact inference can be performed. Large and complex GBNs, however, limit the application of exact inference algorithms. Moreover, generation of such propositionalized models is itself too costly. This issue has already been raised in early works (Pfeffer [2000]) in this field. Kaelin [2011] have proposed a method for performing approximate inference in PRMs. Using the fact that a query can be answered in a Bayesian network by taking into account only the subgraph that contains all event nodes and is d-separated from the full GBN given the evidence nodes, the method constructs a partial GBN for the given query and apply Gibbs sampling method for approximate inference. Their proposed method, *Lazy Aggregation Block Gibbs (LABG)*, is listed in Algorithm 2. Recent works (Wuillemin and Torti [2012], Kisynski and Poole [2009], Milch et al. [2008], Singla and Domingos [2008]) advocate lifted probabilistic inference, which aims at performing as much inference as possible without propositionalizing.

Algorithm 2 Lazy Aggregation Block Gibbs (LABG)

Input: Query, $\mathbb{Q} = (\mathbb{Y}, \mathbb{E})$, where $\mathbb{Y} \subseteq \mathcal{A}(\sigma_r)$ is a set of event variables and $\mathbb{E} \subseteq \mathcal{A}(\sigma_r)$ is a set of evidence variables; Number of samples, N

Output: $P(\mathbb{Y} \mid \mathbb{E})$

- 1: $\mathbb{S} \leftarrow$ Unroll GBN for \mathbb{Q}
 - 2: $P_\phi \leftarrow$ Compute Full Conditional for $x.A \in \mathbb{S}$
 - 3: $s^{(0)} \leftarrow$ Sample initial state
 - 4: **for** $t = 1$ to N **do**
 - 5: $s^{(t)} \leftarrow s^{(t-1)}$
 - 6: $X.A \leftarrow$ Select an attribute in $\mathcal{A}(\mathbb{S})$
 - 7: *LazyAggregation*($X.A$), if necessary
 - 8: **for** all $x.A \in \mathbb{S}(X.A)$ **do**
 - 9: *Aggregation*($x.A$), if necessary
 - 10: $s^{(t)}\langle x.A \rangle \leftarrow$ Sample $P_\phi(x.A)$
 - 11: **end for**
 - 12: **end for**
 - 13: $P(\mathbb{S} \mid \mathbb{E}) \leftarrow$ Density Estimate of $\{s^{(0)}, \dots, s^{(N)}\}$
 - 14: $P(\mathbb{Y} \mid \mathbb{E}) \leftarrow$ Marginalize $\mathbb{S} \setminus \{\mathbb{Y}\}$ from $P(\mathbb{S} \mid \mathbb{E})$
-

2.6 Learning PRMs

As with Bayesian networks, learning a PRM also involves two tasks – learning parameters and learning the dependency structure.

2.6.1 Learning parameters

Given a dependency structure \mathcal{S} and an instance \mathcal{I} , the task in parameter estimation is to learn a parameter set $\theta_{\mathcal{S}}$ that defines the CPD for this structure. For a PRM, the likelihood of a parameter set $\theta_{\mathcal{S}}$ is: $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) = P(\mathcal{I}, \mathcal{S}, \theta_{\mathcal{S}})$. Parameter estimation is performed using standard statistical or Bayesian method. Sufficient statistics are computed in MLE over \mathcal{I} using queries for relational data (e.g. SQL queries in relational database).

2.6.2 Learning structure

PRM structure learning is inspired from classical methods of learning standard BN structures. Friedman et al. [1999] used search-and-score approach (cf. Section 2.2.2) to learn the structure of regular PRMs. Like in BN score-based methods, their relational extension of greedy search algorithm, *Relational Greedy Search (RGS)*, determines the neighboring structure of the starting network, assigns a score to each of them, selects the best scoring one, and iterates through the same process of searching for neighbors and scoring them until a stopping criterion is met. Unlike in BNs, there is, however, a constraint in assigning edges between nodes in a PRM while determining neighboring structures: the edges must be between two attributes from either the same class or the classes reachable through reference slots. Friedman et al. [1999] proposed walking through the slot chains to discover potential parents for each attribute and applying the search procedure on this set of parents. In order to avoid infinite space of relational attributes, the algorithm proceeds in multiple phases keeping the length of slot chain fixed at each phase. The search for potential parents and the corresponding structure begins with the slot chains of length 0 which is increased by 1 at each phase until a predefined slot chain length limit is reached or there is no improvement in the PRM structure. Algorithm 3 lists their overall approach of learning the structure of PRMs, and Algorithm 4 lists the procedure for generating neighboring PRM structures using ‘Add_edge’, ‘Delete_edge’ and ‘Revert_edge’ operations.

Friedman et al. [1999]’s RGS algorithm has been the basis for learning other extensions of PRMs. Getoor [2001] add two new operators ‘abstract’, and ‘refine’ to extend the search space while learning PRM-RU. As an alternative to RGS algorithm, Ben Ishak [2015] have proposed *Relational Max-Min Hill Climbing (RMMHC)*, an adaptation of MMHC algorithm (Tsamardinos et al. [2003]) for learning structure of regular PRMs.

Algorithm 3 Relational Greedy Search

Input: Initial dependency graph, \mathcal{G} ; Relational schema, \mathcal{R} ; Scoring function, $Score$;
Maximum slot chain length, SL_{max}

Output: Local optimal dependency graph, \mathcal{G}'

```

1:  $\mathcal{G}' \leftarrow \mathcal{G}$ 
2:  $S_{max} \leftarrow Score(\mathcal{G}')$ 
3:  $SL \leftarrow 0$  ▷ Current slot chain length
4: repeat
5:   repeat
6:      $\mathcal{N} \leftarrow \text{Generate\_Neighbors}(\mathcal{G}', \mathcal{R}, SL)$ 
7:      $N^* \leftarrow \arg \max_{N' \in \mathcal{N}} Score(N')$ 
8:      $S^* \leftarrow Score(N^*)$ 
9:     if  $S^* > S_{max}$  then
10:        $\mathcal{G}' \leftarrow N^*$ 
11:        $S_{max} \leftarrow S^*$ 
12:     end if
13:   until No change in  $\mathcal{G}'$ 
14:    $SL \leftarrow SL + 1$ 
15: until  $SL > SL_{max}$ 

```

Algorithm 4 Generate_Neighbors (PRM)**Input:** A PRM, $\Pi = \langle \mathcal{R}, \mathcal{S} \rangle$; Slot chain length, SL ; Available aggregators, Agg **Output:** Neighbors of \mathcal{S} , \mathcal{S}'

```

1:  $\mathcal{S}' \leftarrow \{\}$ 
2: for each  $X.A \in \mathcal{A}(X)$  and  $X \in \mathcal{R}$  do
3:    $P \leftarrow \text{Find\_Accessible\_Classes}(X.A, \mathcal{R}, SL)$   $\triangleright$  All classes in  $\mathcal{R}$  accessible from
    $X.A$  with the given slot chain length  $SL$ 
4:   for each  $\langle Y, \rho \rangle \in P$  and  $Y.B \in \mathcal{A}(Y)$  do
5:     if  $(X.\rho.B, X.A) \notin \mathcal{S}$  then  $\triangleright$  If the edge  $(X.\rho.B, X.A)$  does not exist in  $\mathcal{S}$ ,
   add it
6:                                      $\triangleright \text{Add\_edge}(\mathcal{S}, X.\rho.B, X.A)$ 
7:       if  $\rho$  contains at least one reverse slot then
8:         for  $\gamma \in Agg(Y.B)$  do
9:            $\mathcal{S}'' \leftarrow \mathcal{S} \cup \{(\gamma(X.\rho.B), X.A)\}$ 
10:          if  $\mathcal{S}''$  has no cycle then
11:             $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}''$ 
12:          end if
13:        end for
14:      else
15:         $\mathcal{S}'' \leftarrow \mathcal{S} \cup \{(X.\rho.B, X.A)\}$ 
16:        if  $\mathcal{S}''$  has no cycle then
17:           $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}''$ 
18:        end if
19:      end if
20:    else  $\triangleright$  If the edge  $(X.\rho.B, X.A)$  exists in  $\mathcal{S}$ , remove it or revert it
    $\triangleright \text{Delete\_edge}(\mathcal{S}, X.\rho.B, X.A)$ 
21:       $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S} \setminus \{(X.\rho.B, X.A)\}$ 
22:       $\triangleright \text{Revert\_edge}(\mathcal{S}, X.\rho.B, X.A)$ 
23:       $Y.\rho' \leftarrow \text{Reverse}(X.\rho)$ 
24:      if  $\rho'$  contains at least one reverse slot then
25:        for  $\gamma \in Agg(X.A)$  do
26:           $\mathcal{S}'' \leftarrow \mathcal{S} \setminus \{(X.\rho.B, X.A)\} \cup \{(\gamma(Y.\rho'.A), Y.B)\}$ 
27:          if  $\mathcal{S}''$  has no cycle then
28:             $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}''$ 
29:          end if
30:        end for
31:      else
32:         $\mathcal{S}'' \leftarrow \mathcal{S} \setminus \{(X.\rho.B, X.A)\} \cup \{(Y.\rho'.A, Y.B)\}$ 
33:        if  $\mathcal{S}''$  has no cycle then
34:           $\mathcal{S}' \leftarrow \mathcal{S}' \cup \mathcal{S}''$ 
35:        end if
36:      end if
37:    end if
38:  end for
39: end for
40: end for

```

Evaluating candidate structures

While learning structure of a PRM, we need to compare different structures to find the one that fits the data well. In Bayesian model selection approach, the best structure is the one which maximizes its posterior probability given an instantiation \mathcal{I} . As we need to consider the relational skeleton σ_r of the given instantiation \mathcal{I} for a PRM, we need to find the structure that maximizes $P(\mathcal{S} | \mathcal{I}, \sigma_r) \propto P(\mathcal{S} | \sigma_r)P(\mathcal{I} | \mathcal{S}, \sigma_r)$.

Assuming that the choice of structure is independent of the skeleton, the prior probability of the structure $P(\mathcal{S} | \sigma_r)$ will be $P(\mathcal{S})$. This prior is often considered uniform for BNs, but due to the infinite number of possible slot chains that can be derived from a relational schema, uniform prior is not relevant for PRMs. Thus, this prior needs to be selected prioritizing simple structures. Getoor et al. [2007] penalize long indirect slot chains by having $\log P(\mathcal{S})$ proportional to the sum of the lengths of the slot chains \mathbf{K} appearing in \mathcal{S} .

$P(\mathcal{I} | \mathcal{S}, \sigma_r)$ is the marginal distribution of the joint probability distribution over the set of parameters of the structure, i.e.,

$$P(\mathcal{I} | \mathcal{S}, \sigma_r) = \int_{\Theta} P(\mathcal{I} | \Theta, \mathcal{S}, \sigma_r)P(\Theta | \mathcal{S}, \sigma_r)d\Theta$$

where, $P(\mathcal{I} | \Theta, \mathcal{S}, \sigma_r)$ is defined by Equation 2.2, and $P(\Theta | \mathcal{S}, \sigma_r)$ is the prior over parameters.

Computation of this integral is simplified if we can decompose this integral into a product of simpler integrals. This can be achieved by the use of, for example, Dirichlet prior. In this case, $p(\mathcal{I} | \mathcal{S}, \sigma_r)$ will be

$$p(\mathcal{I} | \mathcal{S}, \sigma_r) = \prod_{X \in \mathcal{X}} \prod_{A \in \mathcal{A}(X)} \prod_{\mathbf{u} \in \mathcal{V}(Pa(X.A))} \text{DM}(\{C_{X.A}[v, \mathbf{u}]\}, \{\alpha_{X.A}[v, \mathbf{u}]\}) \quad (2.3)$$

where $\alpha_{X.A}[v, \mathbf{u}]$ are hyper-parameters of Dirichlet prior, $C_{X.A}[v, \mathbf{u}]$ are sufficient statistics for $v \in \mathcal{V}(X.A)$,

$$\text{DM}(\{C[v]\}, \{\alpha[v]\}) = \frac{\Gamma\left(\sum_v \alpha[v]\right)}{\Gamma\left(\sum_v (\alpha[v] + C[v])\right)} \prod_v \frac{\Gamma(\alpha[v] + C[v])}{\Gamma(\alpha[v])}$$

and $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ is a Gamma function.

2.7 Evaluating PRM learning algorithms

PRM learning algorithms can be evaluated by following the process of evaluation of Bayesian networks. Standard way to evaluate learning algorithms in Bayesian networks is to generate a database from a *gold-standard network*, then learn a network from the generated database, and compare the *learned network* with the gold-standard network. Usually, well-known networks such as MUNIN (Andreassen et al. [1989]), Barley (Kristensen and Rasmussen [2002]), Insurance (Binder et al. [1997]) etc., are taken as gold-standard networks. Unfortunately, no such networks are available for PRMs. Another option is to start with an arbitrary or synthetic network. Ben Ishak [2015] has proposed an algorithmic approach to generate random PRMs, which they use to obtain synthetic datasets for evaluating PRM structure learning algorithms. The

following sections explain their approach of evaluating PRM learning algorithms using synthetic datasets. Section 2.7.1 presents their strategy for comparing PRMs to assess learning algorithms, and the evaluation metrics they have used. Section 2.7.2 explains their algorithm for generating PRM benchmarks. In Section 2.7.3, we point out the shortcomings of their approach, and propose some improvements in Section 2.7.4.

2.7.1 Evaluation strategy and metrics

Two approaches of comparing Bayesian networks can be found in the literature – (1) by comparing DAG structures, and (2) by comparing equivalence classes (de Jongh and Druzdzel [2009]). Several metrics (Heckerman et al. [1995], Spirtes et al. [2000], Tsamardinos et al. [2006] etc.) are well established and widely used for comparing Bayesian networks. Though PRMs are based on Bayesian networks, these metrics are not directly applicable for comparing PRMs because firstly, the presence of slot chains and aggregators in PRMs makes it complicated to compare the DAG structures directly, and secondly, the notion of equivalence classes is not yet developed for PRMs. Maier et al. [2013] have used precision, recall, and F-score metrics to measure structural difference between two RBNs by comparing their DAG structures. Ben Ishak [2015] has refined these metrics by adding penalization for wrong slot chains and wrong aggregators, and have proposed hard and soft versions for precision, and recall.

Let \mathcal{S}_{true} be the dependency structure of the gold standard PRM, $\mathcal{S}_{learned}$ be the dependency structure of the learned PRM, Nb_{true} be the number of dependencies in \mathcal{S}_{true} , and $Nb_{learned}$ be the number of dependencies in $\mathcal{S}_{learned}$. Then,

$$\text{hard_Precision} = \frac{\text{Number of relevant dependencies retrieved in } \mathcal{S}_{learned}}{Nb_{learned}} \quad (2.4a)$$

$$\text{hard_Recall} = \frac{\text{Number of relevant dependencies retrieved in } \mathcal{S}_{learned}}{Nb_{true}} \quad (2.4b)$$

$$\text{soft_Precision} = \frac{\sum_{i=0}^{Nb_{learned}} \omega_i}{Nb_{learned}} \quad (2.4c)$$

$$\text{soft_Recall} = \frac{\sum_{i=0}^{Nb_{learned}} \omega_i}{Nb_{true}} \quad (2.4d)$$

where relevant dependencies are the ones that have the same edge, slot chain and aggregator as in the gold standard model,

$$\omega_i = \begin{cases} 1, & \text{for relevant dependencies} \\ 0, & \text{for reversed edges and the edges not present in the gold standard model} \\ 1 - \psi, & \text{when the edges match but slot chains and/or aggregators do not} \end{cases}$$

ψ is the arithmetic mean of the penalization for wrong slot chains (α) and that for wrong aggregators (β) in true and learned dependencies, i.e.,

$$\psi = \frac{\alpha + \beta}{2} \quad (2.5)$$

$$\alpha = 1 - \frac{\text{Length of the longest common sub slot chain in true and learned dependencies}}{\text{Max}(\text{Length of the true dependency, Length of the learned dependency})}$$

and $\beta \in [0, 1]$ is the user-defined cost for penalizing wrong aggregators.

F-score is the harmonic mean of precision and recall, and is given by the following equation.

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.6)$$

Ben Ishak [2015] has also proposed a metric, called *Relational Structural Hamming Distance (RSHD)*, for comparing *Directed Acyclic Probabilistic Entity Relationship (DAPER)*³ models. This metric can be applied only on PRMs that are DAPERs.

2.7.2 Generating PRM benchmarks

The PRM benchmark generation process proposed by Ben Ishak [2015] is depicted in Figure 2.7. The process involves 3 steps – 1) generation of a random PRM, 2) instantiation of the generated PRM with a random relational skeleton, and 3) sampling the ground Bayesian network obtained as a result of the PRM instantiation. In the first step, a random relational schema is created, and the dependency structure is specified by generating random dependencies between attributes present in the schema. Random CPDs are then assigned to each attribute to define a complete PRM. For the second step, the author has proposed an algorithm to generate a random relational skeleton given the approximate number of objects per class. A GBN is then generated by instantiating the PRM over the generated relational skeleton, and finally, in the third step, a standard sampling algorithm for sampling Bayesian networks is applied on this network to generate a sample, which is then stored in a database. These steps are listed in Algorithm 5.

Generation of a random relational schema

Ben Ishak [2015] treats the task of generating a relational schema as the process of generating a *connected DAG* such that a node in the graph corresponds to a class in the schema and an edge corresponds to a reference from one class to another. Thus, a relational schema generated in that way will not have any cyclic references, and each class will be related to any other classes through slot chains.

Generation of a random dependency structure

To complete the process of random PRM generation, a dependency structure \mathcal{S} together with CPDs for each node in \mathcal{S} are required. In Ben Ishak [2015]’s approach,

3. Similar to a PRM, a DAPER model describes probabilistic dependencies between attribute classes of entity and relationship classes through a directed acyclic graph, and each attribute class is associated to a local distribution. A PRM collapses to a DAPER when reference slots of a (relationship) class refer to two different (entity) classes (i.e., when a relationship table contains two foreign keys referring to two different tables in the relational schema).

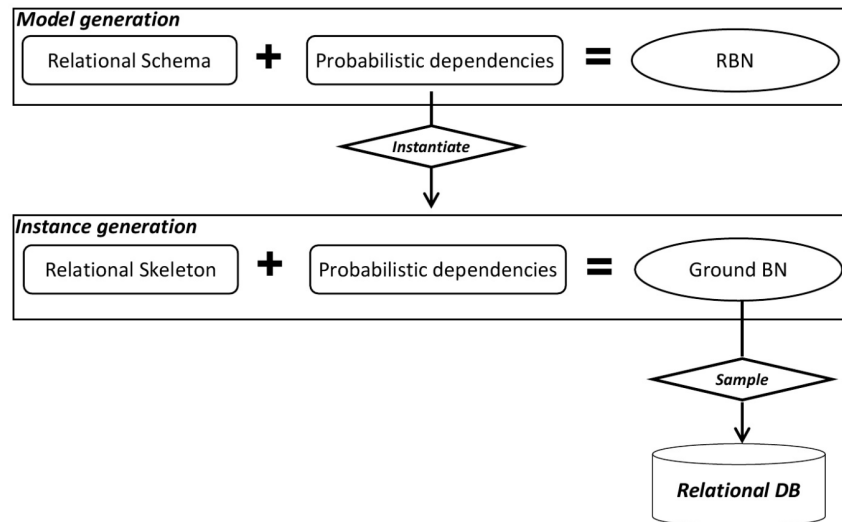


Figure 2.7 – Overview of PRM benchmark generation process proposed by Ben Ishak [2015].

they generate a DAG among the descriptive attributes present in the relational schema to ensure that the instantiation of the generated PRM would yield an acyclic GBN. To control the complexity of the structure, they limit the maximum length of slot chains, and give more preference to intra-class dependencies and shorter slot chains. Once a dependency structure is determined, random CPDs are assigned to all of its nodes to obtain a complete PRM.

Generation of a random relational skeleton

In the second step in Algorithm 5, the PRM generated in the first step is instantiated for a relational skeleton to obtain a GBN. Ben Ishak [2015] has proposed an algorithm to generate a relational skeleton, where they generate nearly the same number of objects of each class and iteratively add random links (or references) between objects of a pair of classes such that the direction of the links conform to the underlying schema.

Database population

To obtain a complete dataset without missing values, Ben Ishak [2015] applies forward sampling algorithm on the GBN obtained by instantiating the PRM for the relational skeleton generated in the previous steps. The authors have used well-known forward sampling (Henrion [1988]) algorithm implemented in ProBT API⁴ to sample GBNs in their experiments.

2.7.3 Limitations

Ben Ishak [2015] has successfully generated several benchmarks and applied their structure learning algorithm RMMHC. However, there are limitations at each phase of their approach of PRM benchmark generation.

Their schema generation process assumes that foreign keys cannot be empty. Also, it cannot generate schemas where a foreign key refers to the same class. Such schemas

4. www.probayes.com/~mazer/html/index.html

Algorithm 5 Generate_Random_PRM-DB (Ben Ishak [2015])

Input: Number of classes, N ; Maximum length of slot chain length, \mathcal{K}_{max}

Output: $\Pi : \langle \mathcal{R}, \mathcal{S}, \theta \rangle$; A database instance, \mathcal{I}

Step 1: Generate a random PRM

- i) $\mathcal{R} \leftarrow$ Generate a random relational schema with N number of classes
- ii) $\mathcal{S} \leftarrow$ Generate a set of dependencies between attributes of classes in \mathcal{R}
- iii) Determine slot chains on \mathcal{S} with maximum length \mathcal{K}_{max}
- iv) $\theta \leftarrow$ Generate CPDs for \mathcal{S}
- v) $\Pi \leftarrow \langle \mathcal{R}, \mathcal{S}, \theta \rangle$

Step 2: Instantiate the PRM

- i) $\sigma_r \leftarrow$ Generate a random relational skeleton conforming to \mathcal{R}
- ii) $\mathcal{G} \leftarrow$ Unroll Π for σ_r into a GBN

Step 3: Database population

- i) $\mathcal{I} \leftarrow$ Apply any Bayesian network sampling algorithm to generate a sample dataset
-

with cyclic references and null/empty foreign keys are very common in real-world applications. For instance, in a social network, users are related to another users, and this results in a cyclic reference in `User` class in the relational schema.

Their dependency generation algorithm is limited to the generation of PRMs that are guaranteed to be acyclic at class level. Their approach does not support the cases where only `IDG` is acyclic.

In real world, relational skeletons tend to be *scale-free*, i.e., degree of the vertices of the graph follows *power-law* distribution (Newman [2003]). Their approach does not create realistic skeletons.

Though their approach of database population is theoretically possible, it may be impractical when it comes to the generation of very big datasets because in such case, the GBN would be huge. Moreover, GBN generation is itself an expensive task.

Evaluation of the generated datasets is missing in their approach. They evaluate their structure learning algorithm on the generated datasets without actually assessing the quality of the generated datasets. If the dataset is not consistent with the model from which it is generated, then comparing the model learned from this dataset with the original model does not make any sense. Thus, it is necessary to assess the quality of the generated dataset before evaluating structure learning algorithms on those datasets. For this, we can use Chi-square goodness-of-fit test. Null hypothesis of this test is that the generated data for the given node are consistent with the original distributions used for generating the sample. The nodes that reject the test cannot be considered well-sampled.

2.7.4 Proposals for improvement

Here, we propose some improvements to overcome shortcomings of Ben Ishak [2015]’s approach to PRM benchmark generation. Our proposals will mainly address the limitations related to skeleton generation and database population. In the following, we present a novel algorithm to generate scale-free relational skeletons, and an adaptation of forward sampling and LABG algorithms for sampling PRMs.

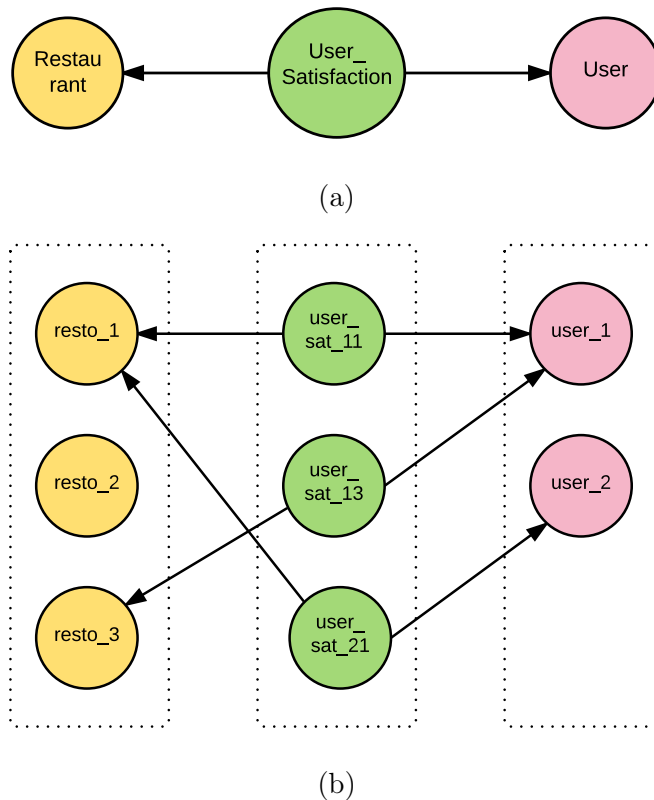


Figure 2.8 – (a) Relational schema DAG of Figure 2.1a (considering only three classes), (b) Relational skeleton of Figure 2.2a as a k -partite graph

Generating realistic skeletons

In general, relational skeletons in real-world applications tend to be scale-free. In real datasets, the number of objects for classes with foreign keys tends to be very high compared to that for classes which do not have foreign keys and are referenced by other classes. We propose a novel approach to generate a realistic relational skeleton.

A relational skeleton can be imagined as a DAG where nodes are objects of different classes present in the associated relational schema and edges are directed from one object to another conforming to the reference slots present in the relational schema. For example, the relational skeleton of Figure 2.2a can be seen as a graph shown in Figure 2.8b. Here, we have considered objects of only three classes to make the graph small. Nodes in this graph represent objects, and edges indicate the foreign key constraints. Clearly, this graph is a 3-partite graph with three independent sets of objects, each set corresponding to one of the classes in the relational schema (Figure 2.8a) such that no two objects in an edge belong to the same set. This graph is, in fact, a special case of k -partite graph of Definition 12. In this regard, relational skeleton generation process can be considered as a problem of generating objects and assigning links (or foreign keys) between them such that the resulting graph is a k -partite graph of Definition 12. Our approach to generating such k -partite graph is presented in Algorithms 6 and 7. We adapt Bollobás et al. [2003]’s directed scale-free graph generation algorithm for our special k -partite graph and use *Chinese Restaurant Process (CRP)* (Pitman [2002]) to apply preferential attachment.

Definition 12 Relational skeleton as a k -partite graph

A relational skeleton (of a relational schema with k classes) is a special case of k -partite

Algorithm 6 Generate_Relational_Skeleton

Input: Relational Schema as a DAG, $\mathcal{G} = (V_g, E_g)$; Total number of objects in the resulting skeleton, N_{total} ; Scalar parameter for CRP, α ; Percentage of unreferenced objects, β

Output: A relational skeleton, $\mathcal{I} = (V, E)$

```

1: for  $node \in V_g$  do
2:    $N(node) \leftarrow 0$             $\triangleright$  Total number of objects of each type generated so far
3: end for
4:  $V \leftarrow \{\}$                     $\triangleright$  Set of objects
5:  $E \leftarrow \{\}$                     $\triangleright$  Set of directed edges between objects
6:  $m \leftarrow$  Number of nodes without any parents in  $\mathcal{G}$  (number of roots)
7: if  $m > 1$  then
8:   Divide  $\mathcal{G}$  into  $m$  subgraphs such that each subgraph contains a root and all of
   its descendants.
9: end if
10: repeat
11:   if  $m > 1$  then
12:      $g \leftarrow$  one of the  $m$  subgraphs picked randomly
13:   else                                $\triangleright$  i.e., if  $\mathcal{G}$  has only one root
14:      $g \leftarrow \mathcal{G}$ 
15:   end if
16:    $obj_{root} \leftarrow$  A new object of the root of  $g$ 
17:    $n_{root} \leftarrow n_{root} + 1$ 
18:    $children \leftarrow$  Children of the root in  $g$ 
19:    $((V', E'), N') \leftarrow$  Generate_SubSkeleton( $obj_{root}, g, children, N, \alpha$ )    $\triangleright$  Perform
   depth-first search over  $g$  and add edges recursively
20:    $V \leftarrow V \cup V'$ 
21:    $E \leftarrow E \cup E'$ 
22:    $N \leftarrow N'$             $\triangleright$  Update the set of number of generated objects of each type
23:    $n \leftarrow cardinality(V)$     $\triangleright$  Total number of objects generated so far.
24: until  $n \geq N_{total} * (1 - \beta)$ 
25: Generate  $N_{total} * \beta$  unreferenced objects of the classes which do not have any children
26:  $\mathcal{I} \leftarrow (V, E)$ 

```

graph, $G_k = (V_k, E_k)$, with the following properties:

1. The graph is acyclic,
2. All edges are directed (an edge $u \rightarrow v$ indicates that the object u refers to v , i.e., u has a foreign key which refers to the primary key of v),
3. Edges between two different types of objects are always oriented in the same direction, i.e. for all edges $(u - v)$ between objects of U and V where $u \in U$, and $v \in V$, the direction of all edges must be either $u \rightarrow v$ or $u \leftarrow v$ and not both,
4. Both in-degree and outdegree of any object can be greater than 1 but there must not be more than one edge between any two objects. ■

In our approach, we view a relational schema as a DAG and instantiate this graph into a k -partite graph. The basic idea here is to iteratively generate an object of a class with no parents in the relational schema DAG and then recursively add an edge from this object to objects of its children classes. This process is essentially a *Depth-First*

Search (DFS), where we begin by generating an object of the root node of the graph and then at each encounter of a node in *DFS*, we add an edge from the object of the parent node to either a new or an existing object of the encountered node. The object of the parent node gets connected to a new object with probability $p = \alpha / (n_p - 1 + \alpha)$, where n_p is the total of objects of the parent node generated so far, and α is a scalar parameter for the process. When it gets attached to an existing object, an object of the correct type is picked randomly from the set of existing objects. For a true *CRP*, we need to select an object at this step with probability $n_k / (n_p - 1 + \alpha)$, where n_k is the in-degree of the object to be selected and n is the total number of objects generated so far. Thus, as the skeleton graph grows, probability of getting connected to new objects will decrease and the objects with higher in-degree will be preferred for adding new edges. However, when there are objects with very high in-degree, *GBN* for this skeleton may become very complex. Therefore, in our implementation of this algorithm (will be explained in Chapter 9), we keep both options – picking an object uniformly at random or picking it based on its in-degree. At each iteration, a *DFS* is performed starting from one of the nodes without parents in the relational schema *DAG*. Thus, if there is only one node that does not have any parent, then each iteration will visit all classes in the relational schema resulting in a complete set of objects and relations for all classes, otherwise only a subset of classes will be visited in each iteration. So, at the beginning of each iteration, one of the nodes without parents is picked randomly in the latter case. The iteration process is continued until the skeleton contains the required number of objects. If some dangling objects (i.e., objects which are not referenced by any object at all) are required, we can generate some objects of the leaf classes at the end.

Note that this algorithm ensures that foreign keys will never be null. For relational schemas that can have null foreign keys, our algorithm needs to be modified such that in some iterations, a complete *DFS* is not performed. That way, we can get objects without children even though the corresponding classes in the underlying schema *DAG* have children. However, in this thesis, we work only with those kind of schemas where we can never have a null foreign key.

Example 2.4 *k*-partite graph-based relational skeleton generation

We illustrate our approach by generating a relational skeleton for a simple relational schema shown in Figure 2.9. Colors used in the relational schema DAG do not bear any meaning here. They are used only to distinguish different classes. Algorithm 6 creates a relational skeleton for this schema by performing DFS on the schema DAG. The first three iterations of the DFS are shown in figures 2.10 and 2.11. As the schema has only one node without any parent (i.e., a class without any foreign key), one complete DFS returns a set of objects of each class as shown in Figure 2.10. At each iteration, we obtain different number of objects. As we can see in Figure 2.11, the first iteration created five objects whereas the second and third iteration resulted in four and two objects respectively. We continue the iteration until we obtain the required number of objects in the skeleton.

Figure 2.10 shows one iteration of a DFS performed on the schema. A DFS will visit the classes in the following order: Class3 → Class2 → Class1 → Class0 → Class1 → Class1. In each of the sub-figures in this figure, the upper graph is the concerned relational schema and the lower graph is the relational skeleton being generated. The node (and the edge) encountered at each step of the DFS is shown by thick lines. Colors are used to only as a visual aid to distinguish between objects of different classes and add no significant meaning to the process. We begin by selecting a node of the schema

Algorithm 7 Generate_SubSkeleton

Input: Parent object obj_p ; Graph g ; Parent node, $parent$; Children nodes, $children$;
Set of the number of objects of each class generated so far, N ; Scalar parameter α

Output: Relational skeleton, $\mathcal{I} = (V, E)$; Set of the number of objects of each class generated so far, N

- 1: $V \leftarrow \{obj_p\}$
- 2: $E \leftarrow \{\}$
- 3: $n_p \leftarrow N(parent)$ ▷ Total number of parents generated so far
- 4: **for** $C \in children$ **do**
- 5: $n_c \leftarrow N(C)$ ▷ Total number of the child C generated so far
- 6: $p \leftarrow \alpha / (n_p - 1 + \alpha)$
- 7: $r \leftarrow$ A random value between 0 and 1 ▷ $r \in [0, 1]$
- 8: **if** $r \leq p$ **then**
- 9: $obj_c \leftarrow$ Create a new object of type C
- 10: $N(C) \leftarrow n_c + 1$
- 11: $e_{pc} \leftarrow (obj_p, obj_c)$ ▷ Add an edge from obj_p to obj_c
- 12: $E \leftarrow E \cup \{e_{pc}\}$
- 13: $children_c \leftarrow$ Children of C in the graph g
- 14: $((V', E'), N') \leftarrow$ Generate_SubSkeleton($obj_c, g, C, children_c, N, \alpha$)
- 15: $V \leftarrow V \cup V'$
- 16: $E \leftarrow E \cup E'$
- 17: $N \leftarrow N'$
- 18: **else**
- 19: $obj_c \leftarrow$ An existing object of type C picked randomly with probability $1/n_c$
▷ For true CRP, pick an existing object of type C with a probability $n_k / (n - 1 + \alpha)$ where $n_k =$ in-degree of obj_c (instead of picking randomly)
- 20: $e_{pc} \leftarrow (obj_p, obj_c)$ ▷ Add an edge from obj_p to obj_c
- 21: $E \leftarrow E \cup \{e_{pc}\}$
- 22: **end if**
- 23: **end for**
- 24: $\mathcal{I} \leftarrow (V, E)$

DAG that does not have a parent. So, we create a new object of the node ‘Class3’ (Figure 2.10a) as it does not have any parent. Then, we traverse to one of the children of this node in the schema (‘Class2’ here). As there is no object of this class so far, a new object will be created (Figure 2.10b). Now, continuing the *DFS*, we encounter the node ‘Class1’ (the child of ‘Class2’), and then ‘Class0’ (a child of ‘Class3’). Like earlier, new objects of ‘Class1’ and ‘Class0’ will be generated (Figures 2.10c and 2.10d). As ‘Class0’ has a child, we reach ‘Class1’. At this step, an object of ‘Class1’ is already present. So, the object ‘Class01’ can either create a new object or get connected to ‘Class11’. In this example, it gets linked to a new object ‘Class12’ (Figure 2.10e). In the next step of the *DFS*, ‘Class1’ is encountered again as it is a child of ‘Class3’. Here, ‘Class31’ gets attached to an existing object of ‘Class1’ (Figure 2.10f).

Next two iterations of *DFS* are shown in Figure 2.11. At each iteration, a new object of ‘Class3’ will always be generated as it does not have any parent. The object will then be linked to an existing object or a new one, and the same thing goes on for the new objects. Here, the skeleton after the first iteration has five objects. The second iteration creates four new objects, whereas the third iteration creates only two objects.

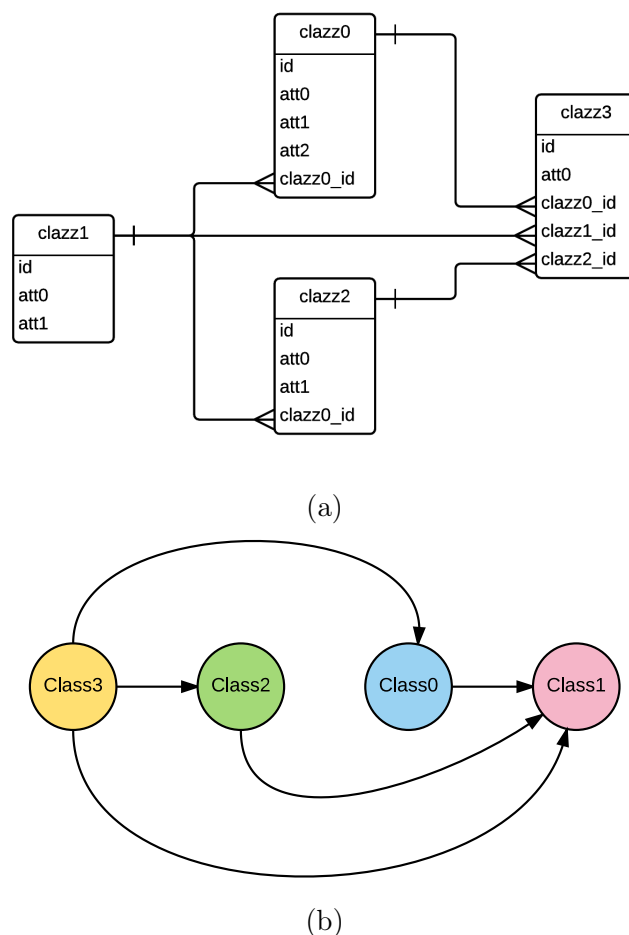


Figure 2.9 – A dummy relational schema: (a) Entity-Relationship diagram, (b) as a directed graph

If needed, we can add dangling objects after completing the iterations. In this example, only ‘Class1’ can have dangling objects as it does not have any foreign keys. ❖

PRM sampling

To obtain a complete dataset without missing values, Ben Ishak [2015] applies forward sampling algorithm on the GBN obtained by instantiating the PRM for the relational skeleton generated in the previous steps. Though this approach is theoretically possible, it may be impractical when it comes to generate very big datasets because the GBN would be huge for big datasets. Moreover, GBN generation is itself an expensive task.

Relational forward sampling (Algorithm 8) aims at sampling a PRM without using a GBN. It adapts forward sampling algorithm (Henrion [1988]) for relational context and works directly with databases. This algorithm samples each PRM node in a topological order, and generates a random value for the corresponding attribute of all objects in the skeleton. Because this algorithm does not need to deal with GBN, GBN generation time is saved with this algorithm. The only time consuming operation in this algorithm is the communication with databases. A limitation of this approach is that it cannot be applied on partially observed skeletons, where some attributes are already observed. An option to overcome this could be to adapt this algorithm

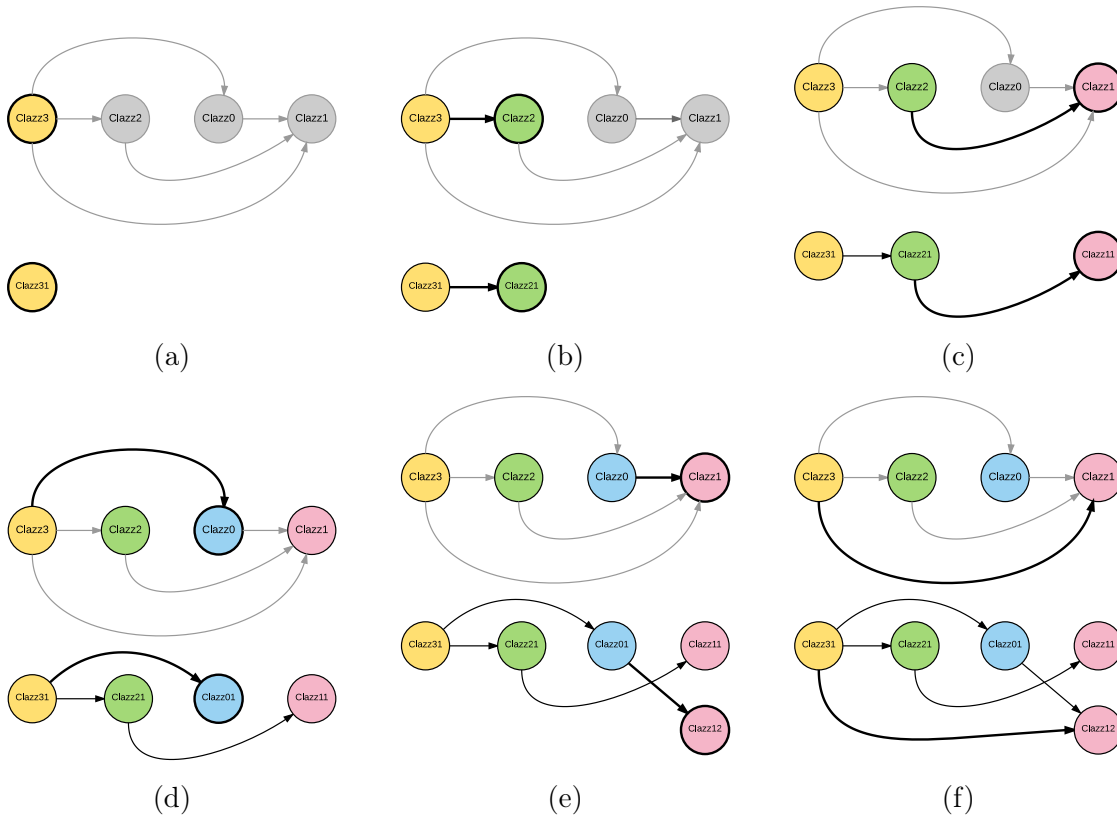


Figure 2.10 – Generating objects while performing Depth First Search (DFS) on the relational schema. This shows one iteration of a DFS performed on the schema.

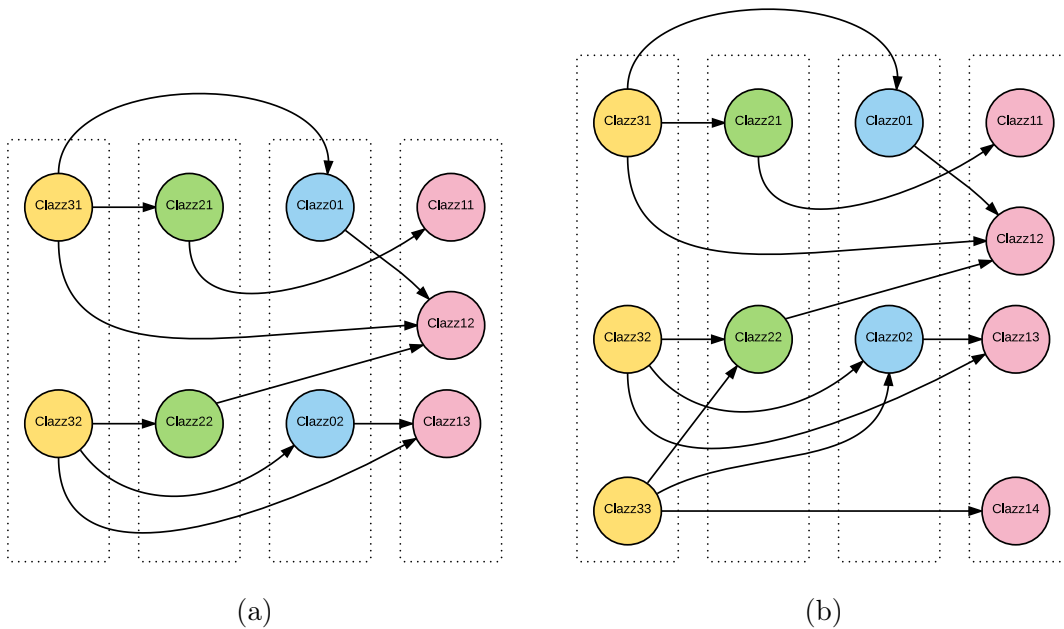


Figure 2.11 – Next two iterations of DFS on the relational schema of Figure 2.9 following the first iteration of Figure 2.10 to generate relational skeleton graph.

to perform *rejection sampling*. Alternatively, we can devise relational extensions of other BN sampling algorithms that support evidences. We propose *Relational Block Gibbs (RBG)* sampling algorithm, which is capable of dealing with partially observed

Algorithm 8 Relational forward sampling

Input: A PRM, $\Pi = \langle \mathcal{R}, \mathcal{S} \rangle$; A relational skeleton, σ_r **Output:** An instance (or a sample), \mathcal{I}

```

1:  $\mathcal{G} \leftarrow$  Dependency structure of  $\Pi$  in topological order
2: for each node  $X.A \in \mathcal{G}$  do
3:   for each object  $x \in \sigma_r(X)$  do
4:     if  $X.A$  has no parent then
5:       Sample  $x.A$  from  $P(X.A)$  and write to  $\mathcal{I}$ 
6:     else
7:        $e \leftarrow \{\}$ 
8:       for  $X.\gamma.B \leftarrow$  Parents of  $X.A$  do
9:          $Z.B \leftarrow X.\gamma.B$ 
10:        if Aggregation needed then
11:           $e \leftarrow e \cup$  Aggregate all  $z.B$  that have  $x.A$  as their child
12:        else
13:           $e \leftarrow e \cup z.B$  that has  $x.A$  as its child
14:        end if
15:      end for
16:      Sample  $x.A$  from  $P(X.A | e)$  and write to  $\mathcal{I}$ 
17:    end if
18:  end for
19: end for

```

skeletons.

RBG sampling algorithm is based on Kaelin [2011]’s LABG algorithm. Good points about RBG algorithm are that it can be applied on partially observed skeletons, and it can also support the PRMs that have cycles in class level but are guaranteed to be acyclic in instance level. LABG starts with a partial GBN induced by the query. In our case, the query is the set of all unobserved variables. This can lead to the generation of a complete GBN (when all attributes are not observed). To avoid this, only the structure of the GBN is generated in our approach; full CPDs are not computed for a couple of reasons because full CPDs are big tables and may require quite a good amount of memory for large and complicated GBNs. Besides, only a small number of values from these CPDs are required during actual sampling of the nodes. So, we compute those values only when required. After generating the structure and setting evidences, an initial sample is generated by assigning random values to unobserved nodes. This structure is imagined to be partitioned into blocks, where each block contains all nodes corresponding to the same attribute $X.A$. Then, an attribute $X.A$ (or a block) is randomly selected with probability proportional to the size of its block. For each unobserved node in that block, its Markov blanket is identified to compute full conditional distribution P_ϕ and the node is then sampled according to this distribution. The steps of selecting a block and performing Gibbs sampling is performed a finite number of times or until convergence. Algorithm 9 presents our approach.

In Appendix A, we will present an empirical study of these sampling algorithms, and show that relational forward sampling is very efficient in terms of time but the quality of datasets obtained from RBG is better.

Algorithm 9 Relational Block Gibbs sampling (based on Kaelin [2011]’s LABG)

Input: A PRM, Π ; A relational skeleton, σ_r , with or without observations, \mathbf{e} ; burn-in, N

Output: An instance (or a sample), \mathcal{I}

- 1: $\mathcal{G} \leftarrow$ Generate GBN structure of Π for σ_r
- 2: Set evidences if any
- 3: Sample initial states $s^{(0)}$
- 4: **for** $t = 1$ to N **do**
- 5: $s^{(t)} \leftarrow s^{(t-1)}$
- 6: $X.A \leftarrow$ Select an attribute for sampling
- 7: **for each** $x.A \in \mathcal{G}(X.A)$ **do**
- 8: **if** $x.A$ is not observed **then**
- 9:
$$P'_\phi(x.A) \leftarrow P(x.A \mid Pa(x.A)) \prod_{y.B \in Ch(x.A)} P(y.B \mid Pa(y.B))$$
- 10: $P_\phi \leftarrow \text{Normalize}(P'_\phi)$
- 11: $s^{(t)}\langle x.A \rangle \leftarrow \text{Sample } P'_\phi(x.A)$
- 12: **end if**
- 13: **end for**
- 14: **end for**

2.8 Conclusion

In this chapter, we introduced Statistical Relational Learning (SRL) and Probabilistic Relational Models (PRMs). As PRMs operate on relational data and are based on Bayesian Networks (BN), we started with a short introduction to relational data representation as considered in this thesis, and Bayesian networks. We then defined PRMs, and provided a brief overview of PRM inference, and PRM learning. We focused our discussion on Ben Ishak [2015]’s approach to evaluation of PRM structure learning algorithms. We explained their algorithm for generating datasets from random schemas, and pointed out some shortcomings of their approach. We also presented our contributions to overcome some of those shortcomings. To improve the random dataset generation process, we propose a novel algorithm for generating realistic relational skeletons, and adapt forward sampling and LABG sampling algorithms for relational context.

Recommender Systems: A Common Application of Relational Data

Contents

3.1	Introduction	42
3.2	Recommendation models and techniques	43
3.2.1	Recommendation data	44
3.2.2	Recommendation techniques	45
3.2.3	New developments	51
3.3	Evaluation of recommender systems	51
3.3.1	Evaluation approaches	52
3.3.2	Accuracy metrics	52
3.3.3	Other evaluation metrics	54
3.3.4	Benchmark datasets and evaluation tools	55
3.4	Challenges	57
3.5	Conclusion	59

3.1 Introduction

Recommender system (RS) is one of the most common applications that exploit the relational information from data. It aims at discovering potentially interesting items for users from a (usually large) collection of items. RSs are probably popularized by the growth of e-commerce websites, where the collection of products is usually large, and finding interesting products from such a big collection is a daunting task. Nowadays, we can find such systems in many websites that we visit frequently, e.g., YouTube, Facebook, IMDb, Amazon, Instagram, Spotify etc. Successful implementation of RSs can be found in many domains such as e-commerce, music/video/movie recommendations, social networking, online newspapers/blogs, hotel/flight reservation, online dating etc. Most RSs deal with data involving relationships between users and items, and make item-to-user recommendations. Recently, several studies have concentrated at user-to-user recommender systems, (e.g. online dating (Pizzato et al. [2010]), job recommendation (Hong et al. [2013]) etc.), which make use of relationships between users. It is clear that relational data underlies in both types of recommender systems. Due to the growth of services targeted at ever-increasing community of internet users, data involving relationships between users and items or between users and users are easily available in many domains. Consequently, research and development of RS has flourished, and has become an active field for over two decades. Recent reviews (Véras et al. [2015], Park et al. [2012], Jannach et al. [2012]) have shown that the interest in RSs is still increasing significantly, and will continue in the future too.

RSs are originated from the fields of *Information Retrieval (IR)*, *Machine Learning (ML)*, and *Decision Support System (DSS)* (Jannach et al. [2012]). From IR perspective, RSs help users to discover items relevant to the users' needs. From ML perspective, recommendation is the task of learning a model that can predict the user feedback on a specific item as accurately as possible. RSs can be considered as a DSS that helps users compare the items and decide which one(s) to choose. Though RSs can be viewed from different perspectives, the basic principle underlying RSs is to compute a relevance score that gives how relevant an item could be to a user, given a set of users (or user models) and a set of items. User models can be users' ratings, preferences, demographics, situational context, tags defined by users etc. whereas items can be anything depending on the domain, e.g. music, movies, books, news, cities, restaurants, jobs, products etc. or even users themselves.

In this thesis, we focus only on classical RSs, which deal with relationships between users and items. Such systems basically involve three types of objects – *items* to suggest, *users* who made some kind of interactions with the items and who will receive recommendations, and *transactions* (or interactions) between the users and the items, e.g. users' ratings, buying actions etc. A traditional way to visualize this data is as a matrix (table), aka *User-Item matrix*, where each row represents a user, each column represents an item, and each cell represents the transaction between the user with the item, empty cells denote no transaction between the user and the item. Such matrix does not contain additional information (or attributes) about items and users. In the field of relational learning, the use of such matrix is discouraged, and instead, the data is treated in its relational form, usually as relational databases or graphs. This thesis is more concerned with relational databases.

This chapter is organized as follows. Section 3.2 will introduce recommender systems formally, discuss on the types of data used in recommender systems, and provide a brief overview of different types of recommendation techniques. Some recent advances

in the field of [RS](#) will also be discussed. Section [3.3](#) will present techniques for evaluating recommendation algorithms, and present some frequently used metrics, benchmark datasets and tools for assessing recommender systems. In Section [3.4](#), we will discuss about challenges that recommender systems often encounter.

3.2 Recommendation models and techniques

The fundamental task of a [RS](#) is to predict which items a specific user would find interesting, in other words, to filter items for a target user. This is done by predicting the utility (or relevance score) of the items that have not been discovered yet by the target user. Then, the task of recommending items to a user would be to extract the items that have high predicted utility.

Definition 13 *Recommender system (RS) (Adomavicius and Tuzhilin [2005])*

Let U and I be the sets of users and items respectively, and f be a utility function that measures the usefulness of an item i to a user u , i.e.

$$f : U \times I \rightarrow R$$

where R is a totally ordered set (e.g., 5-star rating, real numbers within a certain range etc.). Then, a recommender system (RS) is defined as a system that extracts a set of top- N items $i' \in I$ that maximize the utility of the user $u \in U$, i.e.

$$\text{top}N(u) = \{i' \mid i' = \arg \max_{i \in I}^N f(u, i)\}$$

Sometimes, this type of recommender system is also referred to as a 2D recommender system. ■

In many recommender systems, the utility of an item is usually represented by a rating, which indicates a particular user's preference over that item. For example, in a movie recommender system like IMDb, users can rate movies in a scale of 1 to 10, in YouTube, users can like or dislike videos to show how useful the items were to them. However, in general, the utility can be any arbitrary function.

In the simplest case, $U \times I$ is a matrix, called *user-item matrix* (see Figure [3.1a](#)), where each cell indicates how a particular user rated a particular item, and f predicts the value for the cells where the rating value is missing. This kind of data has been widely used in many recommender systems, particularly the ones that use *collaborative filtering* approach (explained in Section [3.2.2](#)). Even though it is very natural to define each element of the user space U with a *profile* that includes various characteristics of users (e.g., age, gender, occupation etc.), the recommender systems that use only this matrix ignore such profile. Similarly, such systems do not exploit various attributes of items as well. A substantial amount of research has been performed to make use of user profiles and item characteristics for improving recommendations. This has led to different types of recommendation methods, which will be discussed in Section [3.2.2](#).

Recent studies attempt to go beyond user profiles and item features, and utilize contextual information to make recommendations for different situations. *Context* can be any circumstances that can affect recommendations, such as the user's location, emotional status or intention for making purchase (e.g., buying something as a gift or for improving skills etc.), time of the day, period of the year (e.g., high/low touristic

seasons), weather conditions and so on. Recommender systems that extend traditional recommender systems with contextual information are referred to as *Context-aware Recommender System (CARS)*. A typical case of CARS is location-based RS, which exploits geographical context of objects in the system. We will discuss on such type of RS in Chapter 6.

Definition 14 *Context-aware recommender system (CARS) (Ricci [2014], Adomavicius and Tuzhilin [2011])*

Let U and I be the sets of users and items respectively, C be the set of possible situations under which the items can be experienced (i.e. contextual information associated with the application), and f be a utility function that measures the usefulness of an item i to a user u in the situation c , i.e.

$$f : U \times I \times C \rightarrow R$$

where R is a totally ordered set (e.g., 5-star rating, real numbers within a certain range etc.). Then, a context-aware recommender system (RS) is defined as a system that extracts a set of top- N items $i' \in I$ that maximize the utility of the user $u \in U$ in the situation $c \in C$, i.e.

$$\text{top}N(u, c) = \{i' \mid i' = \arg \max_{i \in I}^N f(u, i, c)\}$$

From these definitions, we can identify two fundamental aspects of a recommender system: (1) the type of data available in the system, and (2) the recommendation technique used to predict the usefulness of items. In the following, we will discuss on these topics.

3.2.1 Recommendation data

We recognize two types of data used in RSs: tensor, and relational data.

Tensor A *tensor* is a multidimensional or multimode array. In 2D recommender systems, users' ratings for items are represented in the form of an $n_u \times n_i$ matrix¹ where n_u is the number of users and n_i is the number of items in the system. This matrix, commonly referred to as user-item matrix, provides the basis for collaborative filtering (discussed in Section 3.2.2), the most widely used recommendation technique so far. An example of a user-item matrix is shown in Figure 3.1a. In CARS, a 3-dimensional array is often used, where users, items and contextual information are the three dimensions of the tensor.

Relational data Relational data for recommender systems organize data as sets of objects of different types (users, items and ratings), where each object is defined by a set of attributes. Usually, user objects are described by users' demographic information, and item objects by various features/characteristics of the items. However, several recommendation methods attempt to enhance these objects by adding extra information (e.g., hierarchical categories of items, keywords describing items, more objects of different types related to items/users etc.) from different data sources. In

1. A matrix is a 2-mode tensor.

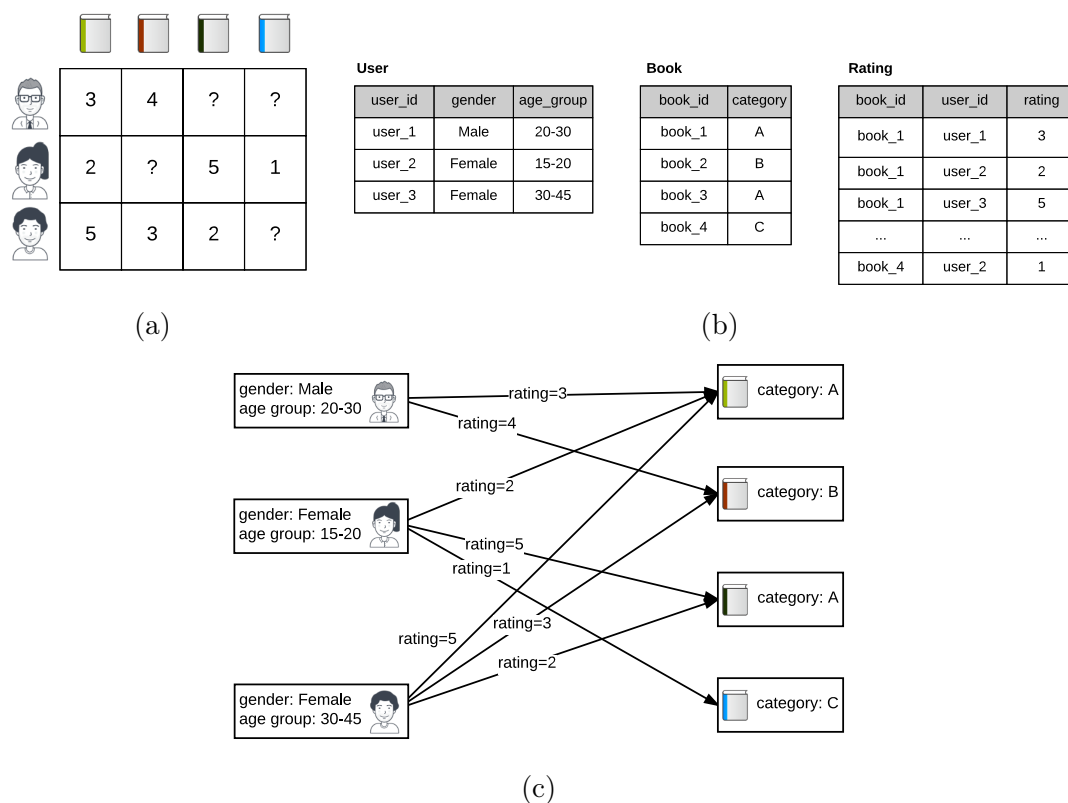


Figure 3.1 – ²Data representations commonly used in recommender systems: (a) a user-item matrix, (b) a relational database, and (c) a graph

CARS, contextual information may appear as additional attributes of users/items/ratings or as objects of different types. Relational databases, and graphs are commonly used for storing such relational data. Examples are shown in Figure 3.1. As mentioned in Chapter 2, this thesis is concerned only with relational data stored in relational databases.

3.2.2 Recommendation techniques

While representation of data is an important aspect for recommender systems, the heart of a recommender system is the underlying recommendation technique, which is responsible for predicting the usefulness of items for users. The algorithms used to perform such predictions are classically divided into five categories (Bobadilla et al. [2013], Adomavicius and Tuzhilin [2005]): *Collaborative Filtering*, *Content-based*, *Demographics-based*, *Knowledge-based* and *Hybrid* algorithms. The first four recommendation approaches are depicted in Figure 3.2.

Collaborative filtering

Collaborative Filtering (CF) is considered to be the most popular and widely implemented technique in RS (Ricci et al. [2011], Ekstrand et al. [2011]). This approach

2. User icons by Users Insights (<https://www.iconfinder.com/UsersInsights>) are licensed under CC BY 3.0, and book icons by Snip Master (<https://www.iconfinder.com/snipicons>) are licensed under CC BY-NC 3.0 / Changed color

makes recommendations based on the user-item matrix. CF can be performed in two ways: *user-based CF*, and *item-based CF* (Ekstrand et al. [2011]).

The fundamental assumption behind user-based CF is that similar users show similar behaviors (or preferences). This concept is illustrated in Figure 3.2a. In this figure, based on the information that all three users like the same books (at the top), CF assumes that all of them have similar taste for books, and recommends to the second user another book (bottom) liked by the first and the third users. User-based CF is carried out as follows. First of all, the neighbors of the target user (i.e. the users whose rating history is similar to the current user) are identified. To find the neighborhood of a user, a function is required to compute similarity between users. *Pearson's correlation coefficient*, and *cosine similarity* are two commonly used similarity measures. Once the neighbors are identified, items that the neighbors have rated but unknown to the target user are filtered, and the neighbors' ratings on those items are used to predict how the target user will rate them. Then, based on the predicted ratings, a list of items for recommendation is constructed. This is usually done by selecting the N items that have the highest rating.

Item-based CF (Sarwar et al. [2001]) is similar to user-based CF except that item-based CF deals with the neighborhood of items instead of that of users. It uses similarities between the rating patterns of items. Similarity functions used for user-based CF can be applied for item-based CF. Item-based CF differs from content-based filtering in that the former approach deduces similarity between items from transaction history instead of characteristics of items.

Su and Khoshgoftaar [2009] have categorized CF algorithms into three classes – *Memory-based*, *Model-based* and *Hybrid CF* algorithms. Memory-based CF algorithms use the complete set of data to make recommendations. These algorithms iterate through the entire set of up-to-date transaction list to discover neighbors of the target user and predict items that may interest the user. The advantage of this technique is that even the last piece of information in the system is taken into consideration while making a recommendation. However, it suffers the *scalability* problem. On the other hand, model-based approach deals with the recommendation problem by first building a model from the observed data and then using this model to make predictions. The models incorporate factors that help the system predict users' future interactions. Because building the model is done offline, this approach can build scalable recommender systems. However, recent information may not be taken into account by such models due to the offline processing. Hybrid CF algorithms combine different CF techniques to build systems that perform better than the individual techniques.

Content-based filtering

Content-based filtering (Pazzani and Billsus [2007], Lops et al. [2011]) considers items' characteristics and the target users' past behavior to make recommendations. It assumes that items with similar features will receive similar ratings by the same user. For example, if a user liked an action movie in the past, the user might like another action movie. Similarity between items can be calculated using the cosine similarity measure or using predictive models such as Bayesian classifiers, decision trees etc. Figure 3.2c shows content-based filtering approach.

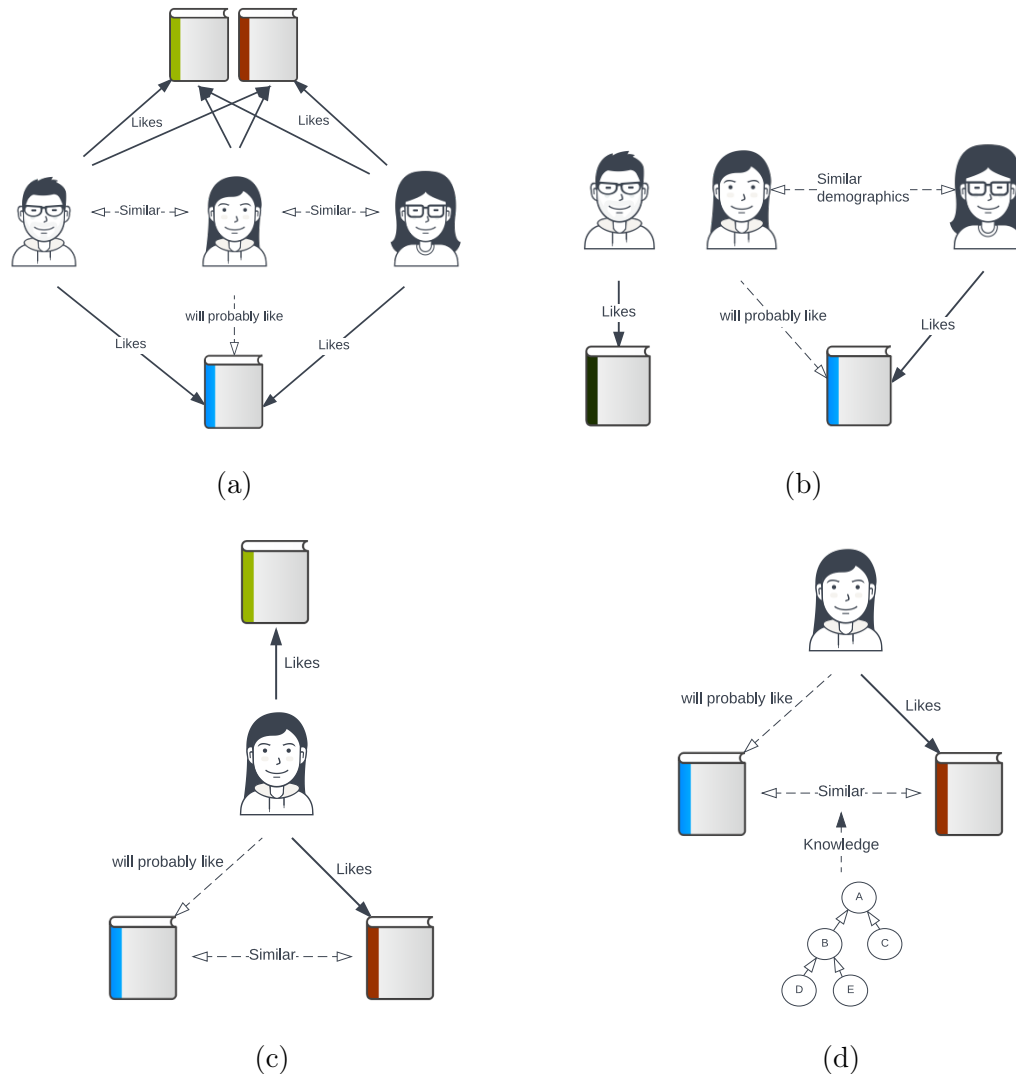


Figure 3.2 – ³Recommendation approaches: (a) Collaborative filtering, (b) Demographics-based filtering, (c) Content-based filtering, and (d) Knowledge-based filtering

Demographics-based filtering

Demographics-based filtering makes recommendations based on users' demographic information such as age, gender, occupation etc. It assumes that users' interests can depend on their demographics. For instance, animation movies might be popular among users of certain age group whereas thriller movies might be interesting for users of different age group. Figure 3.2b depicts this approach.

Knowledge-based filtering

Knowledge-based filtering algorithms aims at enhancing recommendation quality by adding domain-specific knowledge about how items might be relevant for the target user. Knowledge can be in the form of a query (i.e. the set of preferred features of a product), a case in a case-based reasoning system, an ontology, additional information

3. User icons by Users Insights (<https://www.iconfinder.com/UsersInsights>) are licensed under CC BY 3.0, and book icons by Snip Master (<https://www.iconfinder.com/snipicons>) are licensed under CC BY-NC 3.0 / Changed color

collected from external sources etc. An example of a knowledge-based RS that uses an ontology to adapt item similarity metrics is shown in Figure 3.2d.

Hybrid approaches

Hybrid approach combines multiple recommendation techniques to improve recommendation performance. Burke [2007] and Jannach and Friedrich [2013] have presented different ways for hybridization. A simple way to design hybrid RSs is to combine the results of multiple recommendation systems (see Figures 3.3a and 3.3b). Linear weights, for example, are commonly used to combine results numerically (Gao et al. [2007], Ye et al. [2011]). Alternatively, the recommendation lists obtained from different recommender systems can be simply merged while presenting the recommendations (Smyth and Cotter [2000], Barragáns-Martínez et al. [2010]). Another approach to hybridization is to switch between different recommendation systems depending on some selection criteria, as shown in Figure 3.3c. This can also be achieved by properly weighting the outcome of the recommender systems such that the hybrid system presents the outcomes of only one recommender at a time depending on the weights. Figure 3.3d shows another hybridization technique where recommender systems are cascaded so that output of one system would be the input of another system. Such design can be employed when a recommender system, which produces good recommendations, is too expensive to be applied on the complete dataset. The idea is to then exclude or shortlist items using weaker recommender systems which will provide a refined set of candidate items to the stronger recommenders. All of these techniques involve multiple recommender systems. Burke [2007] have also presented some techniques⁴ for producing hybrid recommendations from a single recommender system. These techniques include *feature combination*, where external knowledge is used to derive additional information of features that might be useful for recommendation, and *feature augmentation*, where new features of items are generated by using the recommendation logic of the contributing domain (e.g., Melville et al. [2002]). These techniques are illustrated in Figure 3.3e.

Though this conventional taxonomy of recommendation techniques are still widely accepted, some researchers have proposed categorizations of recommender systems from different perspectives or for specific domains. A recent taxonomy by Pu et al. [2012], for example, broadly classifies recommender systems into two categories based on the way systems gather and build user preference profiles: *preference-based* recommenders, and *behavior-based* recommenders. In the following, we will provide an overview of these categories.

Preference-based recommenders

Preference-based recommenders are based on the preferences explicitly stated by the users. In such systems, users actively state their preferences and provide feedback. Pu et al. [2012] have further divided such recommenders into three categories: *rating-based* systems, *feature-based* systems, and *personality-based* systems.

Rating-based recommenders These are classical recommender systems, where users express their preferences over items by the means of binary or multi-scale scores (ratings, likes/dislikes etc.). Traditional recommender systems let their users rate only

4. Jannach and Friedrich [2013] refer to them as *monolithic* techniques.

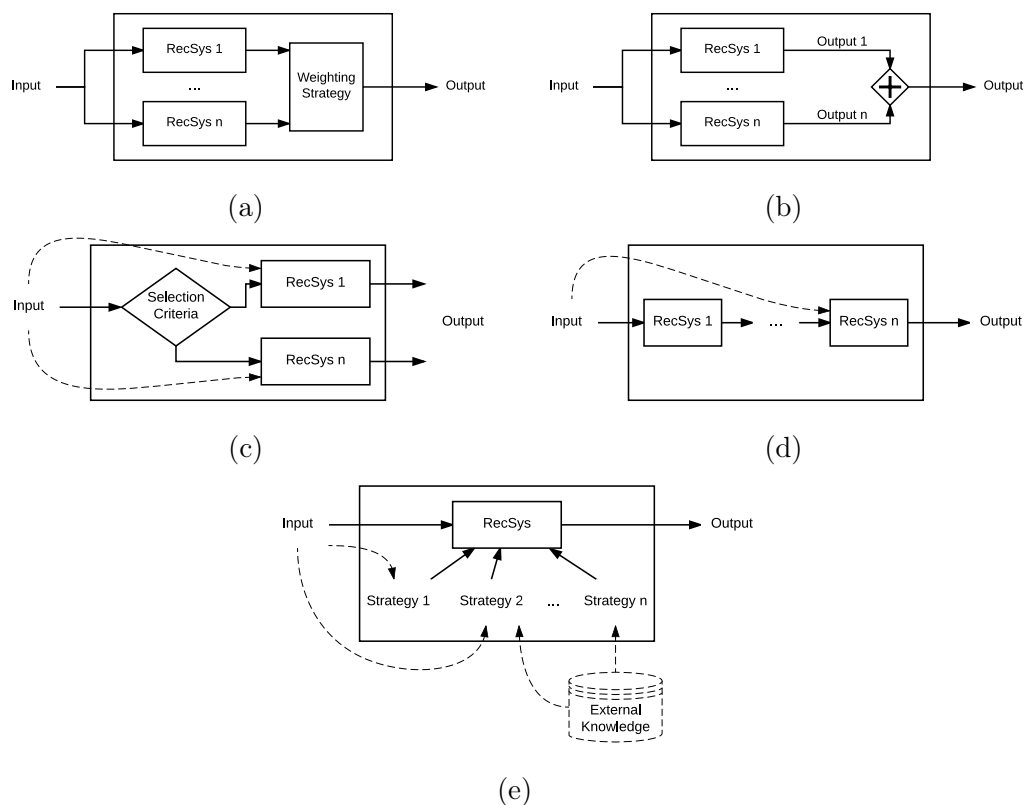


Figure 3.3 – Hybridization techniques (Burke [2007], Jannach and Friedrich [2013]): (a) Weighting the outputs, (b) Switching between recommendation modules, (c) Combining/merging the outputs, (d) Cascading, (e) Feature augmentation/combination (monolithic approach).

on a single aspect of items only. Ricci et al. [2015] have used the term *single-rating recommenders* for such systems. *Multi-criteria rating recommenders* consider rating on items' detailed attributes (Adomavicius et al. [2011], Nilashi et al. [2014]). For example, booking.com⁵ asks users to provide ratings for location, cleanliness, hospitality etc. of the booked hotels, and TripAdvisor⁶ let users to provide reviews on seat comfort, leg room, value for money, cleanliness etc. of airlines.

Feature-based recommenders Feature-based recommenders are similar to classical content-based systems but the difference is that feature-based recommenders allow users to express their preferences on specific item features and suggest items to the users based on the match between the users' preferences about items' attributes and the available items. These systems aim at capturing ephemeral preferences of the users and, thus, are suitable for recommending products infrequently purchased by users, perhaps due to a significant financial commitments, (e.g., real-estate, laptops, digital cameras, flights, hotel booking etc.). Pu et al. [2012] have further classified this type of recommenders into four categories: *case-based*, *utility-based*, *knowledge-based*, and *critiquing-based* systems.

Case-based recommenders (Smyth [2007]) use case-base / database of past problem solving experiences. Items are represented as cases, which are structured representations of item features, and recommendations are generated by retrieving those cases

5. <https://www.booking.com>

6. <https://www.tripadvisor.com>

that are most similar to a user's query or profile. As they rely on structured representation of items' features, they are suitable for the systems where items' detailed feature descriptions are readily available. Due to the structured representation, sophisticated similarity measures can be applied to find similar items/cases for recommendations.

Utility-based recommenders make recommendations based on the utility of items for users derived from their (explicit or implicit) preferences for a set of attributes. They use features of items as background data, elicit utility functions over items from users to describe user preferences, and apply the function to determine the rank of items for a user (Huang [2011]). The central problem of this type of system is creating the utility function. Most recommenders of this type apply concepts from *Multi-Attribute Utility Theory (MAUT)* or *Multi-criteria Decision Making (MCDM)* to derive utility functions.

Knowledge-based recommenders are similar to knowledge-based recommenders from the classical taxonomy. This type of recommenders is a variant of case-based recommenders that includes functional knowledge on how a particular item meets a particular user need, and can therefore reason about the relationship between a need and a possible recommendation. The knowledge can be in many forms.

Critiquing-based recommenders simulate an artificial salesperson that recommends options based on users' current preferences and continuously adapts the recommendations by observing users' feedback in the form of critiques such as "I would like something cheaper" or "I would like something newer". This is an iterative approach where at each iteration, users provide feedback about the recommended items and the system improves the recommendation list. This process of critiquing and improving recommendations continue as long as the users do not find the best item(s). This kind of system is useful in the situations where users tend to have hidden preferences about items or they might not be clear about their preferences at the beginning. For example, when looking for an apartment to rent, a user might have few preferences, e.g., her preferred number of bedrooms, or preferred location etc., but as she explores the first recommendations provided by the system, she may discover more features of apartments (e.g., construction year of the apartment, or whether there is an elevator etc.), so she may add a new preference (e.g prefer new apartments to old ones) which helps the system refine the recommendation list. Chen and Pu [2012] have presented a detailed study of critiquing-based recommenders.

Personality-based recommenders Inspired from psychological studies which suggest that there is a significant connection between personality and people's tastes and interests (Hu and Pu [2010]), personality-based recommenders aim at making personalized recommendations from users' personality. Such systems may acquire users' personality explicitly (e.g., through questionnaires) or implicitly (e.g., by observing users' behaviors). A typical case is affective recommender systems that are associated with human behavior, human factors, mood, senses, emotions, facial expressions, body gesture and physiological with *Human-Computer Interaction (HCI)* (Katarya and Verma [2016]).

Behavior-based recommenders

In behavior-based recommenders, users do not state their preferences for items explicitly, so recommendations are made based on users' behavior or actions on the systems. For example, visiting some pages or buying something can infer that the

users are interested in those items.

3.2.3 New developments

Recommender systems discussed so far deal with the situations which seek one-sided items-to-people recommendations, i.e. only users receive recommendations about items and not the other way round. Recently, there is a growing interest in people-to-people (aka reciprocal) recommenders, where the recommendations must address the needs of both parties. Examples of such recommenders include online dating ([Pizzato et al. \[2010\]](#)), and job recommender systems ([Hong et al. \[2013\]](#)). Meanwhile, many researchers ([Brun et al. \[2010\]](#), [Liu et al. \[2009\]](#)) advocate the use of users' ordinal preferences about items (i.e., relative orders between items) instead of their numerical preferences (such as ratings) for making recommendations. They argue that it is easier for users to provide ordinal preferences (e.g., through pair-wise comparison of items) than to provide numerical preferences as the numerical scale is usually insufficient to express their opinion, and that ordinal preferences often tend to be stable compared to the numerical ones ([Jones et al. \[2011\]](#)). RS community is also attracted towards tag-based recommendations ([Gupta et al. \[2010\]](#)) as a result of the increased use of tags in a variety of systems (e.g., social networks, photo sharing, social bookmarking, news etc.) for various purposes such as expressing opinions, discovering, organizing, or understanding entities etc. Two primary tasks for such systems are: exploiting tags to provide enhanced user/item recommendations (e.g., [Sen et al. \[2009\]](#)), and recommending tags to users (e.g., [Lops et al. \[2013\]](#)). Another line of research in the field of RS is oriented towards combining characteristics of ubiquitous systems and recommender systems to enhance the experience of personalized recommendations ([Mettouris and Papadopoulos \[2014\]](#)). Group recommender systems ([Masthoff \[2011\]](#)), which recommend items to a group of users instead of a single user, have also been frequently studied in the RS community. Community-based or social recommender systems have gained popularity due to the rise of social networks. They make recommendations based on the preferences of users' friends ([Ricci et al. \[2015\]](#)). The assumption behind this type of recommenders is that users tend to rely on recommendations from their friends or acquaintances rather than anonymous persons with similar taste. Cross-domain recommender systems ([Fernández-Tobías et al. \[2012\]](#)) have emerged as a solution to reduce users' involvement in building their profile for recommendations. These systems collect user profiles from other domains, and thus eliminate the need to ask users to provide their details explicitly.

3.3 Evaluation of recommender systems

Growing use of recommender systems in various domains has resulted in numerous recommendation algorithms. As choosing an appropriate recommendation method for a specific system requires comparison of several algorithms, the RS research community has been active in the study of effective evaluation techniques (e.g., [Said and Bellogín \[2014a\]](#), [Beel et al. \[2013\]](#), [Meyer et al. \[2012\]](#), [Shani and Gunawardana \[2011\]](#), [Herlocker et al. \[2004\]](#)). In this section, we will present commonly used techniques and metrics for evaluating recommender systems.

3.3.1 Evaluation approaches

Different classifications of **RS** evaluation methods can be found in the literature. Many researchers evaluate recommender systems using off-line analysis, live user experimental methods (aka on-line evaluation) or a combination of these two approaches (Herlocker et al. [2004]).

Off-line evaluation

Off-line evaluation is a widely used technique for evaluating recommender systems. This approach measures the accuracy of recommendations made by a **RS** without the involvement of actual users. It uses an existing data as ground truth, and expects the results of a **RS** to match the ground truth. To evaluate a **RS**, the dataset is split into a training set and a test set. Then, using the training set, the **RS** tries to predict the ratings in the test set. The predicted ratings are then compared to the actual ratings in the test set. Several metrics (as explained in Section 3.3.2) can be computed to analyze the results.

On-line evaluation

On-line evaluation measures the acceptance rate of recommendations of a **RS** by real users. In this approach, actual users interact with a **RS**, and receive recommendations, which they may or may not accept. Recommendation quality can be then measured by asking users for their feedback or by monitoring the users' behavior during their interaction with the system (for example, by observing if they click on the recommended items or not). On-line evaluations are usually expensive and time-consuming.

3.3.2 Accuracy metrics

Among various metrics for evaluating recommender systems, accuracy metrics are the ones that are widely adopted. These metrics measure how close the predictions made by the recommender systems are to the ground truth. Herlocker et al. [2004] have identified three types of accuracy metrics: classification accuracy metrics, prediction accuracy metrics, and rank accuracy metrics. In the following, we will discuss some commonly used accuracy metrics.

Classification accuracy metrics

As **RSs** are originated from the field of **IR**, classic **IR** metrics for measuring classification accuracy, such as Precision, Recall, and their harmonic mean F1-score, are popular for evaluating recommendation algorithms (Jannach et al. [2012]). These metrics measure to what extent a **RS** is able to correctly classify items as interesting or not.

Precision is defined as the ratio of the number of relevant recommended items to the number of selected items for recommendation.

$$Precision = \frac{\text{Number of relevant recommended items}}{\text{Number of recommended items}} \quad (3.1)$$

Recall is defined as the ratio of the number of relevant recommended items to the number of relevant items in the test set.

$$Recall = \frac{\text{Number of relevant recommended items}}{\text{Number of relevant items}} \quad (3.2)$$

Precision measures how good the recommendations are whereas recall measures how well the algorithm could discover interesting items. It is desirable to have higher values of these metrics. However, these metrics are contradictory to each other. Increasing the number of recommended items increases recall but decreases precision. Their harmonic mean, F-measure (or F-score) is a commonly used metrics that provide a compromise between these two metrics.

$$F_\alpha = (1 + \alpha) \frac{Precision \times Recall}{\alpha \times Precision + Recall} \quad (3.3)$$

Prediction accuracy metrics

Prediction accuracy metrics measure how well users' actual ratings match the ratings predicted by the recommender system. *Mean Absolute Error (MAE)* is a widely used prediction accuracy. As the name suggests, MAE is the average of absolute difference between the actual and the predicted ratings. Let p , and r be the predicted, and the actual ratings respectively, and N be the number of recommended items. Then, MAE is defined as follows.

$$MAE = \frac{1}{N} \sum_{i=1}^N |p_i - r_i| \quad (3.4)$$

Variations of MAE are also commonly used, such as *Normalized Mean Absolute Error (NMAE)*, and *Root Mean Squared Error (RMSE)*. NMAE is the normalized version of MAE, and is defined as follows.

$$NMAE = \frac{1}{r_{max} - r_{min}} MAE \quad (3.5)$$

RMSE emphasizes large errors by squaring each individual error.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (p_i - r_i)^2} \quad (3.6)$$

Rank accuracy measures

These metrics measure to what extent the ranking proposed by the recommender system differ from the actual ranking, assuming that the order of the items in the recommendation list is important. Correlation measures, such as *Spearman's correlation coefficient*, are commonly used to determine the similarity between two lists of items. Spearman's correlation coefficient is given by:

$$\rho = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n \cdot stdev(x) \cdot stdev(y)} \quad (3.7)$$

where x_i , and y_i are the ranks of the item i in the user's ranked list \mathbf{x} , and those in the recommendation list \mathbf{y} respectively, and n is the number of items.

Half-life utility metric is another rank accuracy metric that is applicable in scenarios where users are unlikely to browse very deeply into the ranked list. This metric estimates the expected utility of a ranked list to a user u , and is given by

$$R_u = \sum_i \frac{\max(r_{u,i} - d, 0)}{2^{(i-1)/(\alpha-1)}} \quad (3.8)$$

where $r_{u,i}$ is the rating of the user u on the item j of the ranked list, d is the default rating (which is usually neutral or slightly negative), and α , aka *half-life*, is the rank of an item on the list that has a 50% chance of being viewed.

The overall half-life utility score for a dataset across all users (R) is then computed as:

$$R = 100 \frac{\sum_u R_u}{\sum_u R_u^{max}} \quad (3.9)$$

where R_u^{max} is the maximum achievable utility if the system ranked the items in the exact order that the user did.

Edit distance, widely used in IR to compare strings by counting the number of operations needed to transform one string to another, can also be used to evaluate how well recommender systems can reproduce users' ranked list. The idea is to consider each ranked list as a sequence of characters (or a string), and compare the lists in the same way as done for strings.

3.3.3 Other evaluation metrics

Despite being widely used, there is an increasing consensus in the RS community about the insufficiency of accuracy metrics to assess practical effectiveness, and added-value of recommendations (McNee et al. [2006]). This concern has led to the proposition of several other evaluation metrics for measuring various aspects of recommendations. In particular, coverage, diversity, novelty, and serendipity are often regarded as important aspects of recommendations.

Coverage measure the percentage of items for which recommendations can be made by the system. Low coverage means that many items will remain unexplored, and, hence, such systems will be less valuable to users.

Always recommending similar products can make the system useless and less interesting to users. Thus, diversifying the recommendations is commonly considered as an important task for improving the utility of recommender systems. Ziegler et al. [2005] have proposed to measure the *diversity* of a recommendation list in terms of *intra-list similarity*. Higher intra-list similarity indicates that the recommendations are less diverse. A similar notion that has gained the attraction of the RS community is *novelty*. While diversity refers to how different the items are with respect to each other in the recommendation list, novelty indicates how different items recommended to users are from the one that they have been seen in the past. Higher novelty can indicate higher diversity.

Serendipity is another important dimension of recommendations, which measures the degree to which the recommended items are relevant to users and also surprising at the same time. According to Herlocker et al. [2004], a good serendipity metric is the one that would look at the way the recommendations are broadening the user's interests over time. Ge et al. [2010] have formulated a metric for measuring serendipity as a function of the usefulness of the unexpected recommendations as follows:

$$SRDP = \frac{\sum_{i=1}^N u(RS_i)}{N} \quad (3.10)$$

where $u(RS_i) \in \{0, 1\}$ indicates whether the recommended item i is useful or not (with 1 indicating it is useful), and N is the number of unexpected items.

More recently, *revenue maximization* (Azaria et al. [2013]) is also considered as a primary goal of a recommender system. So, recommender systems are also be evaluated based on how well they yield the expected revenue.

Besides the metrics presented here, there are several other evaluation metrics reported in the literature. For more metrics, and further reading, we refer readers to the articles Herlocker et al. [2004], and Shani and Gunawardana [2011].

3.3.4 Benchmark datasets and evaluation tools

Several datasets used for recommendation research purposes are available online. Some of them are listed in Table 3.1. MovieLens datasets are probably the most used datasets in recommendation research. These datasets contain users' ratings on different movies, and are available in different sizes. Originally collected from the MovieLens⁷ website, these datasets have been enhanced by gathering additional information from other sources, such as Twitter⁸, IMDb⁹, and RottenTomatoes¹⁰. Several datasets are made available to public as a part of some competitions, such as RecSys challenges¹¹, KDD cup¹², Kaggle competitions¹³ etc.

Such public datasets can be considered as benchmark datasets to evaluate new recommendation algorithms. Recommendation libraries like LibRec²¹, and MyMediaLite²², have even published benchmarks results for some common recommendation methods on some public datasets. These results (as well as those reported in many research articles) can serve as baselines for assessing new recommendation algorithms. However, it should be noted that sometimes best prediction results may not be important but other aspects of recommendations, such as the ones discussed in Section 3.3.3.

Active development of tools for building and evaluating recommender systems can be observed in the past years. LensKit²³, LibRec, MyMediaLite, RankSys²⁴, TagRec²⁵,

7. <http://grouplens.org/datasets/movielens/>

8. <https://twitter.com/>

9. <http://www.imdb.com/>

10. <https://www.rottentomatoes.com/>

11. <https://recsys.acm.org/honors-and-awards/challenges/>

12. <http://www.kdd.org/kdd-cup>

13. <https://www.kaggle.com/competitions>

14. <http://www.last.fm/>

15. <http://del.icio.us/>

16. <https://sites.google.com/site/yangdingqi/home/foursquare-dataset>

17. <http://www.yelp.com/>

18. <http://www.macle.nl/tud/LT/>

19. <http://www.comp.nus.edu.sg/~sugiyama/SchPaperRecData.html>

20. <http://www.libimseti.cz/>

21. <http://www.librec.net/example.html>

22. <http://www.mymedialite.net/examples/datasets.html>

23. <http://lenskit.org>

24. <http://ranksys.org/>

25. <https://github.com/learning-layers/TagRec>

Table 3.1 – Some datasets used for recommendation systems research purposes

Dataset	Recommendation Item	Available Information	Rating Scale
MovieLens	Movie	Users; Movies; Ratings	5-star scale (integers only)
Jester (Goldberg et al. [2001])	Joke	User-item matrix (<i>User ID, Item ID, Rating</i>)	[-10, +10] (continuous ratings)
Movie Tweetings (Dooms et al. [2013])	Movie	Users; Movies; Ratings	[0, 10]
Last.fm ¹⁴	Music/Artist	User profile; Artists; Play count	Play count
HetRec2011 MovieLens + IMDb/Rotten Tomatoes (Cantador et al. [2011])	Movie	Movies; Countries; Locations; Directors; Actors; Genres; Tags; Users; Users' ratings; Tag assignments	(0, 5]
HetRec2011 Last.fm (Cantador et al. [2011])	Songs/ Soundtracks	Users; Artists; Tags; Tag assignments; Users' friends; Users' listening history	Play count
HetRec2011 Delicious bookmarks ¹⁵ (Cantador et al. [2011])	URLs	Users; Users' contacts; Bookmarks; Tags; Tag assignments	Tag weight (The number of times the tags were assigned to each URL)
FourSquare ¹⁶	<i>Point-of-interest (POI)</i>	Users; POIs; Check-ins; Tips; Tags	Existence of check-ins
Yelp ¹⁷	Businesses	Businesses (w/ geographical info); Users; Reviews	1 to 5 scale (integers only)
Book crossing (Ziegler et al. [2005])	Book	Users, Books, Ratings	0 to 10 scale (integers only)
LibraryThing ¹⁸	Books/Users	Users; Users' friends; Tags; Tag assignments	1 to 10 scale (integers only)
Restaurant and consumer dataset (Vargas-Govea et al. [2011])	Restaurants	Consumers; Restaurants (w/ geographical info); Ratings	{0, 1, 2}
Scholarly Paper Recommendation Datasets ¹⁹	Scholarly papers	Researchers; Research interests; Papers; Citations/References	Existence of citations
LibimSeTi ²⁰ (Brozovsky and Petricek [2007])	Users	Users; User-user ratings	1 to 10 scale (integers only)

mrec²⁶ by Mendeley²⁷, RiVal²⁸ (Said and Bellogín [2014b]), Idomaar²⁹ and LightFM³⁰ are some tools that provide a framework for evaluating recommender systems as well as implementations of several recommendation methods.

3.4 Challenges

Recommender systems often face several challenges which affect their performance, limit their applicability and hinder the expansion of the systems. Here we discuss some major challenges for developing recommender systems.

Scalability Recommender systems are often employed in large e-commerce websites where the systems aim at selling as much products as possible, and the more customers they can attract, the more revenue they will generate. Such systems need to keep up with the continuously growing users and items. Traditional recommendation algorithms, especially memory-based CF, are generally not scalable because they require large computational resources for large datasets. Several techniques such as dimensionality reduction (Billsus and Pazzani [1998], Sarwar et al. [2000], Koren et al. [2009], Ma et al. [2008]), model-based CF (Mobasher et al. [2006]) etc., and many hybrid recommender systems have emerged to address the scalability issue.

Data sparsity In practice, the set of items in recommender systems is usually very large. Thus, users interact with only a small subset of items in systems, and this leads to a sparse user-item matrix. This may significantly degrade the performance of recommendation algorithms that deal with the user-item matrix directly. Dimensionality reduction techniques, such as *Singular Value Decomposition (SVD)*, *Latent Semantic Indexing (LSI)*, *Principle Component Analysis (PCA)* etc., have been used to address data sparsity issue (Billsus and Pazzani [1998]). A number of works that combine content-based, demographics-based, and CF techniques have also been proposed to alleviate data sparsity issue (e.g., Melville et al. [2002], De Campos et al. [2010], Lika et al. [2014]). In social networks-based recommender systems, this issue is often handled by exploiting the trust information that can be extracted from transitive associations between users (e.g., Papagelis et al. [2005], Jamali and Ester [2009], Guo et al. [2014]).

Cold start There is usually insufficient information for making recommendations to new users because of the lack of their interactions with the system. In the same way, it is difficult to find potential users for new items that are not rated yet. Such situation is called *cold start* situation. New systems are often in even worse position because not only users and/or items are new but also the number of users (and/or items) is usually very low in these systems to build a good recommendation model out of the available data. Such new systems are often referred to as *cold systems*. Cold start problem may be a show-stopper for pure CF. Several techniques have been proposed to address this problem. Information about users and items are often exploited to handle this

26. <https://github.com/Mendeley/mrec>

27. <https://www.mendeley.com/>

28. <http://rival.recommenders.net/>

29. <http://rf.crowdrec.eu/>

30. <http://lyst.github.io/lightfm/docs/index.html>

issue (Schein et al. [2002]). To build user profiles, some systems ask users to provide their demographic information explicitly while others may collect their users' activities implicitly. For example, Lee et al. [2010] provide a preliminary list of recommended music to users to gather implicit information about users' behavior by tracking the music that are ignored, visited or purchased by the users. Baeza-Yates et al. [2015] tracks how long mobile apps are used to address the cold-start problem in their mobile app recommendation system. Domain-specific information can also provide important clues in tackling the cold-start problem. For example, Ahn [2008] use domain-specific concepts about proximity, impact, and popularity to enhance similarity computation. Numerous other techniques have been proposed to address cold-start problem. These mostly employ hybrid recommendation techniques and may be domain-specific solutions. As cold start is a typical case of data sparsity, solutions proposed to deal with data sparsity issue are applicable in cold start situation in many cases (e.g., Guo et al. [2014], Lika et al. [2014] etc.).

Preference Acquisition and Profiling Acquiring users' preferences and their profile is an important topic for developing RSs but several issues related to it are still unanswered. Users' preferences can be gathered explicitly by asking them state their preferences, or implicitly through their actions (e.g., click-through, purchase information etc.). It is easy to collect implicit user feedback, which can be considered as a form of positive feedback. However, in such data, there will be no negative feedback, all of them being either positive or missing. From missing data alone, it is difficult to conclude whether the users did not like the recommended items or missed them because of other influencing factors (e.g. poor user interface), or if they simply chose to ignore them. While developing a RS, it is quite challenging to analyze the trade-off between the cost of preference acquisition and recommendation refinement. Multi-rating RSs can provide refined recommendations from multiple criteria but require a significant level of user involvement compared to single-rating RSs.

Gathering user profiles required to address cold-start problem may be a difficult task because people may not tend to readily trust new systems and may avoid providing personal information as much as possible, which may be especially due to the frequently appearing news about security breach in many applications. Cross-domain RSs attempt at solving this issue by collecting user profiles from other domains and thus eliminating the need to ask users to provide their details explicitly. Personality-based recommenders aim at improving recommendations using personality information, such as mood/emotions, but these systems need to acquire such information in a non-intrusive way.

Interaction The way RSs interact with users can play a significant role in the success of the RSs. Thus, RSs need to properly guide users right from the preference acquisition process to the visualization of recommendations. Adding explanations to the recommendations can improve users' experience in understanding the recommendations. However, whether such explanations can influence users' decision is still not clear (Ricci et al. [2015]). Another important aspect about the interaction with RSs is to achieve a balance between novelty/diversity and accuracy. Whether to keep recommending items that the system can identify as good recommendations or to include novel/serendipitous items or diversified lists has been a challenging topic from a long time. Recently, several researchers (e.g., Vargas and Castells [2011], Zhang et al. [2016]) have shown interest in diversifying recommendations.

Miscellaneous As *RSs* are gaining popularity and being implemented in many different domains, new challenges are emerging steadily. For example, due to the increased use of mobile systems, it is now easier to collect contextual information and also to push recommendations proactively to users' mobile devices. Learning to make appropriate recommendations based on users' contextual information without overwhelming the users with irrelevant recommendations has been a major challenge for *RSs* nowadays.

There are still many domains where implementing *RSs* can be interesting but which are not much explored probably due to the underlying big risks. *RSs* can be observed to be widely adopted in recommendations of simple and inexpensive products like movies, music, books etc. More complex, expensive products or less frequently purchased items such as real estate, travels, financial investments etc. can be challenging for the application of recommender systems. For example, not recommending good travel options may make result in bad reputation of the system and/or decrease revenue as the users will stop using the system.

3.5 Conclusion

In this chapter, we introduced recommender systems. We gave a brief overview of two types of data used in recommender systems: tensor and relational data. We then discussed on different categories of recommendation algorithms. Following the classical taxonomy of recommendation techniques, we presented the following techniques: collaborative filtering, content-based, knowledge-based, demographics-based, and hybrid approaches. Among these approaches, collaborative filtering, which makes use of past interactions between users and items for making recommendations, is commonly adopted and widely studied. This technique is often used in conjunction with other recommendation techniques to provide hybrid solutions that can tackle different issues, such as scalability, data sparsity, and cold start situation. We also presented a more recent taxonomy that classifies recommender systems from the users' perspective into preference-based and behavior-based recommender systems. We then discussed on how to evaluate recommender systems, and presented some frequently used metrics, public datasets, and tools for evaluating recommendation methods. We also discussed on some important challenges, which are commonly encountered by recommender systems.



4

Using Probabilistic Relational Models for Recommendation

Contents

4.1	Introduction	62
4.2	Existing approaches	62
4.2.1	Collaborative Filtering using PRMs (Getoor and Sahami [1999])	62
4.2.2	A unified recommendation framework based on PRMs (Huang et al. [2004])	63
4.2.3	Hierarchical Probabilistic Relational Models (hPRM) (Newton and Greiner [2004])	64
4.2.4	Combining User Grade-based Collaborative Filtering and PRMs (UGCF-PRM) (Gao et al. [2007])	65
4.2.5	A RBN-based recommender system architecture (Ben Ishak et al. [2013])	65
4.3	Comparison and discussion	66
4.4	Conclusion	69

4.1 Introduction

We have discussed in Chapter 3 that data involved in recommender systems (RSs) are rich in relational information. Because of this relational nature of data in RSs, the relational learning community has shown a great interest in RSs since quite a long time. A plethora of recommendation methods have been proposed in the past decades. These methods have been applied in various domains of different sizes. To keep up with the ever-growing data, researchers are attracted towards scalable techniques for recommendations. As discussed in Section 3.4, model-based recommendation techniques (such as matrix factorization methods, probabilistic models and various other machine learning approaches) have become popular in recent years specially due to their capability to address the scalability issue. PRMs have also found their application as a model-based recommendation technique in this field. Using PRMs in recommender systems has, in fact, been a topic of research from the beginning of PRM formalism. Getoor and Sahami [1999] were the first ones to propose a PRM-based recommender system. They have been followed by several other researchers who have proposed different manners of integrating PRMs into the recommendation process. In this chapter, we will review some of such works, and present a comparative study about them.

This chapter is organized as follows. In Section 4.2, we will provide a brief description of some recommender systems that use PRMs and in Section 4.3, we will compare and discuss on those approaches.

4.2 Existing approaches

In this section, we will review five existing PRM-based recommendation approaches, and will explain different ways of integrating PRMs to the recommendation process present in the literature.

4.2.1 Collaborative Filtering using PRMs (Getoor and Sahami [1999])

The earliest approach to PRM-based recommender systems was proposed by Getoor and Sahami [1999]. It is based on Bayesian network-based collaborative filtering proposed by Ungar and Foster [1998] and Hofmann and Puzicha [1999]. The main idea behind these models is to cluster users and items separately, and make predictions based on the clusters instead of dealing with a large set of user-item matrix. Clustering is performed based on the history of users (or items) and their neighborhood. Ungar and Foster [1998] proposed the repeated clustering technique to find clusters of users and items, where firstly, users are clustered based on the items they like (or purchase) and items based on the users who like them, and in the subsequent phases, users are clustered based on the item clusters and items based on user clusters. In these models, we have latent variables C_{x^i} for each user $x^i \in \{x^1, x^2, \dots, x^m\}$ and C_{y^i} for each item $y^i \in \{y^1, y^2, \dots, y^n\}$, where m and n are the number of users and items respectively. For each user-item pair (x^i, y^j) , the existence of relation r^{ij} between user x^i and item y^j depends on their clusters. The model is depicted in Figure 4.1. Here, strong assumptions are made that each person (also each item) belongs to only one cluster and that every relation r^{ij} must have the same local probability model. These assumptions make it possible to represent this model compactly by PRMs, where every user and item are assigned to a class, and these classes determine the relation between

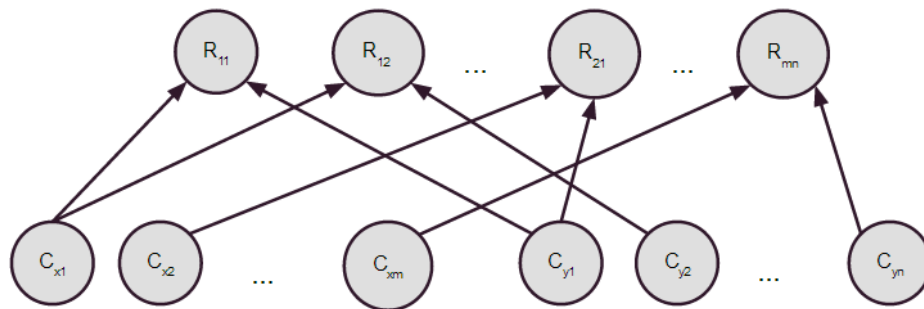


Figure 4.1 – A Bayesian Network for two-sided clustering (Getoor and Sahami [1999]). Here, C_{x^i} and C_{y^i} represent the cluster of a person x^i and that of an item y^i respectively, and R_{ij} denotes the relation between a user x^i and an item y^j .

the user and the item. With the use of PRMs, dependencies of other attributes of the entities can also be included in the model. For example, age, gender, profession etc. of users can determine their classes or vice versa and there could be dependencies between these attributes themselves or even with the attributes of items. This enables PRMs to make better predictions because the effect of many variables can be seen from a single model. This also enables PRMs to handle cold start problem. If a user has not liked / purchased any items yet, PRMs can still make recommendation for him based on the attributes of this user. The system, however, would be no more doing collaborative filtering in that case. It will rather be more like a demographic filtering system or some sort of hybrid system.

4.2.2 A unified recommendation framework based on PRMs (Huang et al. [2004])

Huang et al. [2004] have proposed a PRM-based recommendation framework that combines concepts from PRMs-EU and PRM structure learning. They address the recommendation task as the problem of predicting links between users and items, and show that PRMs-EU can be useful in such problems. Like in a PRM-EU, they introduce a boolean attribute ‘*Exists*’, which indicates whether the link between a user and an item exists or not. They employ the fundamental concept behind PRM structure learning (i.e. walking through the slot chains) to capture the information that are potentially interesting for recommendation. Then, they derive a partial dependency structure among these information for the ‘*Exists*’ attribute to build a recommendation model. Their method is essentially a hybrid approach because it is capable of combining different data patterns employed by content-based, demographic filtering, and collaborative filtering algorithms (e.g., users’ demographics for demographics-based recommendation, items’ characteristics for content-based recommendation, ratings made by users’ neighbors for collaborative filtering etc.).

The basic work-flow for their method is as follows. First, relational attributes are generated by walking through the slot chains in the relational schema starting from the target attribute (i.e., the ‘*Exists*’ attribute). They limit the length of slot chains in heuristic structure search algorithm (Friedman et al. [1999]) to create a finite attribute space, from which attributes that can be relevant for recommendation are filtered out as *Markov blanket* attributes of the target attribute. A Naïve Bayesian classifier is, then, built from the *Markov blanket* such that all *Markov blanket* attributes are independent

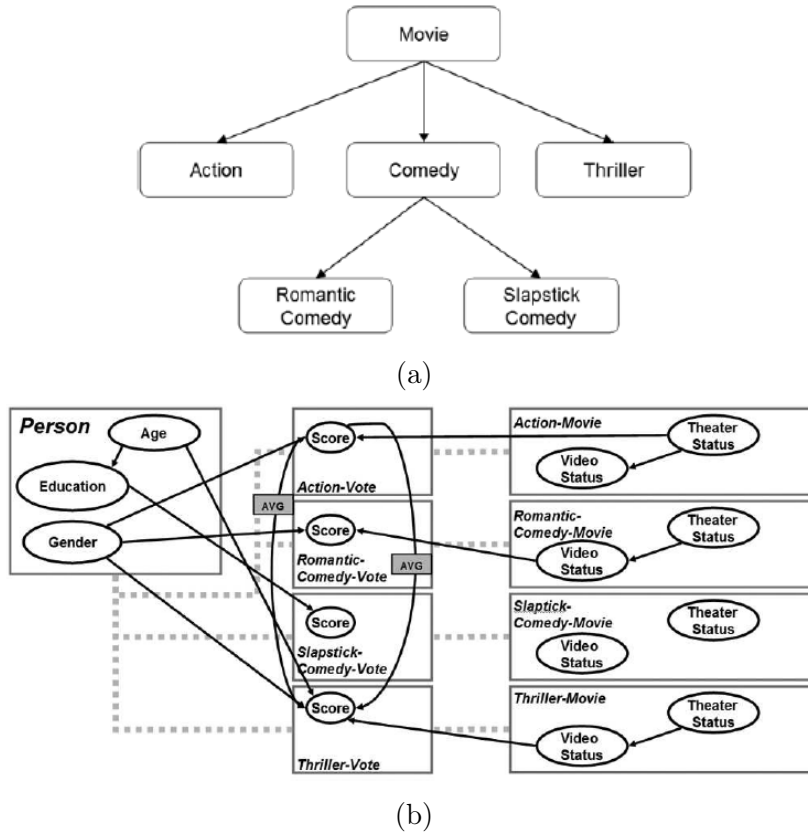


Figure 4.2 – (a) Hierarchy of Movie class and (b) Hierarchical Probabilistic Relational Model (hPRM) proposed by [Newton and Greiner](#)

of each other given the target attribute. This classifier is then used to recommend items. They have illustrated that long slot chains can capture interesting patterns and the use of such long slot chains along with aggregation and multiset operations can enhance the performance of recommender systems. [Perlich and Huang \[2005\]](#) have used this framework in the context of Customer Relationship Management (CRM).

4.2.3 Hierarchical Probabilistic Relational Models (hPRM) ([Newton and Greiner \[2004\]](#))

[Newton and Greiner \[2004\]](#) have proposed hPRM, inspired from [Getoor \[2001\]](#)'s PRM-CH. It aims at addressing the issue of cyclicity that usually appears in recommendation tasks. For instance, in the context of a movie-rating dataset, a user's rating on some movies may depend on ratings of the same person on some other movies. Because a rating depends on itself, this information cannot be expressed in a PRM due to the violation of the acyclicity constraint. [Newton and Greiner \[2004\]](#) address this issue by dividing the concerned class into hierarchical sub-classes, and defining the dependency structure using only leaves of the hierarchy. They propose greedy partitioning approach to learn the hierarchy. hPRM is learned and instantiated in the same way as a standard PRM. Inference is performed on the unrolled hPRM. Figure 4.2 depicts the hPRM, which they applied on a movie-rating context, where the rating of one type of movies can depend on the (average) rating of different type of movies. They divided the class Movie hierarchically based on the genre of movies as shown in Figure 4.2a, and then derived a hPRM using the leaves of the hierarchy as shown in Figure 4.2b.

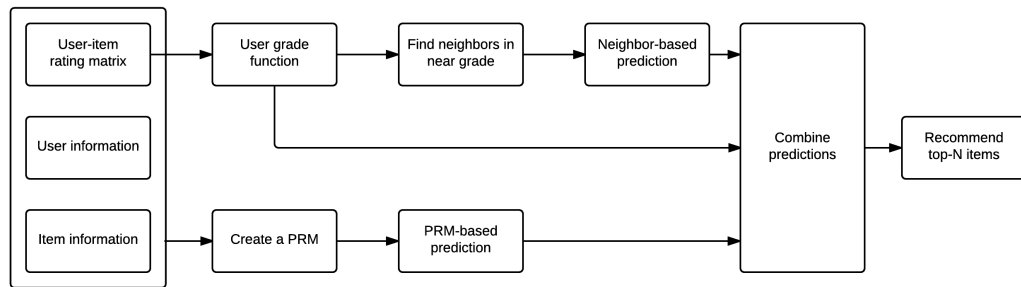


Figure 4.3 – Overview of the recommender approach proposed by Gao et al. [2007]

4.2.4 Combining User Grade-based Collaborative Filtering and PRMs (UGCF-PRM) (Gao et al. [2007])

Gao et al. [2007] have proposed a hybrid method for recommendation that combines collaborative filtering and PRMs. Their approach, referred to as *User Grade-based Collaborative Filtering - PRM (UGCF-PRM)*, is presented in Figure 4.3. The basic idea is to make predictions from both user-item matrix-based CF and a PRM, and then combine those predictions using a *user grade function* to finally recommend top-N items. A user grade function increases with the number of rating items. Hence, the user-grade for the users with large number of rating items will be higher than that for the users with few ratings. The user-grade function, in fact, acts like an adaptive weight for different grade users when combining the prediction from collaborative filtering with that from a PRM. The combined prediction P_{ui} is given as the weighted sum of the two predictions as follows:

$$P_{ui} = G_u \cdot P_{ui}^{NBS} + (1 - G_u) \cdot P_{ui}^{PRM} \quad (4.1)$$

where G_u is the user-grade function for the target user u , and P_{ui}^{NBS} and P_{ui}^{PRM} are the prediction from neighbor-based CF and that from PRM respectively for the target item i .

It is clear from Equation 4.1 that as the user grade function increases, UGCF-PRM is dominated by CF. Thus, only when the target user has few ratings, the prediction from PRM will be dominant.

4.2.5 A RBN-based recommender system architecture (Ben Ishak et al. [2013])

Through their hybrid approach for recommendation based on *Relational Bayesian Network (RBN)*¹, Ben Ishak et al. [2013] have tried to find a solution to the same problem that Newton and Greiner [2004] had addressed, i.e. the scenario where users' ratings depend on the previous ratings of the users' neighbors. As a solution to this, they propose to make a copy of the Rating class such that the original Rating class, called *Sound-votes*, represents the observed votes/ratings, and the duplicate class, called *Forecast-votes*, represents the objects that are supposed to be present. Now, the ratings from *Forecast-votes* can depend on the ratings from *Sound-votes* without creating a cycle in the original class dependency structure. The advantage of this

1. The term PRM used in this thesis is the same as the term RBN used by Ben Ishak et al. [2013].

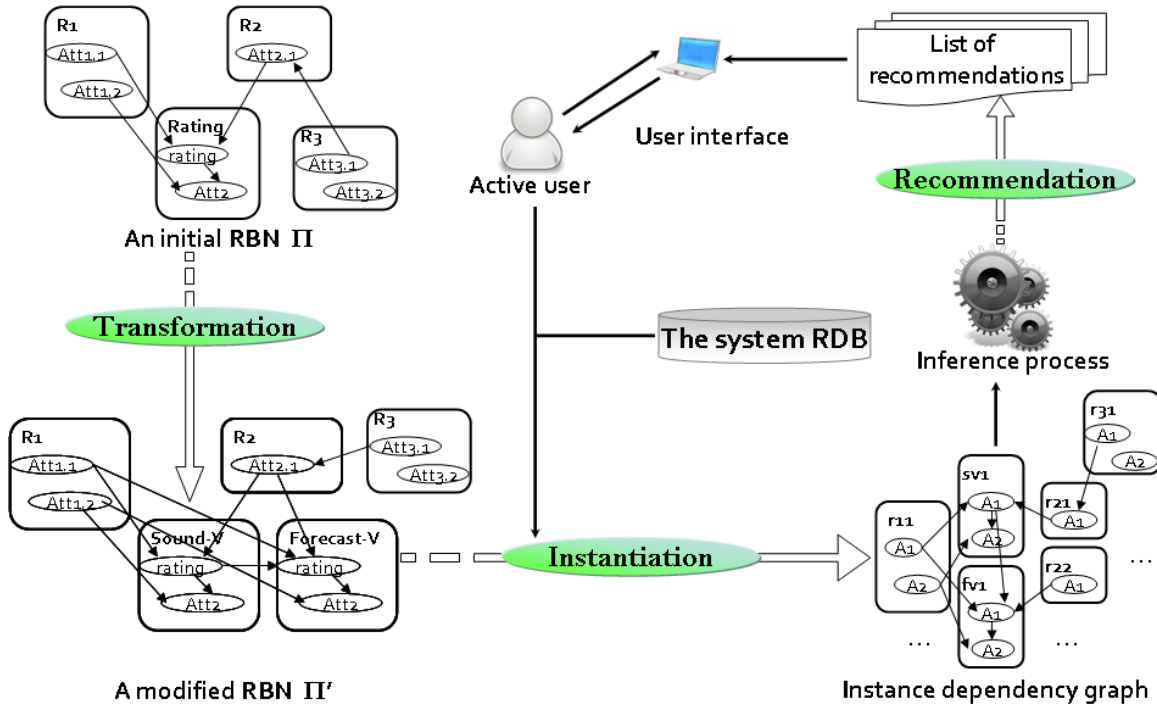


Figure 4.4 – The overall architecture of the recommender system proposed by Ben Ishak et al. [2013]

approach is that it limits structure search procedure and generates a simple model. However, this obtained model is not a strong model for recommendation. Hence, the model is enriched by providing an appropriate model instantiation to each active user based on a set of rules derived from recommendation requirements. This approach is capable of addressing data sparsity, cold start and scalability problems, and can also provide different recommendation techniques from the same model. The overall architecture of their approach is illustrated in Figure 4.4.

4.3 Comparison and discussion

Table 4.1 shows a summary of comparison of the PRM-based recommender approaches presented in Section 4.2.

Getoor and Sahami [1999]’s work builds a hybrid recommender system rather than a collaborative one as explained in the paper. The good points are that this model is easier to interpret than a BN-based collaborative filtering model, is extensible and can handle cold start and scalability problems. However, the authors do not clearly explain how to learn such PRMs and how to make actual recommendations from such models. Besides, the effectiveness of the model mainly depends on the quality of clusters, and it is difficult to obtain good clusters that can assign items/users to only one cluster. Also, the authors have not presented experimental results. Hence, we do not have a clear vision of how effective the model could be. Moreover, the specification of PRM used in this paper is different from the current specification, which is mature and, hence, more advanced than the one presented in the paper.

The introduction of multi-set operation has increased the expressiveness of the recommendation framework proposed by Huang et al. [2004]. It allows to capture data

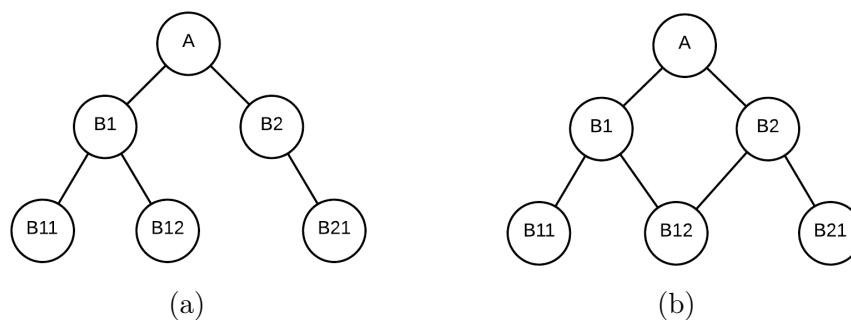


Figure 4.5 – Sample hierarchies an [hPRMs](#) cannot address: (a) a hierarchy where a node has only one (leaf) child, (b) a lattice structure where a node can be a subclass of more than one class

patterns that cannot be captured by simple (short) slot chains in regular [PRMs](#). The framework unifies different recommendation techniques simultaneously. The authors have illustrated the unification of content-based, demographics-based and collaborative filtering techniques in a book recommendation system. They have evaluated their model on a book sales data from a major Chinese bookstore. They have used precision, recall, F-measure and rank score as their evaluation metrics. Their experimental results show that their unified framework resulted in improved recommendation performance. However, the difficulty with this approach is that the relational feature space can grow very large depending on the length of the slot chains used in the model, and this can lead to computationally intensive feature construction and model estimation process. In spite of this difficulty, we found this approach of recommendation promising for a couple of reasons. First, it can address sparsity and cold start problems, and second, a single model can combine different recommendation techniques using the concepts of [PRMs](#) and [BNs](#) only.

[Newton and Greiner \[2004\]](#) have implemented their [hPRM](#) in a system, called the *Tadpole* system, and have evaluated this model on EachMovie dataset, which was the basis of MovieLens dataset (more information about this dataset in [Table 3.1](#)) and contained movie-rating data. Though their experimental results show that [hPRMs](#) have competitive performance in terms of *average absolute deviation* (cf. [MAE](#) in [Section 3.3.2](#)) against some algorithms engineered specifically for the recommender task (such as [BN](#) model, vector similarity-based algorithm, Bayesian clustering etc.), there are some limitations with this approach. First, though the same notations and examples from [Getoor’s PRM-CH](#) have been used, [hPRM](#) is different from [PRM-CH](#) and it does not actually use the hierarchical structure of [PRM](#). Only leaves are considered to define subclasses. This is like dividing a class into different subclasses and learning [PRM](#) as a regular [PRM](#). Thus, the inner classes are always missed out. For example, [hPRM](#) cannot address the hierarchy as shown in [Figure 4.5a](#), where class B2 can be an interesting class but since it has one child, it is never considered in the [hPRM](#). Second, it cannot address lattice structure, such as the one in [Figure 4.5b](#), which is very common in real world. For example, a movie can have more than one genre. Sub-classes of *Movie* may not always be disjoint as opposed to what was proposed by the authors. Third, the greedy partitioning used to create the hierarchy is not a good approach because the partitions created using this approach depend on datasets.

The [UGCF-PRM](#) approach aims to overcome simultaneously the most common problems of [CF](#): sparsity, scalability and cold start. The authors have validated this model on MovieLens dataset (more information about this dataset in [Table 3.1](#)) and

Table 4.1 – Comparison of some PRM-based recommendation approaches

	Getoor and Sahami [1999]	Huang et al. [2004]	Newton and Greiner [2004]	Gao et al. [2007]	Ben Ishak et al. [2013]
Recommendation method	CF	CF, Content-based, and hybrid	CF	CF/Hybrid	Hybrid
Structure Learning	Structure is defined by experts	Structure includes the target variable and its Markov blanket, and is learned from data.	Prior to applying standard PRM learning, greedy partitioning of some classes (such as <i>Movie</i>) is involved.	Not mentioned (probably defined by experts).	Application of standard PRM structure learning is proposed.
Parameter Learning	Proposed to apply score-based approaches for BN learning (to find the best number of clusters of users and items) but not clearly formalized	Proposed to apply standard parameter estimation procedures	Standard PRM learning algorithms are applied.	Standard PRM learning algorithm is applied	Proposed to apply standard PRM learning algorithms
Address cold start problem?	Yes	Yes	No	Yes	Yes
Implemented?	No	Yes	Yes (The <i>Tadpole</i> system)	No	No
Evaluated?	No	Yes	Yes (Mean absolute deviation (same as MAE))	Yes	No
Evaluation metrics	–	Precision, Recall, F-measure, and Rank score	EachMovie ^a	MAE	–
Evaluation dataset	–	Book sales dataset from a major Chinese bookstore	–	MovieLens	–

a. No longer available for download (<http://groupLens.org/datasets/eachmovie/>)

have shown that UGCF combined with PRM provide better performance in terms of MAE. However, their model does not actually exploit the capability of PRMs. The use of a PRM in their model is basically to handle the cold start problem. For a target user with many ratings, the effect of PRM becomes negligible. Besides, the authors give an example of a PRM but do not say anything about learning of such structure. In the sample PRM, the target attribute depends on attributes of all other classes with some intra-class dependencies. However, to achieve such model, we need either experts' knowledge or some probabilistic computation, and this is not mentioned by the authors. This leads us to expect that the PRM might not have represented the underlying data well in their experiments, which showed that PRM alone resulted in the worst MAE. So, we cannot fully rely on the result obtained from the PRM. In fact, the whole model can be achieved from Huang et al. [2004]'s approach which does not use traditional CF algorithms, instead makes use of Naïve Bayesian classifier for simplification of the problem.

The framework proposed by Ben Ishak et al. [2013] is claimed to be scalable and capable of making predictions even in cold systems. However, it has not been demonstrated through experiments. Because of the lack of its implementation and experimental results, it is not easy to determine whether making copies of objects as proposed in the article would cause problems related to memory for large datasets.

4.4 Conclusion

In this chapter, we reviewed some approaches that use PRMs for recommendation. The earliest PRM-based recommendation model proposed by Getoor and Sahami [1999] clusters users, and items separately, and make predictions based on those clusters. Inspired from PRM-EU formalism, Huang et al. [2004] proposed an interesting framework for recommendation that predicts the presence of a user-item link from a classifier derived from Markov blanket of the boolean 'Exists' attribute that tells whether the user-item link exists or not. Both of these models are extensible, scalable, and address the cold start problem. Newton and Greiner [2004] and Ben Ishak et al. [2013] try to solve the problem of cyclic dependency that usually appear in recommendation tasks. To avoid such cycles, Newton and Greiner [2004] proposed to extend regular PRMs into hPRM, whereas Ben Ishak et al. [2013] proposed to adapt underlying relational schema and the original PRM. Gao et al. [2007] combined user-grade based collaborative filtering with PRMs to tackle the cold start problem. In their approach, recommendations from PRMs are preferred only when the system is cold. All of these methods have good as well as bad sides. However, we found that Huang et al. [2004]'s recommendation framework is the most interesting among them as it is completely based on PRMs, easily extensible, capable of dealing with cold start situation, and also well documented. That is why we chose to implement this model (explained in Section 9.4).

Spatial Data

Contents

5.1	Introduction	72
5.2	Spatial data representation	72
5.2.1	Tessellation data representation	73
5.2.2	Vector data representation	73
5.2.3	Network data type	74
5.3	Characteristics of spatial data	75
5.3.1	Spatial heterogeneity	75
5.3.2	Spatial autocorrelation	75
5.4	Spatial operators	76
5.4.1	Metric operators	76
5.4.2	Topological operators	76
5.5	Conclusion	78

5.1 Introduction

The word *spatial* is originated from Latin word ‘*spatium*’, which means space. Thus, any kind of data that is related to space is referred to as *spatial data*. It is characterized by their location in two-(or three-)dimensional space, and includes not only things that exist at some location in the space such as countries, cities, roads, rivers, buildings, mountains, different elements in an image (in image recognition systems), a 3d model of the human brain etc. but also events such as accidents, floods, earthquakes, disease outbreaks, festivals, tapping on a touchscreen etc., which are related to the space in some way. A class of spatial data whose space is limited to the surface of the Earth is referred to as *geographic data*. The term ‘*geo*’ comes from Greek word ‘*gaia*’, which means the Earth. Thus, geographic data refers to the features or phenomena distributed on or close to the Earth’s surface. Often the terms geographic, *geospatial*, and spatial are used interchangeably even though geographical data is a subset of spatial data. In this research work, we are mainly focused on geographic data and also use the term spatial to refer to geographic data.

Availability of *Global Positioning System (GPS)* to civilian users, advances in mobile communication and wireless technology have opened the door to many spatial technologies. The development of consumer GPS tools, *Volunteered Graphical Information (VGI)* tools and location-aware mobile devices have revolutionized the way of collecting and using location information. This has resulted in the use of spatial information in a wide range of application, thereby increasing the need for analysis of spatial data. In this chapter, we will provide a general overview of basic concepts related to spatial data analysis.

Spatial data analysis requires a framework for modeling spatial information. In spatial statistics, spatial data are considered as a set of observations coming from spatial process, $X = \{X_s, s \in S\}$, where $s \in S$ are the positions of observational sites. In relational settings, such data can be perceived as a set of objects which are described by their location and a set of attributes. In conventional settings, spatial data consist of only one type of such objects whereas in relational settings, more than one type of objects (spatial as well as non-spatial) are considered. In both of these settings, implicit spatial relationships between objects exist. Due to such relationships, the IID assumption does not hold in spatial data. The implicit definition of relationships between spatial objects also give rise to specific characteristics, such as heterogeneity and spatial autocorrelation, in spatial data. These characteristics have been the basic tenets underlying the analysis of spatial data. Spatial data analysis studies patterns or influences resulting from such spatial characteristics. It uses various spatial operations on spatial objects, primarily to study the influence of spatial information in the system. In the following, we will present a brief overview of these concepts, which we will be referring later in the following chapters.

This chapter is organized as follows. Section 5.2 will present basic ways of representing spatial data. Section 5.3 will present two important characteristics of spatial data sets, and Section 5.4 will briefly discuss on some commonly used spatial operators.

5.2 Spatial data representation

Conceptually, spatial information are modeled as *field-based* and *object-based*. In field-based models, the world is seen as a continuous surface over which features vary whereas in object-based model, the world is seen as a surface littered with distinct,

identifiable and relevant objects which can be zero-dimensional (point), 1-dimensional (line) or 2-dimensional (surface) (Malerba [2008]). *Tessellation* and *vector* are two basic data types used to represent spatial information. In some literature (Shekhar et al. [2011]), *network* data type has also been reported. We are concerned with only the vector representation of spatial data in this thesis.

5.2.1 Tessellation data representation

In Tessellation representation of spatial data, space is partitioned into mutually exclusive cells that together make up the complete coverage. Each cell is associated with a thematic/attribute value that represents the condition for the area covered by that cell. A grid of square cells is a special tessellation model called *raster*. The size of the cell defines the level of spatial detail. The smaller the cell (pixel) size, the higher the resolution, therefore, the higher the level of spatial detail. Figure 5.1a shows an example of a raster data, where each cell of the grid is assigned a color (a thematic value).

Though tessellation is a simple data structure, and quantitative analyses can be simple to perform with it, there are some disadvantages of this representation. Since information about each cell must be recorded, tessellation data can become potentially very large, thereby making processing of associated data cumbersome. Besides, tessellation data inherently reflect only one attribute or characteristic for an area. However, in most input data, an area may be described by multiple attributes (ideal for vector representation, which will be described next). Thus, more processing may be needed to convert the input data into this type of representation.

5.2.2 Vector data representation

In vector data representation, spatial objects are modeled using *geometry* and their attributes. The geometry is made up of one or more interconnected vertices. A *vertex* describes a position in space using an x, y and optionally z axis. According to their dimensionality, spatial objects are classified as points, lines, or polygons.

Point is a zero-dimensional object that specifies geometric location. One coordinate pair or triplet specifies the location. Points can represent several kinds of real entities, e.g. restaurants, event locations, touristic places, bus stops etc. Figure 5.1b shows some spatial objects represented as point data placed on a map.

Definition 15 *Point*

A point is a zero-dimensional spatial object located within study area at coordinates (x, y) , where x is called longitude and y is called latitude. ■

Line is a one-dimensional object that consists of two or more vertices the first and last vertex not being equal. When four¹ or more vertices are present, and the last vertex is equal to the first one, an enclosed *polygon* object (or a *ring*) is formed. Rivers, train lines, roads etc. can be represented as line objects, whereas cities, parks etc. can be represented as polygons. Rivers can also be represented by polygons if we consider their width too. Figure 5.1c shows an example of line data. Cities in ‘Loire-Atlantique’ department of France are shown as polygons in Figure 5.1d.

1. Note that a triangle is a polygon with 4 (not 3) vertices such that the first vertex coincides with the last one. Thus, the minimum number of vertices required to form a polygon is 4.

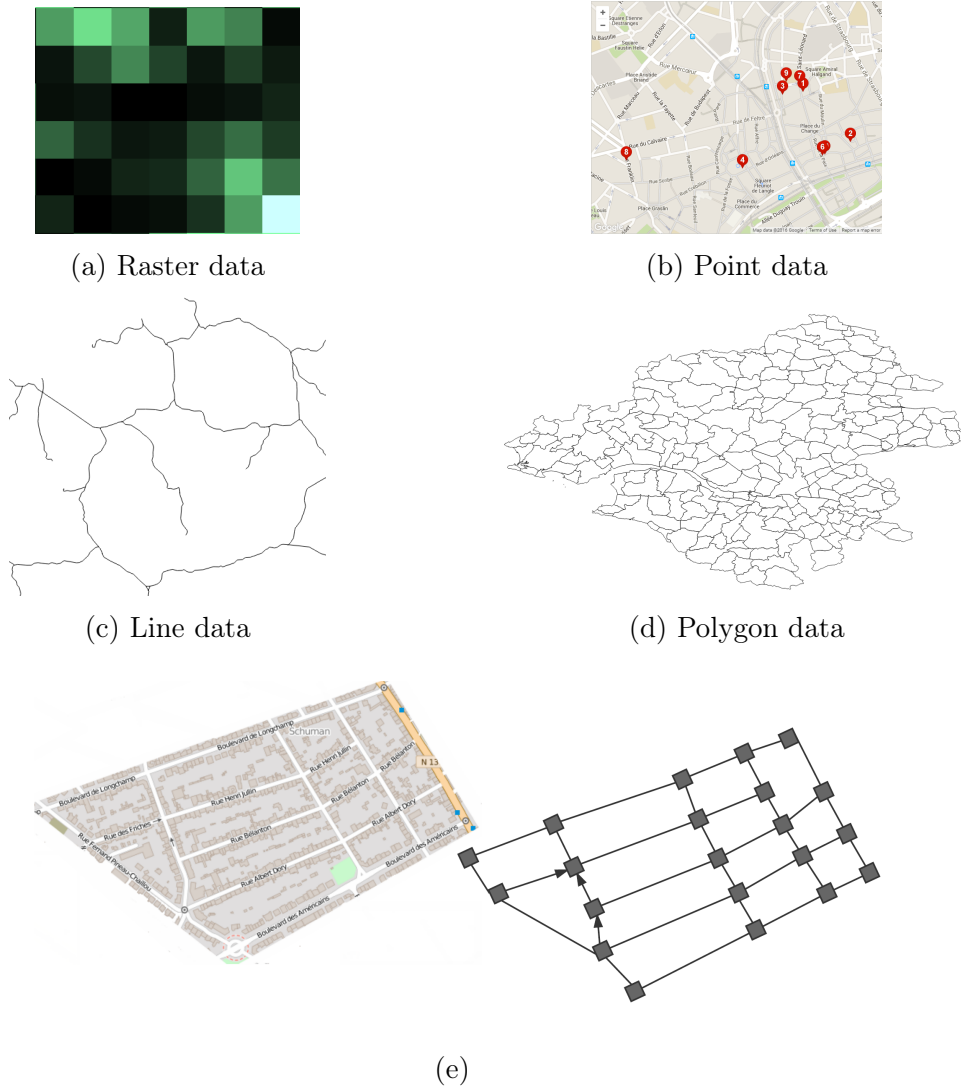


Figure 5.1 – Examples of spatial data

Definition 16 *Line*

A *line* is a one-dimensional spatial object defined by a sequence of at least two points $(p_n : n \in \mathbb{N}, n \geq 2)$ such that the starting point is not the same as the ending point, i.e. $p_1 \neq p_n$, where p is a point. ■

Definition 17 *Polygon*

A *polygon* is a two-dimensional spatial object defined by a sequence of at least four points $(p_n : n \in \mathbb{N}, n \geq 4)$ such that the starting point is the same as the ending point, i.e. $p_1 = p_n$, where p is a point. ■

5.2.3 Network data type

In network data type, a spatial domain is represented in a graph where spatial objects are abstracted as nodes and/or edges such that the graph contains the information connectivity between the objects. The edges can be directed or undirected. Defining a network is usually a problem-specific task, and sometimes the same information can be coded in a network data in different ways. For example, if we represent roads with a network data, nodes may be road segments and edges are the points where roads/road

segments meet (such as intersections, roundabouts etc.). The same roads can also be represented by another network where nodes are intersections/roundabouts and edges are road segments. A common way of defining a spatial network involves representing spatial objects by nodes, and denoting connectivity (e.g., whether there is a physical connection or whether they are within a fixed distance etc.) between those spatial objects by edges. Alternatively, both nodes and edges can be spatial objects as in the previous examples. An example of a spatial network is illustrated in Figure 5.1e. This network is constructed by abstracting intersections as nodes and road segments as edges. Here, directed nodes indicate one-way streets.

5.3 Characteristics of spatial data

When a spatial context comes into play in data, the observations/objects become no more independent. Spatial objects are often implicitly related to each other. Such implicit relationships give rise to special characteristics of spatial data. Two important characteristics of spatial data are *spatial heterogeneity*, and *autocorrelation*.

5.3.1 Spatial heterogeneity

Spatial heterogeneity refers to the uneven distribution of observed process over space. The influence of spatial context on spatial relationships can be seen in the variation of observed attributes over space (Shekhar et al. [2011]). For example, cultures may differ from one country to another, hotels/apartments in city center may be more expensive than those in suburbs and so on. When the value of an attribute of a site is different than its surrounding, it is referred to as local spatial heterogeneity. Such phenomenon gives rise to hotspots/coldspots.

5.3.2 Spatial autocorrelation

Standard statistical analysis are based on the assumption that the observations are made independently. However, this assumption is often (but not always) violated when spatial information is involved. Observation in one geographical place might depend on the observation in nearby places. Such phenomenon is also described by Tobler's first law of geography, which states "Everything is related to everything else, but near things are more related than distant things" (Tobler [1970]). Ignoring geographic influence during analysis may result in unrealistic findings. Spatial autocorrelation occurs in many disparate fields. For example, price of a house tends to be similar to other houses nearby; people with similar income tend to be neighbors; hyperlinked pages tend to share similar topics; proteins located in the same place in a cell are more likely to share the same function than randomly selected proteins (Malerba [2008]) etc. Spatial autocorrelation refers to the dependencies that exist among observations that are attributable to the relative locations, or underlying two-dimensional ordering, of variable values in geographic space (Griffith [1992]).

5.4 Spatial operators

Spatial operators are used to define interactions between spatial objects, or to derive new information. Broadly, these spatial operators can be classified into two categories² – (1) metric operators, and (2) topological operators.

5.4.1 Metric operators

Spatial metric operators are applied on a single spatial object to derive new non-spatial information about the object. For example, length is a metric operator that can be applied on lines and polygons but not on points.

5.4.2 Topological operators

Spatial topological operators utilize connectivity and contiguity information about one or more spatial objects to derive new spatial objects or to identify relationships between those spatial objects. Operators such as *aggregate*, *generalize*, *buffer*, *intersection*, *union*, *difference*, *symmetric difference*, *split*, *cut* etc. result in new spatial objects, whereas relational operators such as *contains*, *touches*, *overlaps* etc. are used to identify topological relationships between a pair of spatial objects. Table 5.1 lists some of these operators. This thesis deals mainly with spatial aggregation.

Aggregation operators

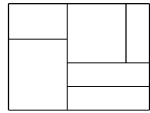



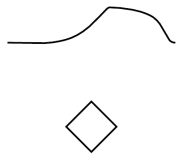
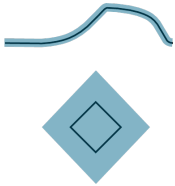
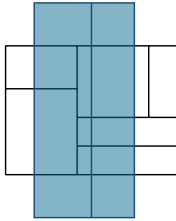
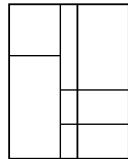
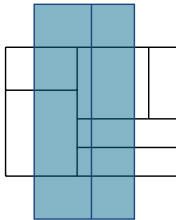
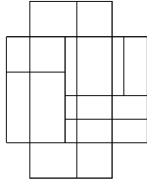
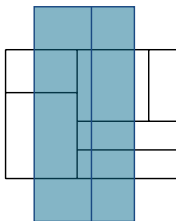
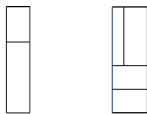
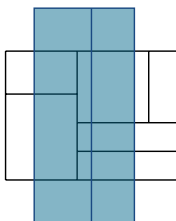
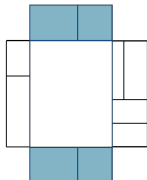
Spatial aggregation operators aggregate or summarize multiple spatial objects. Some of the commonly used spatial aggregators are *convex hull*, *centroid*, *union*, *Minimum Bounding Rectangle (MBR)* etc.

Relational operators

Spatial relational operators are Boolean methods that are used to identify specific topological relationships between spatial objects. Adjacency, overlapping, and containment are typical examples of spatial relationships. Such kind of relationships are derived from the nine-intersection model (Egenhofer and Herring [1990]) as spatial predicates about relations between a pair of spatial objects. The basic idea behind this model is to compare two spatial objects by making pair-wise tests of the intersections between the interiors, boundaries and exteriors of the two objects and classifying their relationship based on the entries in the resulting 3 by 3 ‘intersection’ matrix. The spatial predicates that are derived from this model and standardized by *Open Geospatial Consortium (OGC)* are *equals*, *disjoint*, *intersects*, *touches*, *crosses*, *within*, *contain*, *overlaps*, and *relate*.

2. This section only provides a brief overview of spatial operators. Readers are referred to the article Güting [1994] for in-depth understanding of spatial operators.

Table 5.1 – Examples of spatial operators

Spatial object	Operator	Output
	Aggregate	
	Generalize	
	Buffer	
	Intersection	
	Union	
	Difference	
	Symmetric difference	

5.5 Conclusion

In this chapter, we provided a brief introduction to spatial data. We presented how spatial data are represented for data analysis. We presented three basic representations of spatial data: tessellation, vector, and network. This thesis is concerned with vector representation of spatial data because vector data type is more compact than tessellation data type, and is well supported in major spatial relational databases, such as PostGIS and Oracle Spatial. Also, vector data type provides a general way to model spatial information from real world data whereas network data type is often domain-specific, and can have multiple representations for the same set of spatial information. We also briefly discussed on two important features of spatial data, which arise due to the relationships between spatial objects: spatial heterogeneity and spatial autocorrelation. Some commonly used spatial operators were also presented in this chapter.



6

Recommender Systems with Spatial Data

Contents

6.1	Introduction	80
6.2	Review of some spatial recommender systems	80
6.3	Discussion	82
6.4	Conclusion	84

6.1 Introduction

Over the past decades, there has been a significant growth in the number of the Internet users all over the world¹. Consequently, a huge number of applications has come into existence accumulating unprecedentedly large volume of data. Recent advances in location-acquisition and wireless communication technologies have made it easier to extend the data with spatial information. All of these have contributed to the growing interest of machine learning communities in the study of extraction of useful knowledge from spatial dimensions. Spatial data analysis has been a popular topic in the studies of environment, ecosystems, population, communities, image processing etc. However, due to the proliferation of relational data, recommender system communities are also attracted in exploiting spatial information to improve recommendation quality. Spatial statistics was an early approach to spatial data analysis. Among several machine learning techniques that were adopted later, BNs are popular particularly in environmental and ecological modeling (Barton et al. [2012], Landuyt et al. [2013], Aguilera et al. [2011]). Recently, Chee et al. [2016] have integrated *Geographic Information System (GIS)* data with OOBNs and *State-and-transition Dynamic Bayesian Networks (ST-DBNs)*. The increased use of social networks has also contributed in the growing trend of analyzing spatial data collected from location-based social networks, especially in the area of recommender systems (Bao et al. [2015]). In this chapter, we will review some recommendation systems that exploit spatial information.

6.2 Review of some spatial recommender systems

Levandoski et al. [2012] have proposed a recommender system, called location-aware recommender system (LARS), that is capable of providing recommendations from location-based ratings in three different scenario: (1) where spatial users (users with location information) rate non-spatial items (items without location information), (2) non-spatial users rate spatial items and (3) spatial users rate spatial items. In the first scenario, they propose a method of partitioning users based on their location. LARS employs a partial pyramid structure, as shown in Figure 6.1, which decomposes the space into H levels (i.e., pyramid height). For a given level h , the space is partitioned into 4^h equal area grid cells. In each cell, an item-based collaborative filtering model built using only the spatial ratings with user locations contained in the cell's spatial region is stored. To make top-K recommendation for a user u with location L , LARS finds the cell that corresponds to the location L in the pyramid, and uses the model stored in this cell to make recommendation. If a cell corresponding to L is not found, it returns the nearest maintained ancestor cells. They have also implemented an algorithm to maintain the pyramid structure. LARS* (Sarwat et al. [2014]) is an enhancement of LARS and has an improved, more efficient data structure maintenance algorithm. In the second case, where non-spatial rating for spatial items, they introduce travel penalty, which penalizes the recommendation rank of items based on their distance from the querying user. Ranking of each spatial item i is done based on the score $RecScore(u, i)$ for a querying user u .

$$RecScore(u, i) = P(u, i) - TravelPenalty(u, i)$$

where $P(u, i)$ is the standard item-based CF predicted rating of item i for user u . $TravelPenalty(u, i)$ is the road network travel distance between u and i , normalized

1. <http://data.worldbank.org/indicator/IT.NET.USER.P2>

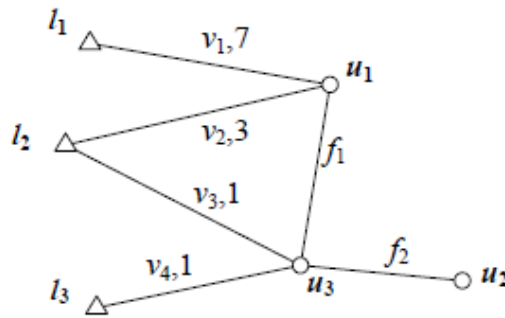


Figure 6.2 – An example of a location-based social network graph (Wang et al. [2013]). Here, u_i and l_j represents i^{th} user and j^{th} location respectively. Edges between users represent friendships, and those between users and locations represent check-ins. The latter type of edges is weighted by check-in counts.

geotagged items for a given user within a specified geographic region. They construct a weighted graph where each node is a user object together with its location, each edge denotes the nodes are similar or related, and the weights of the edges indicate how similar the nodes are. Recommendation is then performed by aggregating the weights. They present four strategies for assigning weights to the edges of the relational graph: weighting the edges uniformly, weighting based on correlation between user profiles, based on distance between users' items or based on partonomy information. With different combination of the weighting schemes, they are able to derive different types of recommendation algorithms, and also to address cold-start situations.

Like Ye et al. [2011], Wang et al. [2013] also support the significance of social interactions (or friendships), and the phenomenon of people visiting locations nearby their usual active area. They propose a system for recommending venues to users from their check-in history in a location-based social network. From users' friendships and their check-in history, the authors construct an undirected graph, as shown in Figure 6.2, where users and locations are represented by nodes of the graph, friendships by unweighted edges, and check-ins at particular locations by edges weighted by check-in counts. For making recommendations, they improve this graph by replacing friendship edges with similarity edges weighted by similarity due to friendships and similarity due to common check-ins. The system computes *Personalized PageRank (PPR)*, which is based on the fundamental idea of PageRank that important pages are referenced by many important pages, by performing random walk with restart on the improved graph. It then recommends to the user N unvisited locations with highest PPR after filtering out the locations that are far away from the user's history.

A summary of the discussed recommender systems are provided in Table 6.1.

6.3 Discussion

Numerous works can be found in the field of spatial recommender systems. Although many of them are very specific to the problem domains, their objectives can be roughly categorized into three types: (1) user recommendation, (2) item recom-

Table 6.1 – Comparison of some recommender systems that exploit spatial data

Article	Objective	Recommendation approach	Domain	Techniques involved	Geographical influence measure	Evaluation dataset
Levandoski et al. [2012] / Sarwat et al. [2014]	Item recommendation	Hybrid	General	KNN	Hierarchical geographic information; Distance	MovieLens; FourSquare
Huang and Bian [2009]	Item recommendation	Hybrid	Tourism	BN, AHP, Ontology	Distance	–
Ye et al. [2011]	POIs (location) recommendation	Hybrid	Tourism	Regression, naive Bayesian method, Power law distribution, location-based social networks	Distance	FourSquare
Marinho et al. [2012]	(Geotagged) Item recommendation	CF / Hybrid		Weighted relational graph	Hierarchical geographic information; Distance	Geotagged photos from Panoramio; Print jobs from HP ePrint mobile service
Wang et al. [2013]	Venue (location) recommendation	CF / hybrid	Tourism	Cosine similarity; Personalized PageRank (PPR) / Random walk with restart	Distance	Gowalla; Brightkite

mentation, (3) location recommendation. Recommender systems proposed by [Huang and Bian \[2009\]](#), [Levandovski et al. \[2012\]](#), [Sarwat et al. \[2014\]](#), and [Marinho et al. \[2012\]](#) aim at recommending items, whereas the one proposed by [Ye et al. \[2011\]](#) aim at recommending location. Location-based social networks are particularly interested in recommending locations and/or users (e.g. recommending ‘People you may know’) ([Bao et al. \[2015\]](#)).

Regardless of the recommendation objectives, geographic distance has been a primary criterion for measuring geographical influence in many spatial recommender systems. All recommender systems reviewed in Section 6.2 and many other (e.g. [Bao et al. \[2012\]](#)) use distance measure. Hierarchical geographical information is also commonly used to measure geographical information (e.g. [Levandovski et al. \[2012\]](#), [Marinho et al. \[2012\]](#) [Ahmed et al. \[2013\]](#)). Very few works have considered spatial relationships such as neighborhood, inclusion etc. (e.g. [Walker et al. \[2005\]](#), which is not a true recommender system but applies spatial relationships information with Bayesian networks to realize spatial Bayesian networks) for the same task.

6.4 Conclusion

In this section, we reviewed some state-of-the-art recommender systems that consider geographical influence while making recommendations. We categorized them based on the recommendation objectives and the techniques used for measuring geographical influence. From this review, we point to an interesting direction for research to explore the applicability of spatial relationships and operators (presented in Section 5.4) for incorporating geographical effect in the recommendation models.



Contributions



A Personalized Recommender System from PRM and Users' Preferences

Contents

7.1	Introduction	88
7.2	The proposed approach	90
7.2.1	PRM for preference-based recommendation (PRM-PrefReco)	90
7.2.2	Personalization	93
7.2.3	Relational attributes and types of model	96
7.2.4	Examples	98
7.3	Experiments	101
7.3.1	Dataset	101
7.3.2	Experiment methodology	102
7.3.3	Evaluation metrics	102
7.3.4	Results and discussion	103
7.4	Conclusion	104

7.1 Introduction

Recommender systems have found their applicability in a variety of domains. However, the application of RS is still dominated by solutions for recommending relatively simple and inexpensive products like movies, music, news, restaurants etc. (Ricci et al. [2015]). Relatively few works on the application of RS for recommending expensive or less frequently purchased items (e.g. real estate properties, flights, travel packages, financial investments etc.) can be found. As mentioned in Section 3.4, the application of RS on such domains might not have been much explored probably due to the underlying financial risks. For example, recommending real estate properties demands a serious responsibility as the RS needs to help users in making a good decision that involves a serious financial matter. Flight recommendations may be less challenging than real estate properties recommendations but flights are more expensive and less frequently purchased than movies or books. An important issue faced by RSs that recommend less frequently purchased items is the lack of user profiles because users tend to have only short-term preferences in such systems, and depending on the domain of the system, building user profiles may not always be interesting for users. As discussed in Section 3.4, acquiring user profiles is challenging, especially when the system is new. Users may be reluctant to provide personal information due to security reasons or because they may not want to readily trust new systems. In this chapter, we present a recommender model that aims at addressing this scenario.

The motivation of our work is the need of a recommender system for helping users find real estate properties in a new real estate search system, Kyzia¹, which is in cold-start situation and lacks user profiles. Though our target application is Kyzia, the solution we have provided is not limited to the domain of real estates only. Our solution is applicable on feature-based systems (cf. Section 3.2) where recommendations need to be made based on users' (usually short-term) preferences for items' features/characteristics. Some examples of such systems include flight search, hotel booking, real estate search, job search etc.

Shearin and Lieberman [2001], Viappiani et al. [2007], and Smyth [2007] have proposed feature-based recommender systems for similar domains (but not necessarily in cold-start situation). The first two works deal with the problem of finding apartments whereas the last one have presented a case-based RS for more general domains. In their critiquing-based recommender system (called Apt Decision), Shearin and Lieberman [2001] try to mimic how human real estate agents work by first providing a set of sample apartments from few criteria (e.g., "I would like to rent a two-bedroom apartment in Somerville for about \$1500."), then getting feedback from the client about various other features of these sample apartments (e.g., whether it has a parking area, or whether pets are allowed etc.), and converging to a complicated set of constraints and priorities from these feedback to finally propose a list of apartments. Viappiani et al. [2007] have developed a critiquing-based recommender, based on the *look-ahead principle* (Viappiani et al. [2006]) to help users express more preferences. The main idea is to get a small set of preferences from the users at the beginning, and let the users add new preferences (which might be hidden initially) to discover better items. For this, they show not only the items that best match to the users' preferences but also the options (or items/products) that would become optimal if the users add a new preference. They build a probability model for predicting users' new preferences. Smyth [2007]'s case-based RS is a more general approach for recommending items in feature-based

1. <http://www.kyzia.fr>

systems. They compute similarity between users' query (preferences) and the features of items to find the best items. The interesting part of their approach is the use of various similarity metrics depending on the type and importance of preferences. They use different similarity metrics for numeric preferences such as price of items, number of bedrooms (of apartments), pixel resolution (of cameras) etc. and non-numeric ones such as vacation types. Pu et al. [2012] and Chen and Pu [2012] have presented a detailed review of more feature-based and critiquing-based RSs, which are applicable in similar domains.

Though a critiquing-based RS sounds appealing for addressing the cold-start problem in a feature-based system, we do not proceed to implement it in our target application due to three reasons. First, it demands users' patience to receive good recommendations because the first recommendations are usually primitive, and the users need to provide their feedback (probably multiple times) to improve the recommendations. Thus, we aim at providing single-shot recommendations rather than implementing a conversational RS. Second, these systems take into account users' preferences for attribute values but in many cases, such preferences may not be enough. Instead, preferences about the attributes themselves may be more important than the attribute values, and such preferences about attributes tend to vary from one user to another. Thus, a better way for providing personalized recommendations is needed. For example, two users looking for a 2-bedroom apartment with the surface area between 50m² to 100m² and the price in the range €500 – €1000 might have completely different preferences about price, surface area etc. One user might have a limited budget, so his preference for price might be stronger than surface area (i.e., he might be fine with a smaller apartment given it fits his budget) but the other one might not be willing to compromise on surface area but might be fine with adjusting his budget. Similarly, in a flight search system, some users might be flexible about departure/arrival time but not about the price whereas some others might be ready to adjust their budget to get the flight that arrives/departs at a very specific time frame or the one without stops. Third, we want to combine the feature-based approach with the concept of collaborative filtering by utilizing the transaction history that a cold system would collect as it grows. In other words, we would like to build a recommender system that a system could use in the cold-start situation and continue to use with as less changes as possible when the system is no more cold.

We have seen in Chapter 3 that RSs are built around relational information about objects (i.e., users, items, and relationships between them), and in Chapter 2 that PRMs are suitable for modeling such relational domains. Thus, we aim at using PRMs to model the recommendation task. Recommending an item to a user is, in fact, the task of predicting whether there would be an interaction between them. A PRM-EU is suitable for solving such problem as it deals with the estimation of probability of existence of relationship between objects. Besides, using PRMs in RSs has long been studied, as discussed in Chapter 4. Therefore, we propose a novel approach of constructing a PRM-based recommender system that is capable of personalizing recommendations using users' preferences in feature-based systems. We propose a generic PRM for preference-based recommendations, which we refer to as PRM-PrefReco and provide details on how to apply users' importance about various decision factors to personalize recommendations from this PRM. Our approach is applicable in cold as well as hot systems. Only a small change in the length of slot chains is needed in the PRM to transform the recommendation model from a content-based model to a collaborative filtering one. We apply this approach on Kyzia and study its effective on

a small dataset.

This chapter is organized as follows. In Section 7.2, we will explain our proposed model in detail. In Section 7.3, we will present the findings from our preliminary experiment performed on a system that is in cold start situation.

7.2 The proposed approach

We propose a recommender system, based on [PRM-EU](#), that can personalize recommendations from users' preferences. The basic idea behind our [RS](#) is as follows. Users state their preferences about values of items' attributes as well as about the items' attributes themselves. The system then makes recommendations from a probabilistic model that takes into consideration the users' preferences. If the user clicks on/visits any of the recommended items, we assume that the user might have found the item interesting. Collecting such implicit data is often needed in a domain where items are infrequently purchased (such as real estate, flights). In this thesis, we refer to such implicit data as 'transaction'. Our goal is to recommend items that are more likely to be visited.

Our proposed approach involves the following two components:

1. A generic [PRM](#) that models preference-based recommender systems. We call it a *Probabilistic Relational Model for Preference-based Recommenders* ([PRM-PrefReco](#)), and
2. A personalized Bayesian network obtained from this [PRM](#) by applying the preferences of the target user.

The [PRM-PrefReco](#) is built off-line using experts' knowledge and/or the available data in the system. So, when a user uses the system, the only thing needed is to build a Bayesian network from the [PRM-PrefReco](#) according to the preferences specified by the user. In the following, we will explain these concepts in detail. Section 7.2.1 will provide details on [PRM-PrefReco](#), and Section 7.2.2 will explain how to construct a Bayesian network from a [PRM-PrefReco](#) to make personalized recommendations.

7.2.1 PRM for preference-based recommendation (PRM-PrefReco)

As a [PRM](#) is defined for a relational schema, we begin by explaining the relational schema of our preference-based [RS](#). We will then discuss on the probabilistic structure and parameters of our proposed [PRM-PrefReco](#).

Relational schema

The relational schema of our proposed [RS](#) is slightly different from that of traditional [RSs](#) but is applicable in any preference-based [RS](#). Traditionally, a [RS](#) involves three kinds of objects – users, items and transactions between them. However, in our proposed [RS](#), users and items are not directly related but are related through users' preferences for items' characteristics. We designate such preferences as objects of class 'Search'. Users' interaction with any item recommended for the given 'Search' object² is represented by 'Transaction' class. Thus, three main classes in our relational schema are Item, Search, and Transaction.

2. Here we use the terms 'Search' object and search session interchangeably

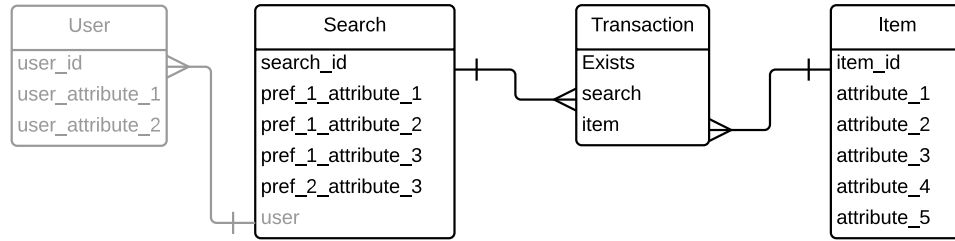


Figure 7.1 – Relational schema of our proposed preference-based recommender system. `Search`, `Transaction`, and `Item` are the main classes of the system. These classes can be accompanied by other classes such as `User`. Here, users’ preferences for `Item.attribute_1`, and `Item.attribute_2` is captured by `Search.pref_1_attribute_1`, and `Search.pref_1_attribute_2` respectively. Users can specify two preferred values for `Item.attribute_3`, which are captured by `Search.pref_1_attribute_3` and `Search.pref_2_attribute_3`.

The `Item` class is described by a set of attributes. Users express their preferences for the value of these attributes. The `Search` class is described by such preferences (aka search criteria). Note that usually the `Item` class can have many attributes, and the `Search` class will have attributes corresponding to a subset of the attributes of the `Item` class. In other words, for each attribute of the `Item` class, there can be zero, one or more attributes in the `Search` class for which users state their preferred values. For example, if a user can specify only one preferred value for an attribute of the `Item` class, then the `Search` class will have only one attribute corresponding to this attribute of the `Item` class; if a user can specify a range (i.e., minimum and maximum values) as his preferred value for an attribute of the `Item` class, then the `Search` class will have two attributes corresponding to this attribute of the `Item` class. Also note that there can be additional classes of objects (e.g., `User` class related to the `Search` class) in the relational schema, as shown in Figure 7.1. Following PRM-EU (cf. Section 2.4), we add a binary attribute ‘exists’ to the `Transaction` class. This attribute indicates whether there is a transaction between a `Search` object and an `Item` object.

Formally, a relational schema of a preference-based RS can be defined as follows.

Definition 18 *Relational schema of a preference-based RS*

The relational schema of a preference-based RS contains at least the following three classes $\mathcal{X} = \{\text{Item}, \text{Search}, \text{Transaction}\}$ defined by the following descriptive attributes:

- $\mathcal{A}(\text{Item}) = \{a_i\}$ for $i \in \{1, \dots, n\}$, where n is the number of attributes of the `Item` class
- $\mathcal{A}(\text{Search}) = \{\text{pref}^j_a_i\}$ for some $a_i \in \mathcal{A}(\text{Item})$, and $j \in \{1, \dots, k_i\}$, where k_i is the number of values the system allows a user to specify as his preferred values for the attribute $a_i \in \mathcal{A}(\text{Item})$
- $\mathcal{A}(\text{Transaction}) = \{\text{exists}\}$

The schema involves at least the following two reference slots: `Transaction.Search`, and `Transaction.Item`. ■

Given the relational schema and users’ preferences (for the attributes as well as the attribute values), the task for our recommendation model is to predict the value of `Transaction.exists`, i.e. to predict how likely the user would visit the item for the given search criteria.

Probabilistic structure

The target attribute `Transaction.exists` can actually depend on many attributes from `Item` and `Search` as shown in the `PRM` of Figure 7.2a. This makes it difficult to know the distribution of the target attribute given very specific configurations of its parents because the distribution table can be very big due to the large number of parents. Thus, to simplify this as well as to capture users' preferences for every search criteria, we *divorce* (Kjærulff and Madsen [2007]) the parents or introduce intermediate variables so that the target attribute will depend only on small number of attributes. These intermediate variables, referred to as *Decision Factors (DFs)*, will belong to the `Transaction` class in the `PRM` and are chosen in such a way that each decision factor represents a search criterion. The `PRMs` before and after parent divorcing are shown in Figure 7.2. It should be noted that decision factors can have more than two parents in complex cases, and that these parents can be further divorced to simplify the conditional distribution tables.

Defining decision factors Decision factors, in fact, give a measure of how close or far the item attribute is from the criterion expressed by the users. Thus, our idea is to add one decision factor for one `Item` attribute and the corresponding `Search` attributes that capture users' preferences for that particular `Item` attribute. The parents of a decision factor DF_i would then be `Transaction.Ki.ai`, and $\{\text{Transaction.Search.pref}^j_{a_i}\}$, where $j \in \{1, \dots, k_i\}$ such that k_i is the number of values the system allows a user to specify as his preferred values for the attribute `Item.ai`, and \mathbf{K}_i is the desired slot chain which it starts from the `Transaction` class and ends in the `Item` class. This slot chain determines the type of the recommendation model and can have a length of 1, 3, 5 and so on. More on the types of the model will be presented in Section 7.2.3. The selection of the attributes for decision factors is, in fact, a domain-specific problem. Thus, we may need experts' advice to define parents of decision factors.

Parameters

Once the structure of the probabilistic model is defined, we need to assign parameters to the model. In our `PRM-PrefReco`, we need the following `CPDs`:

1. $P(\text{Search}.A)$ for each attribute A of the `Search` class
2. $P(\text{Item}.B)$ for each attribute B of the `Item` class
3. $P(\text{Transaction}.DF_i \mid Pa(\text{Transaction}.DF_i))$ for each decision factor DF_i , and
4. $P(\text{Transaction.exists} \mid Pa(\text{Transaction.exists}))$

The first two `CPDs` are learned from data whereas we need experts' knowledge to define the third one. The `CPD` of the target variable, i.e., $P(\text{Transaction.exists} \mid Pa(\text{Transaction.exists}))$, should change according to users' preferences to achieve personalized recommendations. Thus, this `CPD` needs to be computed during the instantiation of this `PRM` to obtain a personalized Bayesian network for a particular user. That means this `CPD` is not well-defined at the time of the construction of the `PRM-PrefReco`. However, from Definition 8, we need to define `CPDs` for all random variables of the `PRM`. Therefore, to comply with Definition 8, we can assign an arbitrary `CPD` to our target variable. This `CPD` will then be revised during the instantiation of the `PRM-PrefReco` for a target user. This will be further explained in the following section.

Formally, we define our proposed `PRM` for preference-based recommendation in the following way.

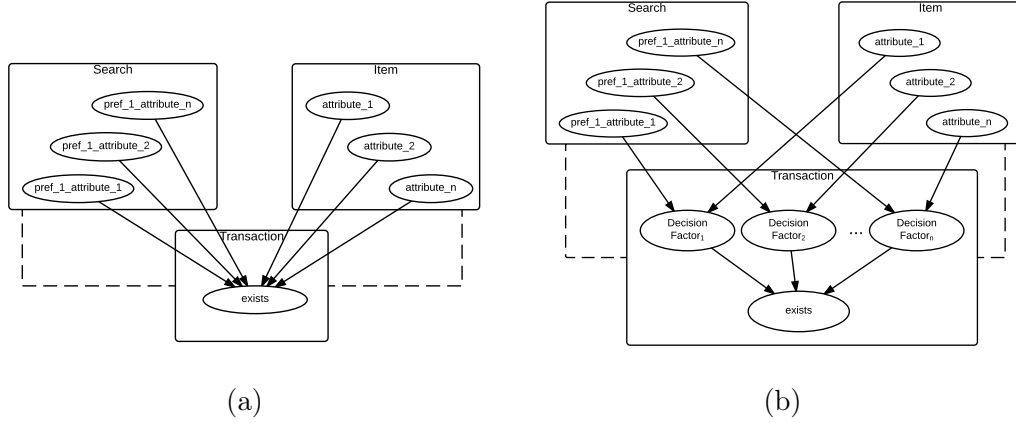


Figure 7.2 – The proposed **PRM** (a) before introducing decision factors, and (b) after introducing decision factors.

Definition 19 *PRM for preference-based recommendation (PRM-PrefReco)*

A *PRM* for preference-based recommendation $\Pi_{PR} = (\mathcal{S}, \Theta)$ for a relational schema of Definition 18 is a *PRM* (Definition 8) with the following

– Structure, \mathcal{S} :

- For each attribute of the classes *Item*, *Search*, and *Transaction*, there is a random variable in \mathcal{S} ,
- For each $\text{Item}.a_i \in \mathcal{A}(\text{Item})$, if $\text{Item}.a_i$ has at least one corresponding attribute in the *Search* class (i.e., $\text{Search}.pref^j_{a_i}$), then a random variable $\text{Transaction}.DF_i$, aka a decision factor, is introduced in \mathcal{S} . Let DF be the set of such decision factors,
- \mathcal{S} has the following dependencies:
 - $Pa(\text{Transaction}.DF_i) = \{\text{Transaction}.Search.pref^j_{a_i}, \text{Transaction}.\mathbf{K}_i.a_i\}$ for all decision factors, where \mathbf{K}_i is the desired slot chain starting from the *Transaction* class to the *Item* class
 - $Pa(\text{Transaction}.exists) = \{\text{Transaction}.DF_i\} \forall DF_i \in DF$

– Parameters, Θ :

- Following *CPDs* are well-defined:
 - $P(\text{Search}.A) \forall A \in \mathcal{A}(\text{Search})$ (learned from data),
 - $P(\text{Item}.B) \forall B \in \mathcal{A}(\text{Item})$ (learned from data), and
 - $P(\text{Transaction}.DF_i | Pa(\text{Transaction}.DF_i)) \forall DF_i \in DF$ (defined by experts)
- $P(\text{Transaction}.exists | Pa(\text{Transaction}.exists))$ is not well-defined at this point because it is revised during personalization (initialization). However, to get a complete *PRM*, each probability distribution in this *CPD* can be assigned a uniform distribution. ■

7.2.2 Personalization

To make recommendations to a user from a **PRM-PrefReco**, we need to initialize the **PRM-PrefReco** with our database instance and the user’s preferences. We receive two types of preferences from users:

1. Preferred values for items’ attributes, and
2. Preferences about the items’ attributes (that is equivalent to users’ preferences about decision factors).

If we do not want to consider the second type of users' preferences, the ground Bayesian network (GBN) obtained by unrolling the PRM-PrefReco over the database instance and the users' preferences would be enough for making recommendations provided that the CPD of the target variable *Transaction.exists* has already been properly defined³ during the construction of the PRM-PrefReco. Most of the existing PRM-based recommenders (see Chapter 4) are observed to follow this approach. We point out two problems with the GBN obtained in such way. First, it obviously does not consider users' preferences about decision factors. Second, it will be largely affected by the distribution of unobserved attributes of *Search* and *Item* when they should not. Not observing a *Search* attribute means that this attribute should not have any effect on the final decision. Therefore, to overcome these problems, we propose another method to build a personalized Bayesian network from the PRM-PrefReco.

Constructing a personalized Bayesian network for recommendation

We construct a personalized Bayesian network for a *Search* object and a target *Item* object and estimate the probability of *Transaction.exists*. We first create relational attributes from the PRM-PrefReco by traversing through reference slots in the relational skeleton. We keep in the model the observed *Search* attributes, their spouses, the corresponding decision factors, and finally the target variable. The attributes from the *Item* class may or may not be observed because in real world, not all the attributes of the items can be expected to be observed. Examples of some recommendation models constructed in this way are shown in Figure 7.3. Details about these models will be provided in Section 7.2.3. Inference will then be performed on the personalized Bayesian network after assigning a conditional distribution table of *Transaction.exists* based on users' preferences about decision factors.

To capture users' preferences about decision factors in the recommendation model, we propose using an approximation method to build the CPD of the target variable. Noisy-OR (Pearl [1988]), and Leaky Noisy-OR (Henrion [1987]) are well-known approximation algorithms, which require separate influence of each decision factor. We view the problem of constructing this CPD as a multi-criteria decision making (MCDM) (Triantaphyllou and Mann [1989]) problem. From the users' preferences about decision factors, we find weights W_i for each decision factor DF_i and apply some heuristics, e.g., *Weighted Sum Method (WSM)* (Fishburn [1967]), Noisy-OR (Pearl [1988]), or Leaky Noisy-OR (Henrion [1987]), to generate the conditional probability table for the target variable. We propose some ways of ranking decision factors to obtain their weights.

Ranking decision factors We propose here three methods to rank decision factors according to users' preferences. Users are asked to provide their preferences over only a subset of decision factors to reduce the number of questions to be asked to the users. Low weights are assigned to the decision factors that are not presented to the users. Here, we use the terms 'decision factors' and 'search criteria' interchangeably because each decision factor is associated with a search criterion.

Ranking method I We ask the users to sort the decision factors according to their importance. We then assign predefined weights to those decision factors such that the most important decision factor gets the largest weight and the least important one

3. Experts can help identify the importance of each decision factor and specify their influence on the target variable to construct the CPD of the target variable.

Table 7.1 – The scale of absolute numbers for expressing the relative importance of a pair of decision factors (Saaty [2008])

Intensity of importance	Definition
1	Equal importance
2	Weak or slight importance
3	Moderate importance
4	Moderate plus
5	Strong importance
6	Strong plus
7	Very strong or demonstrated importance
8	Very, very strong
9	Extreme importance
Reciprocals of above	If DF_i has one of the above non-zero numbers assigned to it when compared with DF_j , then DF_j has the reciprocal value when compared with DF_i

gets the lowest weight. For example, assign 1 to the least important decision factor and increase the weight by 1 for the next decision factor in the sorted list so that the weight of the most important decision factor will be equal to the number of decision factors. The decision factors that are not presented to the users get the lowest weight (1 or less in the previous example).

Ranking method II We apply Analytic Hierarchy Process (AHP) (Saaty [1980]) to rank the decision factors and find their weights. For this, we ask the users for their relative importance for every pair of the decision factors. We can apply the scale shown in Table 7.1, proposed in Saaty [2008], to express the relative importance of the decision factors. These pairwise comparisons can be expressed as a matrix, referred to as *pairwise comparison matrix*, where each row and column represents a decision factor such that each cell represents the relative importance between a pair of decision factors. A widely used solution to derive the relative weight of each decision factor from this comparison matrix is the eigenvector method, where the principal right eigenvector of this matrix gives the weights of the decision factors. Again, the decision factors not presented to the users are assigned weights lower than or equal to the lowest weight.

Ranking method III The first method does not capture users' view on relative importance of the search criteria, which can be a crucial input for better result. The second method captures this information but is not very practical because we need to ask for relative importance for every pair of search criteria and the number of such pairs grows combinatorially. Thus, we propose another approach in which we ask users to rank them in a scale such that the gap between two criteria can represent relative importance of the search criteria. From this information, we prepare the matrix required to perform AHP and follow the method II afterwards.

7.2.3 Relational attributes and types of model

The slot chain \mathbf{K}_i determines which objects to consider in the personalized Bayesian network while making prediction about *Transaction.exists*. Depending on the length of the slot chain \mathbf{K}_i used in the *PRM-PrefReco*, three types of recommendation models can be achieved. In the following, we will be using Tx , S , and I as a short-hand for the *Transaction* class, the reference slot from *Transaction* to *Search*, and the reference slot from *Transaction* to *Item* respectively. Figure 7.3 illustrates examples of the three types of recommendation models obtained for the third *Search* object ($id_search=3$) and the fourth *Item* object ($id=4$) by instantiating the *PRMs-PrefReco* having relational attributes of different slot chain length. In this figure, grayed-out objects are the ones that do not contribute in the recommendation model.

Type I Figure 7.3a shows a Type I model. This type of model is the simplest model where the length of the slot chain is 1. It is a typical scenario of cold start problem where the current user does not have search history at all and the system does not have enough data from existing users to predict based on their behavior. The prediction from this model will be the result from experts' knowledge and users' preferences for the search criteria. Hence, this model is purely a content-based model.

Type II It is an extension of Type I model with some relational attributes of slot chain of length 3. An aggregation function (e.g., mode, mean, cardinality) is required when the slot chains are not guaranteed to be single-valued. In Figure 7.3b, applying mode aggregator would result in a relational attribute $\text{MODE}(\text{Tx.S.S}^{-1}.\text{I.attribute})$ in the model. Clearly, $\text{Tx.S.S}^{-1}.\text{I}$ gives a set of items visited in the current search session. Thus, this model can capture the history of visiting items in the current search session. Even a new user can get recommendation from this model based on his few interactions in the current session.

Type III It is an extension of Type II model with relational attributes obtained from longer slot chains. Such attributes can capture information from previous search sessions and, hence, be applied for new users to recommend existing items that have already appeared in other search results. For example, $\text{MODE}(\text{Tx.I.I}^{-1}.\text{S.S}^{-1}.\text{I.attribute})$ is a relational attribute of slot chain of length 5. $\text{Tx.I.I}^{-1}.\text{S.S}^{-1}.\text{I}$ is a multiset of *Item* objects that are visited in previous *Search* sessions when the target *Item* object is also visited. In fact, this model is equivalent to traditional collaborative filtering where users are recommended the existing items which already have some kind of interactions with the existing users. A Type III model is depicted in Figure 7.3c.

From Figure 7.3, we can see that with the increase in the length of slot chains, the model deals with more and more instances. Thus, model of Type III or the models with very long slot chains actually may not be very suitable for cold systems.

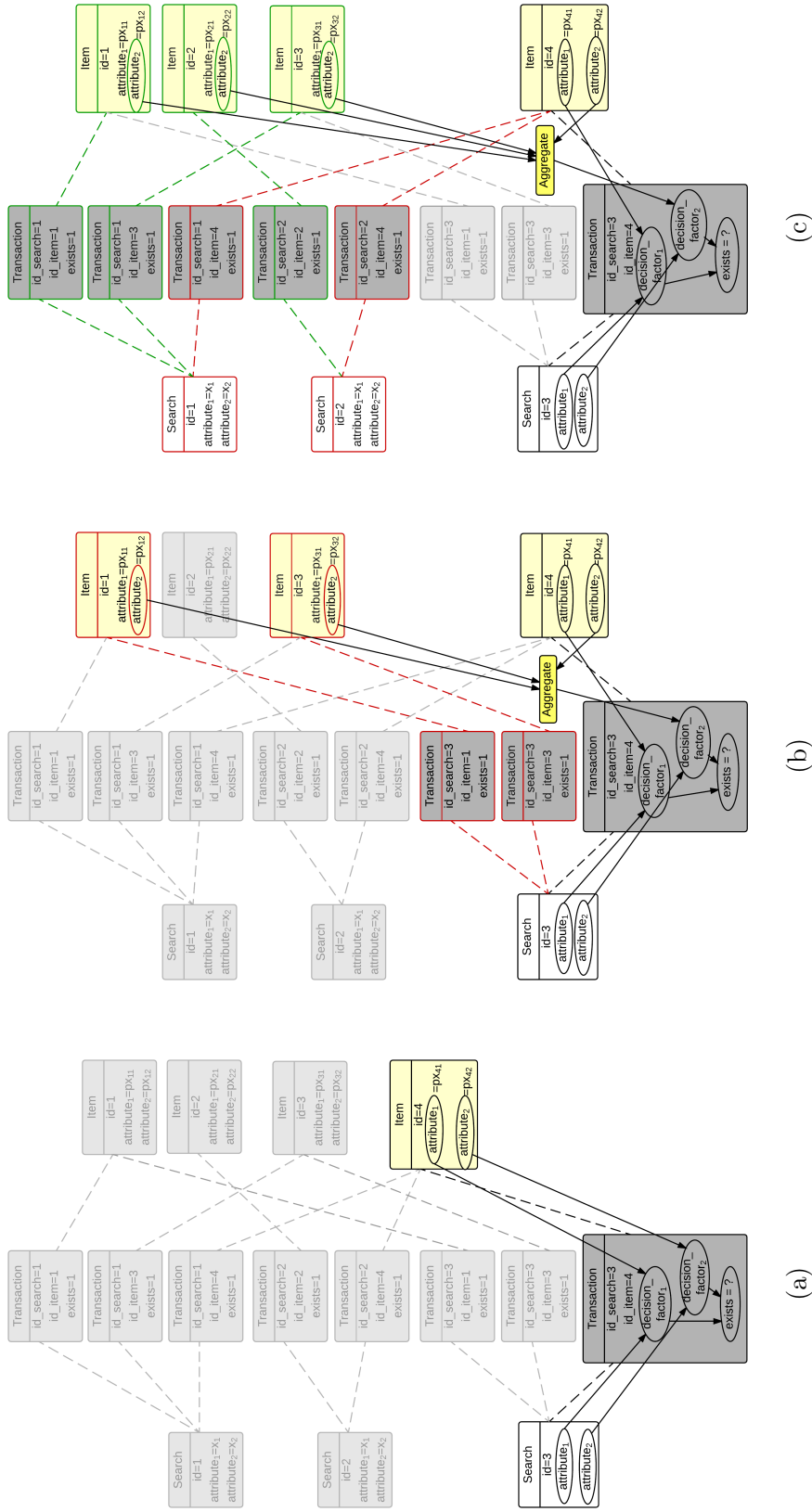


Figure 7.3 – The three types of models: (a) Type I model – Decision factors depend on attributes of slot chain of length 1; (b) Type II model – Decision factors depend on at least one attribute of slot chain of length 3. Here, the attribute $Tx.decision_factor_2$ depends on $Aggregate(Tx.S^{-1}.l.attribute_2)$. The instances of **Item** in red border (all instances i of **Item** such that $i.id \in \{1, 3\}$) are the items visited in the current session; (c) Type III model – It has at least one decision factor that depends on relational attributes of slot chain of length 5. Here, $Tx.decision_factor_2$ depends on $Aggregate(Tx.I^{-1}.S.S^{-1}.l.attribute_2)$. The instances of **Item** in green border (all instances i of **Item** such that $i.id \in \{1, 2, 3\}$) are the items visited in the previous search sessions when the target item ($id = 4$) is also visited.

7.2.4 Examples

In this section, we present some examples of domains where our proposed preference-based recommendation can be applied. We will begin with our target application, Kyzia. As the second example, we will discuss how our approach is applicable for recommending flights. In the third example, we will show how we can re-formulate the work of [Delcroix and Ben Mrad \[2016\]](#) into our [PRM-PrefReco](#).

Example 7.1 Kyzia

Kyzia is an online system for searching real estate properties for buying or renting. A simplified version of the relational schema of Kyzia is shown in Figure 7.4b. Users state their preferences for different characteristics of real estate properties they are looking for, and the system presents them a list of properties matching their preferences. Users' action of clicking on a property is modeled as the Transaction class. Though Kyzia proposes a variety of real estate properties such as apartments, houses, land, parking etc., we consider here only apartments to simplify this example. Users can specify their preferred minimum number of rooms and bedrooms, their minimum and maximum budget, their preferred range of surface area, and their preferences for furnished apartments. Thus, Search.min_nbBedrooms, Search.min_nbRooms, and Search.furnished capture users' preferences for Property.nbRooms, Property.nbBedrooms, and Property.furnished respectively. Similarly, Search.min_surface, and Search.max_surface correspond to Property.surface_area whereas Search.min_budget, and Search.max_budget correspond to Property.price.

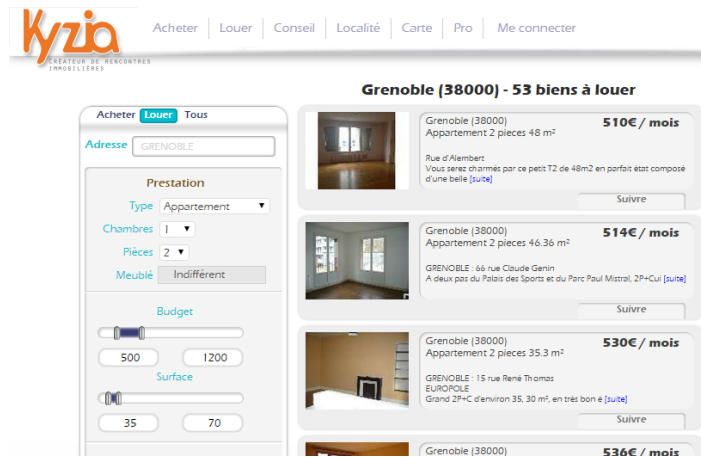
To build a [PRM-PrefReco](#), we choose decision factors according to the *Property* attributes for which users can express their preferences. Here, we identify five⁴ decision factors as shown in the [PRM-PrefReco](#) of Figure 7.4c. In this simple example, we assume that all dependencies between decision factors and *Property* attributes involve slot chains of length 1. For longer slot chains, we would need aggregators. Now, as explained in Section 7.2.1, *CPDs* of *Search* and *Property* attributes are learned from data, and those of decision factors are defined by experts. Table 7.2 shows an example of the *CPD* of the decision factor *Transaction.DF_furnished*, whose parents are *Property.furnished* and *Search.furnished*.

To make recommendations, we need to get the target user's preferences in the form of a *Search* object, and his preferences about decision factors in the form of ranks of decision factors, which can be collected by one of the methods presented in Section 7.2.2. We then instantiate the [PRM-PrefReco](#) of Figure 7.4c for the target user's preferences and all potential properties. Then, we pick those properties that have high probability of getting *Transaction.exists = true* for recommendation. Instantiation of this [PRM-PrefReco](#) for a specific *Search* object and a *Property* object will be similar to the model of Figure 7.3a. If we use longer slot chains in our [PRM-PrefReco](#), we can obtain models of Figure 7.3b or 7.3c. ❖

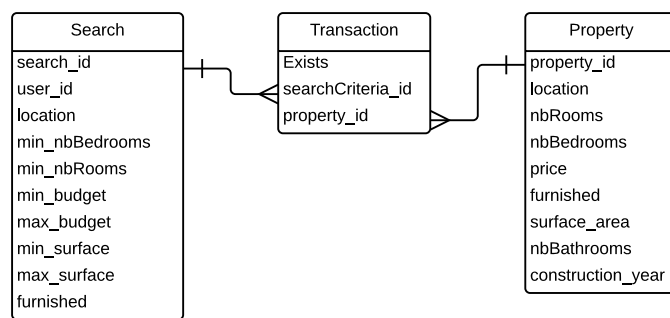
Example 7.2 Flight recommendation

When searching for a flight, users often have not only multiple decision criteria (e.g., price, arrival/departure time, transit duration, number of stops etc.) but also preferences about these criteria. For example, some may be strict about departure time but

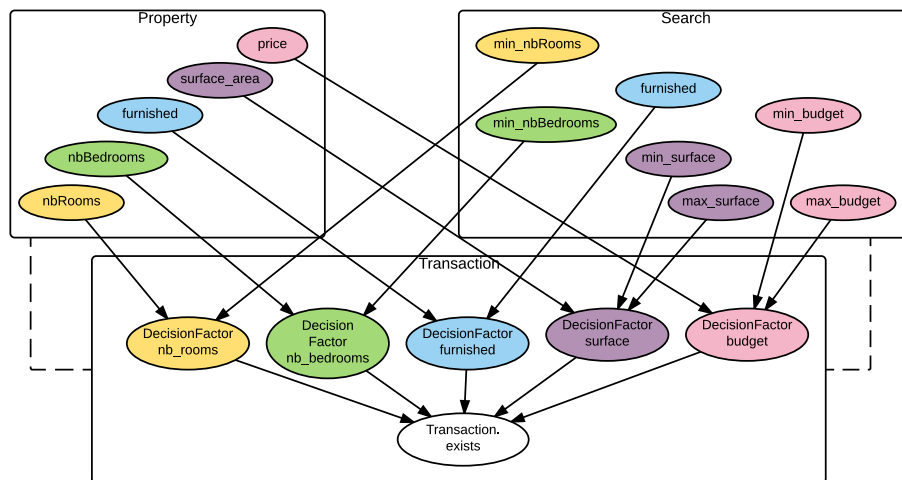
4. There are additional *Property* attributes, e.g., *Property.construction_year*, for which the system does not ask users to state their preferences. These attributes are not used in the [PRM-PrefReco](#) in this example. However, it should be noted that experts may suggest to use such attributes in the [PRM-PrefReco](#). In that case, we will have more decision factors which will be invisible to users.



(a)



(b)



(c)

Figure 7.4 – (a) A screenshot of Kyzia, (b) Relational schema of Kyzia, (c) PRM-PrefReco for Kyzia. Here, colors do not carry any significant meaning and are used only as visual aid to understand the dependencies between decision factors and various Search and Property attributes.

Table 7.2 – CPD of $P(Tx.DF_furnished | Search.furnished, Tx.Property.furnished)$

Tx.Property.furnished	Yes		No	
	Yes	No	Yes	No
Search.furnished				
Positive	0.892	0.402	0.598	0.99
Negative	0.108	0.598	0.402	0.01

some other may be more flexible about departure time but relatively more strict about the number of stops. Thus, our proposed approach for preference-base recommendation is applicable for recommending flights. Relational schema of this domain will be similar to our proposed schema (see Figure 7.1) with three main classes: **Search**, **Flight**, and **Transaction**. The **Flight** class will have attributes like source, destination, number of stops, total duration, airfare, airlines, departure time, arrival time etc., and the **Search** class will have attributes corresponding to these attributes of the **Flight** class. A *PRM-PrefReco* for this domain may have decision factors for budget, the number of stops, flight duration, arrival time, departure time etc., for which users can express their preferences. The decision factor for flight duration, for example, may have as its parents *Flight.duration*, and *Search.max_duration*. Like in the example of *Kyzia*, users will provide their preferences in the form of a **Search** object as well as in the form of ranks of decision factors. Then we generate personalized Bayesian networks for the users to make recommendations. ❖

Example 7.3 Re-formulation of Delcroix and Ben Mrad [2016]’s approach of modeling decision criteria by V-structures in a Bayesian network

Delcroix and Ben Mrad [2016] have presented a work on multi-criteria decision making problem where they model decision criteria in a Bayesian network with the help of V-structures. In their approach, each V-structure is associated with one decision criterion such that the parents in a V-structure represent the factors that can affect decision and the evaluation of an alternative for this criterion, and the child is the level of satisfaction of the criterion. We observe that this is somewhat similar to our approach, and, hence, their problem domains can be re-formulated into our *PRM-PrefReco*. They have presented a problem of choosing cars based on two decision criteria. The *BN* proposed by them for this domain is shown in Figure 7.5a. It is clear that this *BN* has two distinct entities – cars, and users. Users are described by age, sex, *niveauVie* (standard of living) and *taille* (height), and cars by *puissV* (power), *poidsV* (weight), *prixV* (price), *rapportPP* (weight to power ratio) and *indAcceleration* (index of acceleration). The variables *montantDispo* (Budget), and *impAcceleration* (importance of acceleration) are in fact derived from other variables. Finally, *adeqPrix* (adequacy of price), and *satAcceleration* (satisfaction of acceleration) are two decision factors that are supposed to impact users’ decision about choosing a car. Here, we show that our *PRM-PrefReco* is applicable in this domain too.

We can re-formulate their problem as follows. We will have four classes in the schema: **Car** and **User** from the original domain, **Transaction** and **Search** for capturing users’ preferences as usual. All attributes related to users will reside in the **User** class, those related to cars will be kept in the **Car** class, and the decision criteria will be in the **Transaction** class. As the variables *montantDispo* and *impAcceleration* determine the preferred price and the preferred index of acceleration respectively, we add two attributes, *preferred_price* and *preferred_acceleration*, in the **Search** class. Now, the decision factors in the *PRM-PrefReco* will depend on these attributes from the **Search** class and the corresponding attributes from the **Car** class. Alternatively, we can also consider the derived attributes, *montantDispo* and *impAcceleration*, to be the attributes of the **Search** class. However, we preferred to add new attributes in the **Search** class for better understanding. Finally, the **Transaction** class will also contain the target variable *Transaction.exists*, which determines how interesting the **Car** object would be according to the **Search** criteria. All probabilistic dependencies from the original *BN* will be still valid in the *PRM-PrefReco*. After re-formulation, the *BN* of Figure 7.5a can be represented by the *PRM-PrefReco* of Figure 7.5b. Here, the yellow nodes indicate

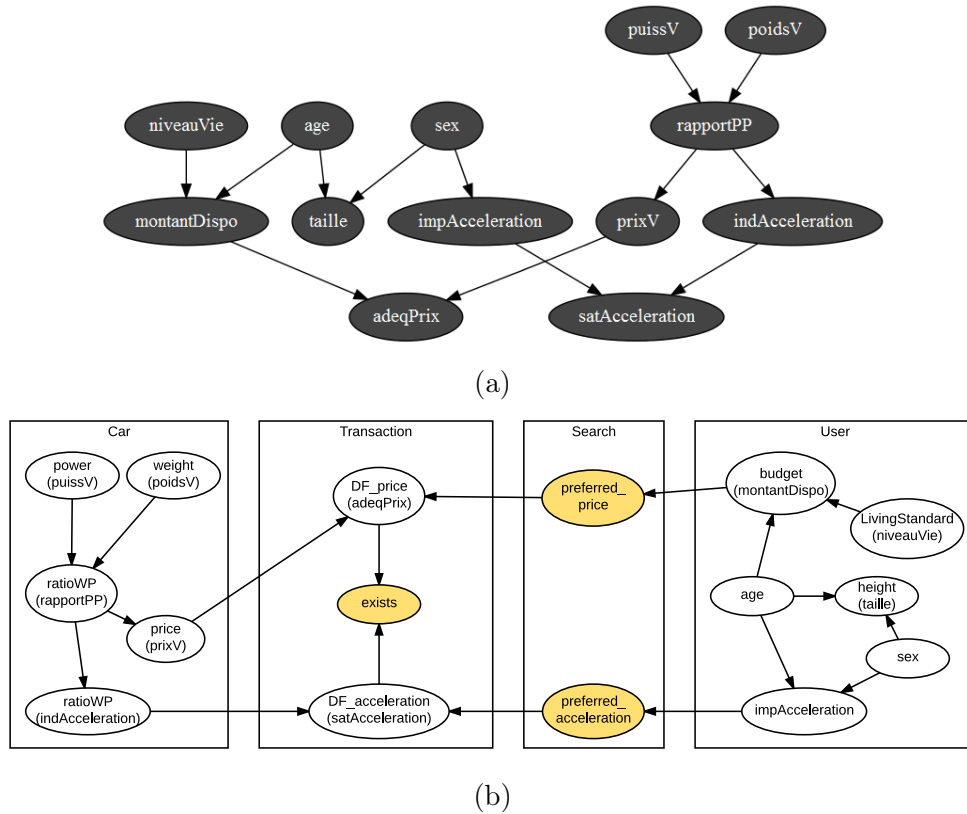


Figure 7.5 – (a) The Bayesian network proposed by [Delcroix and Ben Mrad \[2016\]](#) to model decision criteria by V-structures for the problem of choosing cars, (b) Re-formulating this problem into a [PRM-PrefReco](#)

that they are not present in the original *BN*. Note that if the original system is not an online system, we can achieve only *Type I* recommendation model because there would be no way to track users' navigation history. Otherwise, we can extend it to *Type II* or *Type III* models thereby enhancing the recommendation capacity by including search history of existing users. ❖

7.3 Experiments

We performed our preliminary experiment on a system which is in cold start situation with very few search sessions and a small number of transactions in each search session. In the following sections, we describe our dataset and explain how we performed the experiment and evaluated the model. Then we will present the results and discuss on the findings.

7.3.1 Dataset

The dataset used in this experiment was from Kyzia, a real estate search system developed by DataForPeople. It is a new system and represents cold-start problem well. The relational schema of this system is shown in [Figure 7.4b](#). The system presents the users a list of `Property` objects matching their search criteria. We assume that the users visit the details page of a property if they find it interesting. This information is modeled by the class `Transaction` and were collected by logging the users' click-throughs. The dataset has more than 70,000 real estate properties, around 1400 search

sessions but less than 100 transactions. Ranking method II was used to collect users' preferences. Because the experiment was performed on an evolving system rather than a fixed dataset, we haven't provided the exact size of the dataset here. Users were asked to perform a search on Kyzia website but since the feature of providing their preferences had not been implemented on the website, they were asked to provide their preferences separately to us. Only those search sessions with more than one transaction and whose search criteria preferences are known were kept for evaluating the models. After cleaning, only 7 search sessions were left and the average number of transactions per search session was 3 (2 being the minimum value).

7.3.2 Experiment methodology

A limited number of attributes from `Search` and `Property` classes was used in this experiment. Though there were many more attributes in both classes, only the attributes shown in Figure 7.4b were taken into consideration in order to prove our concept. With the help of experts, all decision factors were identified for our `PRM-PrefReco`. The dependency structure of our `PRM-PrefReco` is shown in Figure 7.4c. If there is enough data, parameters for each attribute can be learned from data. However, available data was not enough to learn parameters of all attributes. Only parameters of `Property` attributes were learned from data. Parameters were assigned to the decision factors with the help of experts. `Search` attributes were assigned a uniform distribution because they are always observed in the final model. The `Transaction` objects were not used during model construction but were kept for evaluation of the models.

For every search session initiated by a user, models were built based on his search criteria and his preferences over those criteria. Type I and Type II models were built from the `PRM-PrefReco`, as explained in Section 7.2.2, by keeping the target attribute `Transaction.exists`, the observed `Search` attributes, and the children (decision factors) and the spouses (i.e., $\text{Tx.K}_i.B$ where \mathbf{K}_i is the desired slot chain that starts from the `Transaction` class and ends at the `Property` class, and B is an attribute of the `Property` class) of the observed `Search` attributes. Type III models were not tested in this experiment because good amount of transaction history is required to create this type of models. While constructing Type I and Type II models, two heuristics, Noisy-OR, and `WSM`, were applied to compute the conditional distribution of the `Transaction.exists`. Thus, the experiment involves two parts – comparison of heuristics used in the model, and comparison of Type I and Type II models.

7.3.3 Evaluation metrics

With the amount of available data, it is not possible to perform extensive off-line evaluation (cf. Section 3.3.1). The evaluation was performed using the available `Transaction` objects in two phases. In each phase, 4 models were created for each search session in the evaluation dataset as explained in Section 7.3.2. Each model was then applied over the properties that were actually shown up to the users during the particular search session, and the properties were ranked based on the probability of existence of transaction given the decision factors, $P(\text{Tx.exists} \mid DF_1, DF_2, \dots, DF_n)$. In the first phase, only top-3 recommendations were considered during evaluation whereas in the second phase, top-5 recommendations were taken. Standard recommendation quality metrics such as precision, recall and F-score were calculated by comparing the top-N properties in this ranked list with the `Transaction` objects for the search session. We

specialize Expressions 3.1 and 3.2 for Type I and Type II models in the following way.

For a search session $s \in \mathcal{I}(\text{Search})$, a set of Transaction objects tx' in the recommendation list and a set of Transaction objects $tx = \{t : t \in \mathcal{I}(\text{Transaction}); t.\text{Search} = s\}$ in the evaluation dataset, the evaluation metrics for Type I model are calculated in the following way:

$$\text{Precision} = \frac{\text{Cardinality}(tx \cap tx')}{N} \quad (7.1a)$$

$$\text{Recall} = \frac{\text{Cardinality}(tx \cap tx')}{\text{Cardinality}(tx)} \quad (7.1b)$$

For Type II model, where the previously visited properties in the current search session are also considered, temporal information from the clickthrough logs were employed to compute the evaluation metrics. For a search session $s \in \mathcal{I}(\text{Search})$, and a set of Transaction objects $tx = \{t : t \in \mathcal{I}(\text{Transaction}); t.\text{Search} = s\}$ in the evaluation dataset, evaluation is performed n times where $n = \text{Cardinality}(tx)$. In each iteration i , only a subset of tx that are visited before the current is used in the model. Average of the metrics in all iterations gives the overall metrics for the particular search session as follows.

$$\text{Precision} = \text{Average} \left(\frac{\text{Cardinality}(tx \cap tx'_i)}{N} \right) \quad (7.2a)$$

$$\text{Recall} = \text{Average} \left(\frac{\text{Cardinality}(tx \cap tx'_i)}{\text{Cardinality}(tx)} \right) \quad (7.2b)$$

where $i = 1 \dots n$, and tx'_i is a set of Transaction objects in the recommendation list in i^{th} iteration.

7.3.4 Results and discussion

Metrics obtained from the experiment are presented in Table 7.3. In the first part of the experiment, two heuristics, Noisy-OR, and WSM, were compared. As we can see from Table 7.3, both of these methods produced similar results. With limited data, the choice of approximation algorithms did not seem to have big impact on the result.

The second part of the experiment involved the comparison of types of models. When N was changed from 3 to 5 in top- N evaluation approach, a slight increase in precision but significant increase in recall was observed. However, in both the cases, Type II models performed better than Type I models did. This signifies that the properties are better ranked in the Type II models. Top-3 recommendation metrics for Type II model were somewhat closer to top-5 recommendation metrics for Type I model. Recall of Type II model is quite good when top-5 models are taken into account for evaluation.

Off-line evaluation in cold systems may not actually give the clear picture of the performance of the model because of insufficiency of test data. However, standard datasets for evaluating recommender systems, such as the ones listed in Table 3.1, are not applicable in our context because they lack users' preferences about items' characteristics. Comparing our model with another recommendation algorithms is also not possible because of the lack of user profiles in our system and the incapability of integrating users' preferences in existing algorithms. Therefore, in this scenario, better

Table 7.3 – Evaluation result

Method	Reco Type	Precision		Recall		F-score	
		Type I	Type II	Type I	Type II	Type I	Type II
NoisyOR	Top-3	0.38 ± 0.39	0.46 ± 0.31	0.38 ± 0.38	0.59 ± 0.25	0.36 ± 0.37	0.47 ± 0.26
WSM	Top-3	0.38 ± 0.39	0.46 ± 0.31	0.38 ± 0.38	0.59 ± 0.25	0.36 ± 0.37	0.47 ± 0.26
NoisyOR	Top-5	0.42 ± 0.36	0.40 ± 0.30	0.68 ± 0.40	0.76 ± 0.29	0.48 ± 0.34	0.49 ± 0.27
WSM	Top-5	0.39 ± 0.37	0.45 ± 0.32	0.60 ± 0.37	0.76 ± 0.21	0.44 ± 0.35	0.52 ± 0.26

approach to assess the model is to perform online evaluation where users interact with the system and take the next step based on the result they get from the model. Evaluating the model in such a way could also reveal how novel or serendipitous the results are. However, it was expensive to evaluate our system that way. Thus, we had to stay with the off-line evaluation approach. It is also worth noticing that the quality of a model might have been affected by hidden causes that can affect users' decision. For instance, users might have made decision for some properties based on the quality of the pictures posted in the announcements, but such information have not been included in our model and are not in the scope of this work.

7.4 Conclusion

In this chapter, we have presented a [PRM](#)-based personalized recommender system for feature-based systems. Our approach is suitable for recommending expensive or less frequently purchased items or for the systems which need to recommend to users the items that best match users' short-term preferences about items. We have exploited the relational information present in such domains to address the recommendation problem as the task of predicting whether there would an interaction between an item and a search session. We have proposed a generic [PRM](#) for modeling recommendation problem from users' preferences in feature-based systems, and have explained how to initialize this [PRM](#) over a relational skeleton for a specific user considering his preferences about various decision criteria to obtain a Bayesian network specialized for that particular user. We then recommend the items that have high probability of having interactions in the search session. Using our approach, content-based, collaborative-filtering and hybrid models can be achieved from the same [PRM](#) structure by varying the length of slot chains. We have also presented some example domains where we can apply our approach of recommendation. Our preliminary experiment on a real-world dataset has shown that we were able to get a good result even with small dataset using our approach.

PRM with Spatial Attributes (PRM-SA)

Contents

8.1	Introduction	106
8.2	Definitions	106
8.3	Learning PRM-SA	112
8.4	Evaluation of PRM-SA learning algorithms	116
8.4.1	Evaluation strategy and metrics	117
8.4.2	Generation of PRM-SA benchmarks	117
8.5	Experimental study	122
8.5.1	Methodology	123
8.5.2	Results and discussion	126
8.6	PRMs-SA in recommender systems	131
8.7	Conclusion	132

8.1 Introduction

Spatial data analysis has been a popular topic in the studies of environment, ecosystems, population, communities, image processing etc. There is a growing trend of using spatial information in a wide range of application domain. Recommender systems (RSs) have also started adopting spatial dimensions for making better recommendations, as we have seen in Chapter 6. The increased use of spatial data has augmented the need for analysis of spatial data. Spatial data analysis is certainly not a new field (Goodchild [2010]). Several methods have been devised to extract patterns from spatial data and understand underlying phenomena. Among various machine learning techniques used for spatial data analysis, Bayesian networks (BNs) have been commonly used in various domains for studying spatial phenomena. For example, Wilkinson et al. [2013], Li et al. [2010] and Huang and Yuan [2007] use BNs for modeling spatial information in environmental studies, Park et al. [2007] use BNs in spatial recommender systems, Cano et al. [2004] have applied BNs in meteorology, Walker et al. [2005] have used BNs in geographical information retrieval, and so on. Recently, object-oriented Bayesian networks have also been used for modeling spatial interactions (Wilkinson et al. [2013]). However, these approaches are mostly dedicated for specific problems, and many of them do not capture spatial dependencies well.

Most of the techniques of spatial data analysis work with flat representation of data. However, real-world applications are generally conceptualized in terms of objects and relations between them, and, hence, data need to be transformed into the required flat format before applying those methods. Besides, analysis of spatial information usually involves the study of interaction between spatial objects. In such relational domains, PRMs can be employed to learn probabilistic models. However, standard PRMs do not support spatial objects. Therefore, we aim at integrating spatial information into PRMs to enable them to handle spatial objects too. Our motivation is also driven by Malerba [2008]’s perspective on spatial data mining in relational domain. Malerba [2008] argue that multi-relational setting is the most suitable for spatial data mining problems and also mention the possibility of using PRMs with spatial relational databases. In this chapter, we propose a new extension of standard PRMs to support spatial objects. Our proposed model, PRM-SA, provides a general way to incorporate spatial information into a PRM and model spatial dependencies. We also provide algorithms for learning such models from data. In this thesis, we are mainly concerned with geographically referenced objects.

This chapter is organized as follows. In Section 8.2, we will present our proposed model with illustrations. We will also point to the possibility of modeling spatial autocorrelation (Griffith [1992]) with our model. In Section 8.3, we will explain how this model can be learned from data, and propose three algorithms for learning the structure of the model. In Section 8.4, we will explain how to evaluate this model. Finally, we will present experimental results in Section 8.5.

8.2 Definitions

We propose to incorporate in PRMs the vector representation of spatial objects, where a spatial object is described by its location in space in terms of geometry and its attributes as defined in Chapter 5. The attributes of a spatial object can form descriptive attributes in the relational schema whereas its geometry cannot be incorporated as descriptive attributes because the geometry of spatial objects is represented by a set

of points (vertices or coordinates). Thus, we coin a term *spatial geometry attribute* or simply *spatial attribute* to describe such attributes to use them in a PRM.

Definition 20 *Spatial geometry attribute*

An attribute is a spatial geometry attribute if its value s is a sequence of pairs of coordinates in geographic coordinate reference systems and defines the geometry of its class. i.e. $s = ((x, y)_n : n \in \mathbb{N})$ where x is called longitude, y is called latitude and n is the cardinality of s . Here, s represents

- a point geometry if $n = 1$,
- a line geometry if $n \geq 2$ and $s_1 \neq s_n$,
- a polygon geometry if $n \geq 2$ and $s_1 = s_n$. ■

Definition 21 *Spatial class*

Let $\mathcal{SA}(X)$ be the set of spatial geometry attributes in a class X . A class X is a spatial class if it contains spatial geometry attributes, i.e. if $\mathcal{SA}(X)$ is not empty. ■

Example 8.1 Restaurant-User-Cuisine schema with a spatial attribute

To illustrate these concepts, we extend the relational schema of Figure 2.1a with a spatial geometry attribute in the class *Restaurant* as shown in Figure 8.1a. Here, the spatial attribute *Restaurant.location* represents the location of the objects of the spatial class *Restaurant*. ♦

As the set of possible values of a spatial attribute is infinite, conditional probability distributions associated with spatial attributes would be very big. It demands an extensive computation for learning as well as inference and this is practically too difficult to achieve. Therefore, we propose to partition this set into a finite number of disjoint subsets with the help of a *spatial partition function*. Each partition is then represented by a class, which we call a *spatial partition class*, and a reference slot (we call it a *spatial reference slot* or *spatial ref. slot*) that refers to the objects of the partition class is added in the corresponding spatial class. Partition functions are responsible for creating the objects of partition classes and mapping the values of a spatial attribute to their corresponding partitions.

Definition 22 *Spatial partition function, Spatial partition class*

Let $X.SA$ be a spatial geometry attribute of a spatial class X . We define a spatial partition function $f_{sa} : X.SA \rightarrow Range[f_{sa}]$ where $Range[f_{sa}]$ is a finite set of spatial partitions represented by a spatial partition class P_{XSA} . Thus, f_{sa} associates each $sa \in Domain[X.SA]$ to an object of P_{XSA} determined by the function itself. ■

Partition functions essentially map a spatial object to a region such that spatial objects meeting some partitioning criteria are grouped together. Such mapping can be achieved in many ways. One way is to use regular square or hexagonal (honeycomb) grids or to apply spatial aggregation operators (cf. Section 5.4) to partition the spatial region. Partitions can also be created by using standard, publicly available knowledge such as administrative boundaries. In the absence of such knowledge, *spatial clustering algorithms* can be used. Han et al. [2001] have presented a survey on several spatial clustering methods. Some clustering algorithms, such as K-means, require users to provide the number of clusters/regions and some others, such as DBSCAN, can determine the number of clusters themselves. It should be noted that granularity of partitioning

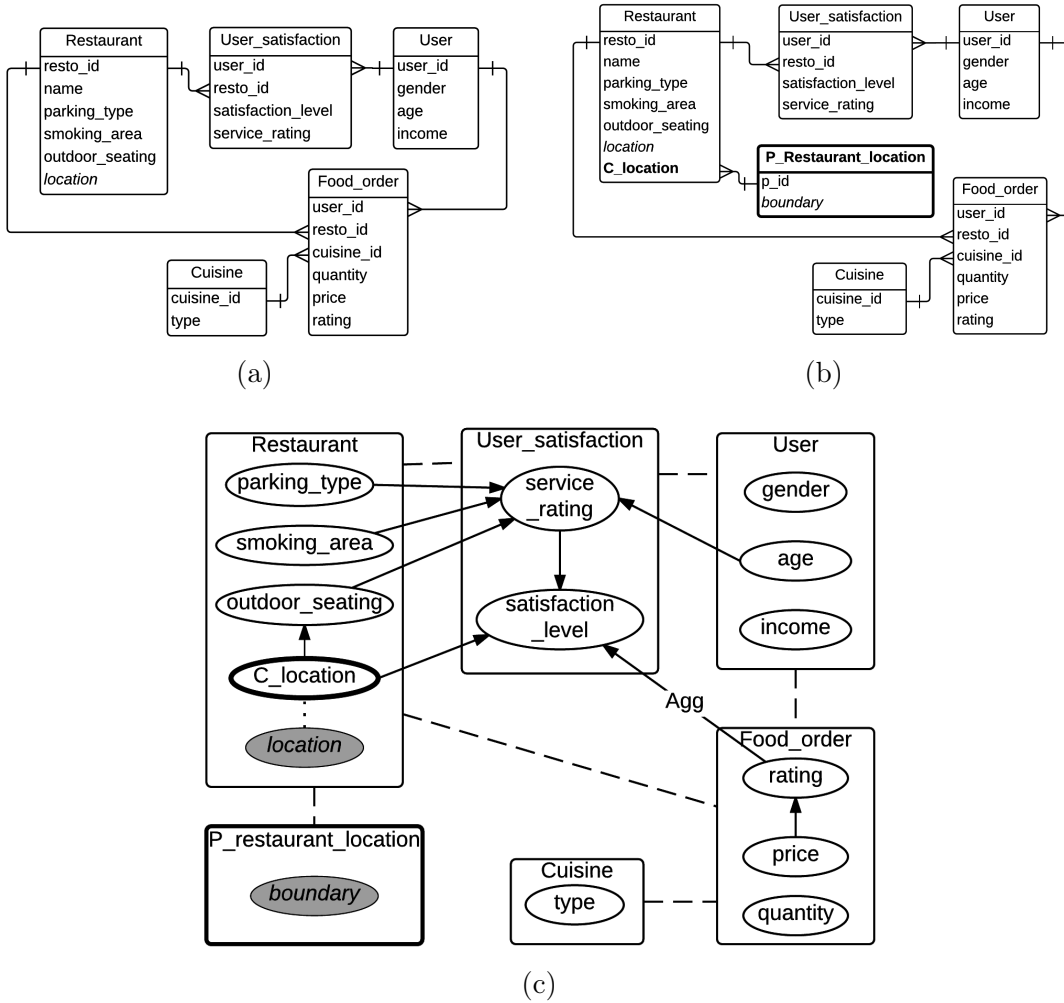


Figure 8.1 – (a) An example of a relational schema with a spatial attribute `Restaurant.location`, which cannot be handled by standard PRMs. (b) The relational schema adapted for the spatial attribute `Restaurant.location` as proposed in Definition 23.1. Here, spatial attributes are shown in italicized font and the added spatial reference slot and the spatial partition class are shown in boldface. (c) A PRM-SA as proposed in Definition 23. The gray nodes are spatial attributes and the one with thick border is the spatial reference slot associated with the spatial attribute `Restaurant.location`. The dotted line between `Restaurant.location` and `Restaurant.C_location` indicates that the spatial attribute and the spatial ref. slot are associated through a spatial partition function.

methods depends on the context of the problem. Also note that when using knowledge to create partitions, we may have access to some extra information. Such information can be considered as descriptive attributes of partition classes. The introduction of partition classes enables us to implement hierarchical clustering as well because partition classes can also contain spatial attributes, which can be further partitioned thereby creating a hierarchy of spatial partition classes.

Example 8.2 Adapted version of Restaurant-User-Cuisine schema

We refer to Figure 8.1a for examples. As the spatial attribute `Restaurant.location` can take infinitely many values, we define a partition function to partition its possible values into a finite set of spatial partitions. Thus, we add a spatial partition class `P_Restaurant_location` that represents the spatial partitions of `Restaurant.location`. The objects of this spatial partition class will then be referenced in the spatial class `Restaurant`

by the spatial reference slot `Restaurant.C_location`. Figure 8.1b shows the relational schema adapted for the spatial attribute. Here, we have assumed that the spatial partition class will have an additional attribute called `boundary`, which is again a spatial attribute. In this example, we can assume that this attribute represents the convex hull formed by all the locations mapped to the particular partition object. However, the attributes present in the partition classes depend on the context. If we use the information about administrative boundaries of cities or regions, `P_restaurant_location.boundary` might represent the boundary of the specified location, and we might even have extra information about the partitions such as population, average income, demographic structure etc. of the location. If hierarchical administrative division is available, we can incorporate hierarchical partitions by further partitioning `P_restaurant_location.boundary` and adding another partition class, say `P_restaurant_location_boundary` (not shown in the figure). ❖

We now define our model, which is based on standard PRMs and supports spatial data.

Definition 23 PRM with spatial attributes (PRM-SA)

Let $\mathcal{A}(X)$ and $\mathcal{SA}(X)$ denote the set of descriptive attributes and geometry attributes respectively in class X .

For each spatial class $X \in \mathcal{X}$ such that $\mathcal{SA}(X) \neq \emptyset$ and for each geometry attribute $SA \in \mathcal{SA}(X)$, we define the following:

- a new partition class P_{XSA} ,
- a partition function $f_{sa} : SA \rightarrow P_{XSA}$ that creates instances of P_{XSA} associating each $sa \in \text{Domain}[SA]$ to one of the instances of P_{XSA} , and
- a new spatial reference slot $X.C_{SA}$ associated with f_{sa} .

Then, we adapt the relational schema for spatial attributes and define the probabilistic model in the following way.

Definition 23.1 Adapted relational schema

The relational schema is adapted for spatial attributes by adding P_{XSA} and $X.C_{SA}$ associated with f_{sa} for each spatial class $X \in \mathcal{X}$ and for each geometry attribute $SA \in \mathcal{SA}(X)$.

Definition 23.2 Probabilistic model of a PRM-SA

Let \mathcal{P}_{SA} and \mathcal{C}_{SA} be the set of partition classes and the set of added spatial reference slots respectively. Then, for each class $X \in \{\mathcal{X} \cup \mathcal{P}_{SA}\}$ and each attribute $A \in \{\mathcal{A}(X) \cup \mathcal{C}_{SA}(X)\}$, we have

- a set of parents $Pa(X.A) = \{U_1, \dots, U_l\}$, where each U_i has the form $X.B$ or $\gamma(X.K.B)$, where B is an attribute of any class, K is a slot chain and γ is an aggregate of $X.K.B$,
- a legal conditional probability distribution CPD, $P(X.A \mid Pa(X.A))$. ■

A PRM-SA that corresponds to the schema in Figure 8.1a is shown in Figure 8.1c. Here, the gray nodes are spatial attributes and the nodes/classes with thick border are the ones that are not present in the original relational schema. The probabilistic dependencies shown in the examples are hypothetical.

Probabilistic inference is performed on a ground Bayesian network (GBN) obtained by instantiating a PRM-SA for a given relational skeleton. Here, the skeleton must

also include the objects of spatial classes. Given such a relational skeleton, a PRM-SA induces a **GBN** that specifies probability distributions over the attributes of the objects. Here, we need to ensure that the probability distributions are *coherent*, i.e. the sum of probability of all instances is 1. In Bayesian networks, this requirement is satisfied if the dependency graph is acyclic (Getoor [2001]). Following Getoor [2001]’s approach, we consider *instance dependency graph* to check whether the dependency structure \mathcal{S} of a PRM-SA is acyclic relative to a given relational skeleton. Due to the presence of spatial attributes, standard instance dependency graphs need to be redefined with some adaptations for PRM-SA. It follows from Getoor [2001]’s proof that the dependency structure \mathcal{S} for a PRM-SA is guaranteed to be acyclic for the given relational skeleton if the corresponding instance dependency graph is acyclic.

Definition 24 Instance dependency graph (IDG)

The instance dependency graph G_{σ_r} for a PRM-SA Π with partition classes \mathcal{P}_{SA} and a relational skeleton σ_r is defined as follows. For each object $x \in \sigma_r(X)$ in each class $X \in \{\mathcal{X} \cup \mathcal{P}_{SA}\}$, we have the following nodes: a node $x.A$ for each descriptive attribute $X.A$, and a node $x.C_{SA}$ for each spatial reference slot $X.C_{SA}$. The graph has the following edges:

1. Type I edges: For each formal parent of $x.A$, $X.B$, we introduce an edge from $x.B$ to $x.A$.
2. Type II edges: For each formal parent $X.K.B$, and for each $y \in x.K$, we define an edge from $y.B$ to $x.A$.
3. Type III edges: For any spatial attribute $x.SA$ in each spatial class $X \in \mathcal{X}$, we define an edge $x.SA \rightarrow x.C_{SA}$.
4. Type IV edges: For any attribute $p.A$ in each spatial partition class $P \in \mathcal{P}_{SA}$ and $p \in \sigma_r(P)$, we add an edge $p.A \leftarrow x.A$ if $P.A$ is derived from $X.A$. ■

Again, it is obvious from Getoor [2001]’s proof that the probabilistic model of a PRM-SA is coherent for *any* relational skeleton if the corresponding *class dependency graph* is acyclic. Here again, because of the presence of spatial attributes, we redefine class dependency graph for PRM-SA.

Definition 25 Class dependency graph (CDG)

The class dependency graph G_{Π} for a PRM-SA Π is defined as follows. The dependency graph has the following nodes: a node for each descriptive attribute $X.A$, and a node for each spatial reference slot $X.C_{SA}$. The graph has the following edges:

1. Type I edges: For any attribute $X.A$ and its parent $X.B$, we introduce an edge from $X.B$ to $X.A$.
2. Type II edges: For any attribute $X.A$ and its parent $X.K.B$, we introduce an edge from $Y.B$ to $X.A$, where $Y = \text{Range}[X.K]$.
3. Type III edges: For any spatial attribute $X.SA$ in each spatial class $X \in \mathcal{X}$, we define an edge $X.SA \rightarrow X.C_{SA}$.
4. Type IV edges: For any attribute $P.A$ in each spatial partition class $P \in \mathcal{P}_{SA}$, we add an edge $P.A \leftarrow X.A$ if $P.A$ is derived from $X.A$. ■

Figure 8.2 shows the class dependency graph for the PRM-SA in Figure 8.1c. Because this graph is acyclic, the dependency structure in Figure 8.1c is guaranteed to be acyclic regardless of relational skeleton.

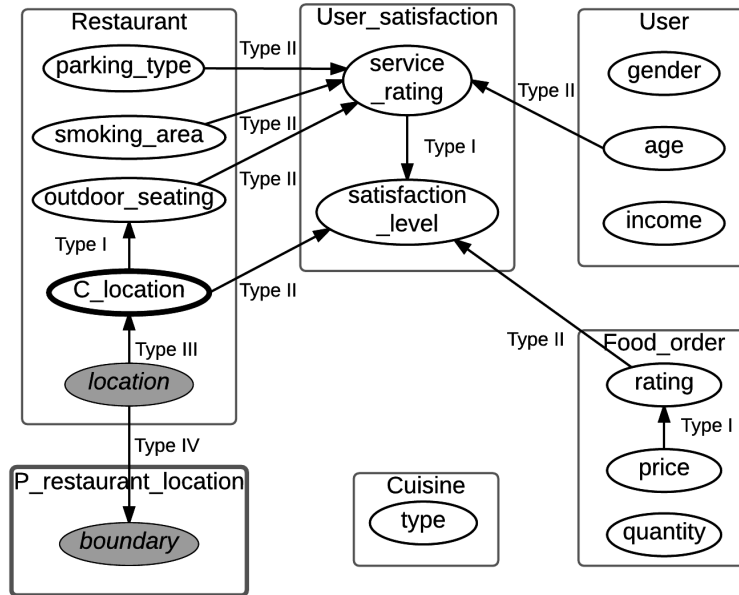


Figure 8.2 – The class dependency graph for the PRM-SA in Figure 8.1c. Because there is no cycle in this graph, we can conclude that the dependency structure in Figure 8.1c is acyclic for any relational skeleton.

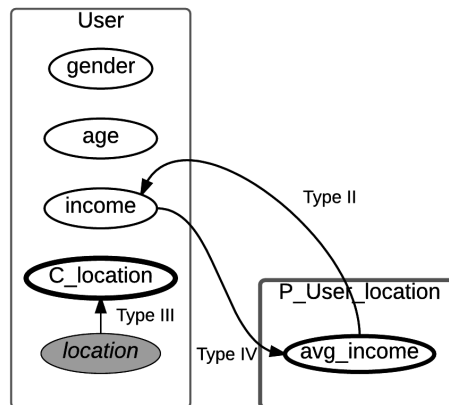


Figure 8.3 – An example of a class dependency graph with a cycle. Because of the presence of a cycle, we can conclude that the dependency structure cannot be guaranteed to be acyclic for any relational skeleton.

Example 8.3 Invalid PRM-SA (due to a cyclic dependency)

Let us consider another hypothetical example where there is an attribute $User.location$ and a partition class $P_User_location$ that represents the partitions of users' location. Suppose there exists a dependency that says the income of a user depends on the average income of the users in his community, i.e. users with similar income tend to live in the same community. In this case, the dependency structure of a PRM-SA will have an edge from $P_User_location.avg_income$ to $User.income$. Although there is no cycle in the structure, it is incoherent because the class dependency graph of this structure contains a cycle as shown in Figure 8.3. This is due to the type IV edge that is added because $P_User_location.avg_income$ is derived from $User.income$. Thus, this PRM is not a valid one. ❖

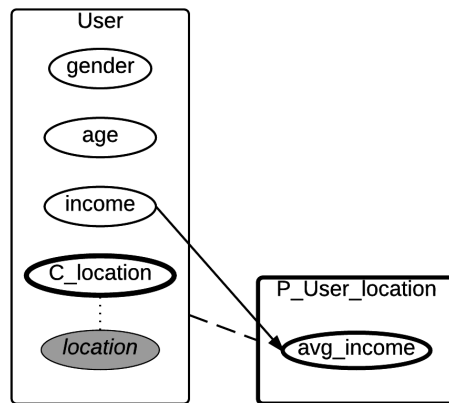


Figure 8.4 – An example of a dependency structure that models the dependency of an attribute with the aggregated value of the same attribute of spatial objects in the same cluster. This concept can be used to model spatial autocorrelation.

PRM-SA and spatial autocorrelation Researchers advocate the consideration of autocorrelation (cf. Section 5.3.2) in spatial data analysis. However, autocorrelation cannot be modeled directly in PRMs because of acyclicity constraint (Neville and Jensen [2003]). An attribute of an instance depending on the same attribute of neighboring instances would create a cycle because of the fact that neighborhood is a mutual concept. Due to this reason, we do not model spatial autocorrelation directly. Nevertheless, it is still possible to extend our model to enforce the modeling of spatial autocorrelation by adding aggregated descriptive attributes in partition classes with a special constraint that the aggregated attribute must always be a child to ensure acyclicity. Example 8.3, in fact, tries to model spatial autocorrelation. Modeling spatial autocorrelation of users’ income (i.e. users’ income depending on the income of neighboring users) would create a cycle in the dependency structure. To avoid this, we could add an attribute *avg_income* in the *P_User_location*, which is, in fact, the average (aggregation) of *User.income*. *P_User_location.avg_income* would, then, be enforced to be a child of *User.income* as shown in Figure 8.4, otherwise we may come across the situation as in Figure 8.3. This models the dependency of an attribute with the aggregated value of the same attribute of spatial objects in the same cluster. RMNs could be another solution to model this spatial autocorrelation because dependencies are represented by an undirected graph in this type of probabilistic graphical models. However, learning RMNs is much more complex than learning PRMs.

8.3 Learning PRM-SA

As with standard PRMs, learning a PRM-SA involves the tasks of parameter estimation and structure learning. Parameters of a PRM-SA can be learned in the same way as for a regular PRM. However, the instances of all partition classes must be included in the instantiation of the relational schema.

As for structure learning, following Friedman et al. [1999]’s approach, we apply relational greedy search algorithm to explore the search space of candidate structures and evaluate legal candidate structures using *score-based* methods. However, due to the introduction of partition classes along with partition functions, we need to adapt the search algorithm to deal with partitions. Here, we come across two situations – 1) when the number of partitions of the spatial attribute (i.e. $Cardinality(Range[f_{sa}])$,

Algorithm 10 Generate_Neighbors (Naïve Approach)

Input: A PRM, $\Pi = \langle \mathcal{R}, \mathcal{S} \rangle$; Slot chain length, SL ; Available aggregators, Agg ,
Step size, $step$, Maximum cardinality, k_{max} **Output:** Neighbors of S , \mathcal{N}

- 1: $\mathcal{N} \leftarrow \text{Generate_Neighbors_For_PRM}(\Pi, SL, Agg)$
 - 2: **for** each $n \in \mathcal{S}$ **do**
 - 3: **if** n is a spatial ref. attribute **then**
 - 4: $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Increase_k}(\mathcal{S}, n, step, k_{max})$
 - 5: $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Decrease_k}(\mathcal{S}, n, step, k_{max})$
 - 6: **end if**
 - 7: **end for**
-

Algorithm 11 Increase_k Operation

Input: A DAG, \mathcal{G} ; Target node, $X.C_{SA}$; Step size, $step$; Maximum cardinality, k_{max} **Output:** New DAG, \mathcal{G}' ;

- 1: $k \leftarrow$ Current number of states of $X.C_{SA}$
 - 2: **if** $k \leq k_{max} - step$ **then**
 - 3: $\mathcal{G}' \leftarrow \mathcal{G}$
 - 4: $k' \leftarrow k + step$ \triangleright New number of states of $X.C_{SA}$
 - 5: Set the number of states of $X.C_{SA}$ in \mathcal{G}' to k'
 - 6: **end if**
-

let's denote it by k_{sa}) is known, and 2) when it is unknown. In the former case, standard relational greedy search algorithm can be applied to explore the search space. However, the situation is complicated in the latter case where the number of partitions needs to be determined by the algorithm. We present two approaches to learn a PRM-SA when k_{sa} is unknown.

Naïve approach

A naïve way (see Algorithm 10) is to add new operators *increase_k* and *decrease_k*, which increases or decreases the number of partitions respectively, and use these operators along with *add*, *delete* and *revert edge* operators of standard greedy search algorithm (cf. Algorithm 3) to find neighborhood of a structure. The best scoring structure is then chosen among the candidate structures. To simplify the computation, we use a *decomposable* Bayesian scoring function (cf. score-and-search approach in Section 2.2.2). So, when comparing candidate structures, it is sufficient to compute the score of only those variables whose score is affected by the particular operation. For example, as *increase_k* or *decrease_k* operations affect the score of the target variable and its children, we can compute the gain in score as in Equation 8.1.

$$\Delta S_{C_{SA}}(k, k') = \sum_{Y \in \{C_{SA}\} \cup Adj(C_{SA})} [Score_{k'}(Y, Pa(Y)) - Score_k(Y, Pa(Y))] \quad (8.1)$$

where $Adj(X) = Pa(X) \cup Ch(X)$, is a set of parents and children of the node X .

Adaptative approach

In our second approach, we separate the tasks of greedy search over candidate structures and finding the optimal number of partitions of spatial attributes. The

Algorithm 12 Decrease_k Operation**Input:** A DAG, \mathcal{G} ; Target node, $X.C_{SA}$; Step size, $step$ **Output:** New DAG, \mathcal{G}' ;

-
- 1: $k \leftarrow$ Current number of states of $X.C_{SA}$
 - 2: **if** $k > step + 1$ **then** \triangleright New number of states must be greater than 1
 - 3: $\mathcal{G}' \leftarrow \mathcal{G}$
 - 4: $k' \leftarrow k - step$ \triangleright New number of states of $X.C_{SA}$
 - 5: Set the number of states of $X.C_{SA}$ in \mathcal{G}' to k'
 - 6: **end if**
-

basic idea is to pick the best scoring structure among candidate structures, and then find the optimal number of partitions of the spatial attributes in this structure if this structure is obtained by changing (i.e. adding, deleting or reverting) an edge that involves a *spatial reference slot*.

An important point to note here is that if a spatial reference slot appears with other spatial reference slots in the local score terms of the scoring function (while finding the optimal number of partitions), we need to vary the cardinality of the set of all spatial reference slots that appear together with the target spatial reference slot. We demonstrate such situation in Example 8.4.

Example 8.4 Multiple spatial ref. attributes

Suppose the PRM-SA in Figure 8.5a is obtained at some point during structure learning by adding an edge from $Class_A.C_location_a$ to $Class_A.a1$. Because the added edge contains a spatial reference slot, we need to find the best number of partitions for this node. The score S of this structure is

$$\begin{aligned}
S = & S(Class_A.a1 \mid Class_A.C_location_a) \\
& + S(Class_A.C_location_a) + S(Class_B.b1 \mid Class_C.c1) \\
& + S(Class_B.b2 \mid Class_A.C_location_a, Class_B.C_location_b) \\
& + S(Class_C.c2 \mid Class_B.C_location_b, Class_C.C_location_c) \\
& + S(Class_C.c1) + S(Class_D.d1 \mid Class_D.C_location_d) \\
& + S(Class_D.C_location_d \mid Class_C.c2) \tag{8.2}
\end{aligned}$$

From Equation 8.2, it is clear that changing the number of partitions of the node $Class_A.C_location_a$ affects the score of the nodes $Class_A.a1$ and $Class_B.b2$. However, to find the best score for the node $Class_B.b2$, we need to find the optimal number of partitions for $Class_B.C_location_b$ too because it appears with the spatial reference slot $Class_A.C_location_a$ in local score terms of the scoring function. $Class_B.C_location_b$ also appears with $Class_C.C_location_c$ in the scoring function. As a result, partition functions of all of the three spatial ref. slots need to be optimized together. \blacklozenge

Identifying the set of spatial ref. attributes for optimization To visualize the situation such as the one in Example 8.4, we propose to *moralize* the structure and find *2-vertex cliques* of spatial reference slots. Those connected through the cliques form a set of variables whose cardinality should be varied altogether when optimizing the number of partitions. This concept is listed in Algorithm 13.

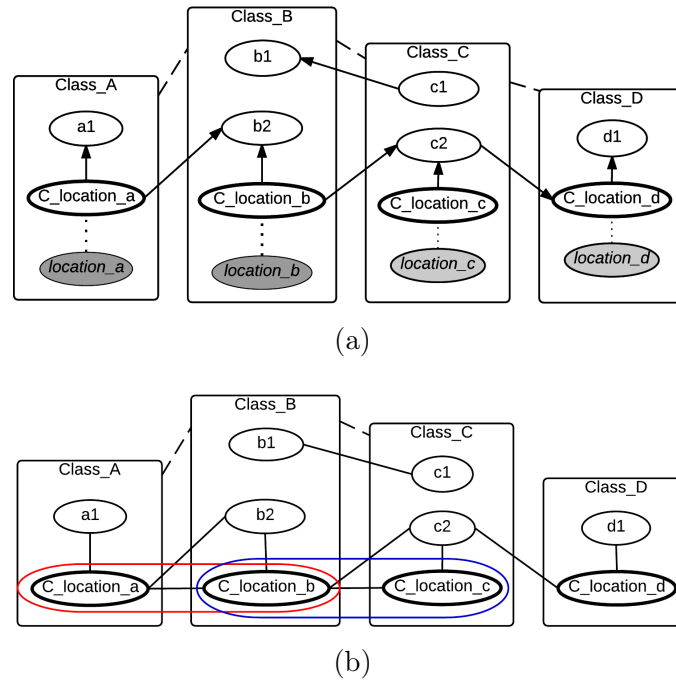


Figure 8.5 – (a) A PRM-SA with multiple spatial attributes. (b) The corresponding moralized graph used to identify the set of partition functions to be optimized. Here, the pairs of spatial reference slots $\{\text{Class_A.C_location_a}, \text{Class_B.C_location_b}\}$ and $\{\text{Class_B.C_location_b}, \text{Class_C.C_location_c}\}$ form 2-vertex cliques. Because these two cliques are connected, we need to find the optimal number of partitions for these three spatial ref. slots together.

To identify the set of spatial ref. attributes that need to be optimized together in Example 8.4, we can moralize the dependency structure of the PRM-SA as shown in Figure 8.5b and find out all 2-vertex cliques of spatial reference slots. Note that spatial partition classes associated with each spatial attribute exist there but they are not shown in the figure (to avoid cluttered diagrams). The three spatial reference slots $\text{Class_A.C_location_a}$, $\text{Class_B.C_location_b}$ and $\text{Class_C.C_location_c}$ in Figure 8.5b are connected through cliques. Therefore, we need to find the best number of partitions for these three spatial ref. slots altogether.

An interesting property of our adaptive approach of learning PRM-SA is that because of the separation of greedy search and partition size optimization, we can come up with different heuristics that involve different combinations of these two tasks. For example, we can perform one operation (add, delete or revert) of greedy search and then find the cardinality of spatial reference attributes, or we can find the optimal cardinality of all spatial reference attributes after a complete greedy search and so on. We present three variations of our structure learning approach in Algorithms 14, 15, and 16. In Algorithm 14, we find the optimal cardinality of all spatial reference attributes after a complete greedy search for the given slot chain length. In Algorithm 15, we find the optimal cardinality of all spatial reference attributes in each iteration of greedy search algorithm for the given slot chain. In the third version (Algorithm 16), during neighborhood generation, add, delete and revert operations are followed by optimization of cardinality of spatial reference attributes (that are involved in the operation) so that these operations always result in edges with spatial reference attributes with optimal number of states.

Algorithm 13 Find_Structure_With_Best_k

Input: A DAG, \mathcal{G} ; Target spatial ref. attribute node, n ; Scoring function, $Score$ **Output:** A DAG with best number of partitions for n , \mathcal{G}'

```

1:  $\mathcal{G}_M \leftarrow \text{Moralize}(\mathcal{G})$ 
2:  $C \leftarrow$  All 2-vertex cliques of  $\mathcal{G}_M$ 
3:  $V \leftarrow \{n\}$  ▷
4: for each  $\{a, b\} \in C$  do
5:   if  $a \in V$  and  $b$  is a spatial ref. attribute then
6:      $V \leftarrow V \cup \{b\}$ 
7:   end if
8:   if  $b \in V$  and  $a$  is a spatial ref. attribute then
9:      $V \leftarrow V \cup \{a\}$ 
10:  end if
11: end for
12:  $\mathcal{G}' \leftarrow$  Optimize the number of states of nodes in  $V$  together

```

Algorithm 14 Adaptive_Structure_Learning (Version 1)

Input: Initial dependency graph, \mathcal{G} ; Relational Schema, \mathcal{R} ; Scoring function, $Score$;
Maximum slot chain length, SL_{max} **Output:** Local optimal dependency graph, \mathcal{G}'

```

1:  $\mathcal{G}' \leftarrow \mathcal{G}$ 
2:  $S_{max} \leftarrow Score(\mathcal{G}')$ 
3:  $SL \leftarrow 0$  ▷ Current slot chain length
4: repeat
5:   repeat
6:      $\mathcal{N} \leftarrow \text{Generate\_Neighbors}(\mathcal{G}', \mathcal{R}, SL)$ 
7:      $N^* \leftarrow \arg \max_{N' \in \mathcal{N}} Score(N')$ 
8:      $S^* \leftarrow Score(N^*)$ 
9:     if  $S^* > S_{max}$  then
10:       $\mathcal{G}' \leftarrow N^*$ 
11:       $S_{max} \leftarrow S^*$ 
12:    end if
13:  until No change in  $\mathcal{G}'$ 
14:  for each  $C_{SA} \in \mathcal{G}'$  do
15:     $\mathcal{G}' \leftarrow \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}', C_{SA}, Score)$ 
16:  end for
17:   $SL \leftarrow SL + 1$ 
18: until  $SL > SL_{max}$ 

```

8.4 Evaluation of PRM-SA learning algorithms

We follow the same principle of evaluating PRM learning algorithms, explained in Section 2.7, for evaluating PRM-SA learning algorithms. We start with a gold-standard PRM-SA, and generate a database from it. We apply our proposed PRM-SA structure learning algorithms to identify dependency structures. Then, we learn parameters of the identified dependency structures. Lack of well-known PRMs leads us to start with synthetic PRMs-SA as our gold-standard model. Because of the presence of spatial attributes, we cannot generate random PRMs-SA (and eventually spatial datasets)

Algorithm 15 Adaptative_Structure_Learning (Version 2)

Input: Initial dependency graph, \mathcal{G} ; Relational Schema, \mathcal{R} ; Scoring function, $Score$;
Maximum slot chain length, SL_{max} **Output:** Local optimal dependency graph, \mathcal{G}'

```

1:  $\mathcal{G}' \leftarrow \mathcal{G}$ 
2:  $S_{max} \leftarrow Score(\mathcal{G}')$ 
3:  $SL \leftarrow 0$  ▷ Current slot chain length
4: repeat
5:   repeat
6:      $\mathcal{N} \leftarrow \text{Generate\_Neighbors}(\mathcal{G}', \mathcal{R}, SL)$ 
7:      $N^* \leftarrow \arg \max_{N' \in \mathcal{N}} Score(N')$ 
8:      $S^* \leftarrow Score(N^*)$ 
9:     if  $S^* > S_{max}$  then
10:       $\mathcal{G}' \leftarrow N^*$ 
11:       $S_{max} \leftarrow S^*$ 
12:     end if
13:     for each  $C_{SA} \in \mathcal{G}'$  do
14:        $\mathcal{G}' \leftarrow \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}', C_{SA}, Score)$ 
15:     end for
16:   until No change in  $\mathcal{G}'$ 
17:    $SL \leftarrow SL + 1$ 
18: until  $SL > SL_{max}$ 

```

from the methods proposed by Ben Ishak [2015] (see Section 2.7.2). Thus, we extend their algorithms and generate PRM-SA benchmarks. To compare the learned PRM-SA with the original PRM-SA, we can compute the evaluation metrics explained in Section 2.7.1. In the following, we present the PRM-SA benchmark generation process in detail.

8.4.1 Evaluation strategy and metrics

PRMs-SA can be compared in the same way as PRMs. So, we compare two PRMs-SA by counting the number of relevant dependencies in their dependency structure, and then computing the hard and soft versions of precision, recall and f-score proposed by Ben Ishak [2015] given by Equations 2.4 in Section 2.7.1. As PRMs-SA do not overlap to DAPERS, we cannot measure RSHD to compare PRMs-SA.

8.4.2 Generation of PRM-SA benchmarks

As PRMs-SA are extended from regular PRMs, generating a PRM-SA benchmark is quite similar to generating a PRM benchmark. So, for this task, we follow Ben Ishak [2015]’s three steps of generating an RBN benchmark, explained in Section 2.7.2. Briefly, the first step generates a PRM-SA, the second step generates a relational skeleton, and the third step applies a sampling algorithm to generate a random dataset from PRM-SA for the generated relational skeleton. Significant modifications are needed in all the three steps for some reasons. First, a PRM-SA needs a spatial relational skeleton but their algorithm is not capable of generating a one. Second, relational skeletons generated by their algorithm are primitive and do not resemble the ones from real-world

Algorithm 16 Generate_Neighbors (Adaptative Structure Learning Version 3)

Input: A DAG, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; Scoring Function $Score$
Output: Neighbors of G , \mathcal{N}

```

1:  $\mathcal{N} \leftarrow \{\}$ 
2: for each  $n \in \mathcal{V}$  do
3:   for each  $n' \in \mathcal{V} \setminus n$  do
4:      $\mathcal{G}_{add} \leftarrow \text{Add\_edge}(\mathcal{G}, n, n')$ 
5:      $\mathcal{G}_{del} \leftarrow \text{Delete\_edge}(\mathcal{G}, n, n')$ 
6:      $\mathcal{G}_{rev} \leftarrow \text{Revert\_edge}(\mathcal{G}, n, n')$ 
7:     if  $n$  or  $n'$  is a spatial ref. attribute then
8:       if  $n$  is a spatial ref. attribute then
9:          $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{add}, n, Score)$ 
10:         $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{del}, n, Score)$ 
11:         $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{rev}, n, Score)$ 
12:       end if
13:       if  $n'$  is a spatial ref. attribute then
14:          $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{add}, n', Score)$ 
15:          $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{del}, n', Score)$ 
16:          $\mathcal{N} \leftarrow \mathcal{N} \cup \text{Find\_Structure\_With\_Best\_k}(\mathcal{G}_{rev}, n', Score)$ 
17:       end if
18:       else
19:          $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{G}_{add} \cup \mathcal{G}_{del} \cup \mathcal{G}_{rev}$ 
20:       end if
21:     end for
22:   end for

```

applications because of the fact that they generate almost equal number of objects in each class, which is very unlikely in real world. To overcome this limitation, we have proposed an algorithm for generating skeletons as *k-partite graphs* (Algorithm 6 in Section 2.7.4). Third, generation of a complete GBN is an expensive task and we want to avoid this. For this, we have adapted Kaelin [2011]’s LABG algorithm for sampling a PRM (presented as Algorithm 9 in Section 2.7.4).

The three steps for generating a PRM-SA benchmark are listed in Algorithm 17 and are explained in the following sections.

Generation of random PRMs-SA

In Ben Ishak [2015]’s approach of generating random RBNs, they first generate a random relational schema, then generate random dependencies between attributes of classes, and finally assign random conditional probability distributions to all attributes. We adopt the same steps for generating random PRMs-SA. However, instead of a relational schema, a spatial relational schema is needed for a PRM-SA.

Generating a spatial schema To generate a spatial schema (see Algorithm 18), first of all, a relational schema is generated as a DAG, where each node represents a class and each edge between nodes represents a relational link between classes. A primary key and some standard attributes are then added to each class. Each attribute is assigned some states according to a policy. Now to add spatial information in this relational schema, the required number of spatial classes are selected from the schema

Algorithm 17 Generate_PRM-SA_Benchmark

Input: Number of non-spatial classes, N_c ; Number of spatial classes, N_{sc} ; Maximum length of slot chain length, \mathcal{K}_{max} ; Maximum number of parents, P_{max}

Output: $\Pi : \langle \mathcal{R}, \mathcal{S}, \theta \rangle$; A database instance, \mathcal{I}

Step 1: Generate a random PRM-SA

- i) $\mathcal{R} \leftarrow \text{Generate_Spatial_Schema}(N_c, N_{sc})$
- ii) $\mathcal{S} \leftarrow \text{Generate a set of dependencies between attributes of classes in } \mathcal{R} (P_{max})$
- iii) Determine slot chains on \mathcal{S} with maximum length \mathcal{K}_{max}
- iv) $\theta \leftarrow \text{Generate CPDs for } \mathcal{S}$
- v) $\Pi \leftarrow \langle \mathcal{R}, \mathcal{S}, \theta \rangle$

Step 2: Generate a relational skeleton

- i) $\sigma_r \leftarrow \text{Generate_Spatial_Relational_Skeleton}(\mathcal{R}, N, \text{policy})$

Step 3: Database population

- i) $\mathcal{I} \leftarrow \text{Generate_Spatial_Relational_Dataset}(\Pi, \sigma_r)$

and spatial attributes are added to each of them. In the context of spatial databases, these spatial attributes would be columns of type 'geometry'. For each added spatial attribute, a spatial ref. attribute is added to the class, a spatial partition class are introduced to the schema, and a relational link between the spatial ref. attribute and the spatial partition class is added.

Generating probabilistic dependencies The next step in random PRM-SA generation is to add probabilistic dependencies randomly between two attributes in the same class or in different classes that are accessible via slot chains. For this, in the original approach of Ben Ishak [2015], they first find all potential parents (with different length of slot chains) for each variable and add links between randomly chosen parents. They do not consider the fact that while finding slot chains, duplicate slot chains might be encountered. By 'duplicate', we mean the slot chains which produce the same result. For example, in the schema of Figure 8.1a, $\text{User_satisfaction.user_id}$ and $\text{User_satisfaction.user_id.user_id}^{-1}.user_id$ are equivalent because traversing through the slot chains, we obtain the same set of User objects. Similarly, the slot chain $\text{User_satisfaction.resto_id}^{-1}.resto_id$ is the same as an empty slot chain because this slot chain results in the target Restaurant object. Thus, to improve Ben Ishak [2015]'s approach, we pick the shorter slot chains to avoid redundant, unnecessary computations when such duplicates are found.

Simplifying slot chains We apply the following rule to simplify slot chains. From Definition 3, a slot chain is represented as a sequence of reference slots and inverse slots as $\rho_1.\rho_2.\dots.\rho_{n-1}.\rho_n$. If ρ_{n-1} is an inverse slot and $\rho_n^{-1} = \rho_{n-1}$, then the slot chain can be simplified by eliminating the last two slots. So, the simplified slot chain would, then, be $\rho_1.\rho_2.\dots.\rho_{n-3}.\rho_{n-2}$. This can be done repetitively until no simplification is possible.

The above rule of simplifying slot chains in PRMs also applies to PRMs-SA. However, slot chains in PRMs-SA can be further simplified if the slot chain involves a spatial reference attribute and the dependency has the same spatial reference attribute as a child. In such case, following rule can be applied to simplify such slot chains:

Algorithm 18 Generate_Spatial_Schema**Input:** Number of non-spatial classes, N_c ; Number of spatial classes, N_{sc} **Output:** A spatial relational schema $\mathcal{R} : \langle \mathcal{X}, \mathcal{P}_{SA}, \mathcal{C}_{SA}, \rho \rangle$

```

1: repeat
2:    $\mathcal{G}(\mathcal{X}, \mathcal{E}) \leftarrow \text{Generate\_DAG}(N_c + N_{sc}, \text{policy})$ 
3: until  $\mathcal{G}$  is a connected DAG.
4:
5: for  $X_i \in \mathcal{X}$  do
6:    $\mathcal{P}k\_X_i \leftarrow \text{Generate\_Primary\_Key}(X_i)$ 
7:    $\mathcal{A}(X_i) \leftarrow \text{Generate\_Attributes}(\text{policy})$ 
8:   for  $X_i.A \in \mathcal{A}(X_i)$  do
9:      $\mathcal{V}(X_i.A) \leftarrow \text{Generate\_States}(\text{policy})$ 
10:  end for
11: end for
12:
13:  $\rho \leftarrow \{\}$  ▷ Set of reference slots
14: for  $e(X_i \rightarrow X_j) \in \mathcal{E}$  do
15:    $Fk\_X_i \leftarrow \text{Generate\_Foreign\_Key}(X_i, X_j, \mathcal{P}k\_X_j)$ 
16:    $\rho \leftarrow \rho \cup \{Fk\_X_i\}$ 
17: end for
18:
19:  $\mathcal{C}_{SA} \leftarrow \{\}$  ▷ Set of all spatial ref. attributes
20:  $\mathcal{P}_{SA} \leftarrow \{\}$  ▷ Set of all spatial partition classes
21: for  $X_i \in \mathcal{X} : 1 \leq i \leq N_{sc}$  do
22:    $\mathcal{SA}(X_i) \leftarrow \text{Generate\_Spatial\_Attribute}(\text{policy})$  ▷ Add spatial attributes
23:    $\mathcal{C}_{SA}(X_i) \leftarrow \{\}$  ▷ Set of spatial ref. attributes in class  $X_i$ 
24:
25:   for  $X_i.SA \in \mathcal{SA}(X_i)$  do
26:      $X_i.C_{SA} \leftarrow$  A new attribute (a spatial ref. attribute for  $X_i.SA$ )
27:      $\mathcal{C}_{SA}(X_i) \leftarrow \{\mathcal{C}_{SA}(X_i) \cup X_i.C_{SA}\}$ 
28:      $\mathcal{V}(X_i.C_{SA}) \leftarrow \text{Generate\_States}(\text{policy})$ 
29:      $P_{X_i\_SA} \leftarrow$  A new spatial partition class corresponding to  $X_i.C_{SA}$ 
30:      $\mathcal{P}_{SA} \leftarrow \mathcal{P}_{SA} \cup \{P_{X_i\_SA}\}$ 
31:      $\mathcal{P}k\_P_{X_i\_SA} \leftarrow \text{Generate\_PK}(P_{X_i\_SA})$ 
32:      $Fk\_X_i \leftarrow \text{Generate\_Foreign\_Key}(X_i, P_{X_i\_SA}, \mathcal{P}k\_P_{X_i\_SA})$ 
33:      $\rho \leftarrow \rho \cup \{Fk\_X_i\}$ 
34:   end for
35:    $\mathcal{C}_{SA} \leftarrow \mathcal{C}_{SA} \cup \mathcal{C}_{SA}(X_i)$ 
36: end for

```

A slot chain that involves a spatial ref. attribute can be simplified by eliminating the last two slots if it meets all of the following criteria:

1. The child in the involved dependency is the same spatial ref. attribute,
2. ρ_n is an inverse slot of ρ_{n-1} (i.e., $\rho_n^{-1} = \rho_{n-1}$),
3. ρ_{n-1} is a spatial reference attribute
4. Either ρ_{n-2} is not an inverse slot or $\rho_{n-2} = \rho_n$

Algorithm 19 Generate_Spatial_Relational_Dataset

Input: A PRM-SA, Π ; Total number of objects, N **Output:** Random dataset \mathcal{D}

- 1: $\mathcal{D}_{partial} \leftarrow$ Relational Block Gibbs sampling (Π, \mathcal{I})
 - 2: $\mathcal{D} \leftarrow$ Generate_Spatial_Attribute($\mathcal{D}_{partial}$)
-

Algorithm 20 Generate_Spatial_Relational_Skeleton

Input: A PRM-SA, Π ; Total number of objects in the resulting skeleton, N ; Scalar parameter for CRP, α **Output:** A spatial relational skeleton σ_r

- 1: $\mathcal{G} \leftarrow$ DAG representation of the spatial relational Schema of Π
 - 2: $\mathcal{P}_{SA} \leftarrow$ The set of partition classes in G
 - 3:
 - 4: **for** $P \in \mathcal{P}_{SA}$ **do**
 - 5: $\sigma_r(P) \leftarrow$ Generate objects for P
 - 6: **end for**
 - 7:
 - 8: $\mathcal{G}' \leftarrow \{\mathcal{G} \setminus \mathcal{P}_{SA}\}$ \triangleright Sub-DAG obtained by removing partition classes from \mathcal{G}
 - 9: $\sigma'_r \leftarrow$ Generate_Relational_Skeleton(\mathcal{G}', N, α)
 - 10: $\sigma_r \leftarrow \sigma_r \cup \sigma'_r$
 - 11: Add_Links($\Pi, \sigma_r(\mathcal{P}_{SA}), \sigma_r(\mathcal{SA})$)
-

Generation of a spatial relational skeleton

A spatial relational skeleton can be generated in the same way as its non-spatial counterpart with a special constraint that the skeleton cannot have arbitrary number of partition class objects. For any spatial attribute, the set of objects of associated partition class is the range of the spatial partition function associated with the spatial attribute, and the corresponding spatial ref. attributes refer to this set of partition class objects only. In other words, for any spatial partition class, the number of objects cannot exceed the cardinality of the domain of the referring spatial ref. attributes. For this reason, we first generate partition class objects, and then generate the non-spatial part of the skeleton. These two operations can be interchanged or be done in parallel as they are independent. Finally, we add links between objects of spatial partition classes and those of respective spatial classes. This process is presented in Algorithm 20.

Generation of a spatial relational dataset

To generate a spatial dataset from a spatial relational skeleton, we need to sample two types of attributes: (1) non-spatial attributes, and (2) spatial attributes.

Sampling non-spatial attributes A spatial relational skeleton differs from a non-spatial relational skeleton in that some of the attributes are already observed in the former one. Spatial ref. attributes, which act as both foreign keys and descriptive attributes, are already initialized during the skeleton generation process. One way to generate a dataset from such partially initialized skeleton is to instantiate the PRM(-SA) over the skeleton to obtain a GBN, set evidences to this network and then apply a BN sampling algorithm that supports evidences, such as Rejection sampling, Gibbs sampling etc. However, GBN generation is an expensive task, especially when the

skeleton is big and complex. Relational forward sampling (see Algorithm 8) aims at avoiding GBN generation by sampling nodes of a PRM in a topological order and working directly with databases. It is, however, not applicable for generating spatial relational dataset because it does not support evidences. To avoid a complete GBN generation, and to support evidences in relational skeletons, we propose to adapt Kaelin [2011]’s LABG algorithm for sampling a PRM (see Algorithm 2 for LABG algorithm, and Algorithm 9 for our proposed Relational Block Gibbs (RBG) sampling algorithm). As the adapted algorithm supports partially initialized skeletons, it is applicable for spatial as well as non-spatial relational dataset generation. Thus, we apply RBG algorithm to sample non-spatial attributes in a spatial relational skeleton.

Sampling spatial attributes Algorithm 9 samples non-spatial attributes only. To get a complete spatial dataset, we need to sample spatial attributes too. Sampling a spatial attribute involves two tasks: (1) assigning the centers of partitions (i.e., sampling the spatial attribute of spatial partition classes), and (2) sampling the remaining spatial attributes in the skeleton. Here, we propose two methods for sampling spatial attributes: *unconstrained randomization* and *constrained randomization*. In the former method, we pick random points from the entire world and assign them as the center of partitions. If the boundary of partitions is also needed, we can generate random polygons around the centers. In constrained randomization, the input can be a collection of points, a fixed polygon or a collection of polygons.

Case I: A collection of points In this case, we pick random points from the collection and assign them as centers of partitions. For example, we need to simulate data for some specific cities, we are given a collection of cities as points, and we pick random cities to be the center of partitions.

Case II: A fixed polygon When we need to assign partition centers from a fixed polygon (e.g., a specific country/city), we divide the polygon randomly into the required number of clusters and pick a random point within the polygons as the center of the partitions.

Case III: A collection of polygons In this case, we pick random polygons as the boundary of partitions and then pick a random point within the polygon as the center of that partition.

Once we have chosen center and/or boundary of the partition classes, we can proceed with the generation of spatial attributes of spatial classes by generating random points around the centers and within the boundary (if boundary is available) such that the points follow a bivariate normal distribution with the center of the partition as mean and a random positive definite matrix as variance covariance matrix.

8.5 Experimental study

The primary objective of this experimental study is to evaluate PRM-SA learning algorithms that we have proposed in Section 8.3. In this experiment, we evaluate these algorithms on synthetic data that are generated by applying RBG sampling algorithm

(see Algorithm 9) on well-defined PRMs-SA over randomly generated relational skeletons. The skeletons used in this experiment are generated using our *k-partite graph* generation algorithm (see Algorithm 6). Prior to this experiment, we had performed an empirical study of PRM sampling algorithms to assess the validity and characteristics of RBG sampling algorithm, and *k-partite graph*-based skeleton generation algorithm. We refer to the findings of that study, which are presented in Appendix A, to choose different parameters for this experiment. In the following, we will explain our experimental methodology, and present the results of this experiment.

8.5.1 Methodology

We begin by defining some PRMs-SA, which we keep as our gold standard models. We apply RBG sampling algorithm on these models over randomly generated *k-partite graph*-based relational skeletons to generate our benchmark datasets for the experiment. We perform Chi-square goodness-of-fit test on these generated datasets to check how well these datasets are sampled (cf. methodology of the empirical study of PRM sampling algorithms in Appendix A). Our objective is to use as many well-sampled datasets as possible in the experiment. Our starting burn-in value for the dataset generation process is 100. If the null hypothesis of Chi-square goodness-of-fit test is accepted by all nodes in the model, we accept this dataset. Otherwise, we increase the burn-in value (by 100) and restart the dataset generation process. As the time taken by RBG sampling algorithm increases exponentially with the size of skeletons, as discussed in Section A.1.2, generating a perfect dataset for bigger skeletons is time-consuming. Thus, if a well-sampled dataset could not be generated even after 3 trials, we choose the one for which the null hypothesis of Chi-square goodness-of-fit test is rejected by the least number of nodes. Next, we learn PRMs-SA from those datasets applying our proposed PRM-SA learning algorithms. Then, we compare the learned models with the corresponding gold standard ones. The goal is to assess how well the algorithms could reconstruct the PRMs-SA.

We used 7 manually-defined PRMs-SA, shown in Figure 8.6, as our gold standard models. These models cover four different situations of spatial ref. attributes, where spatial ref. attributes in the model have

1. neither parents nor children (Models A1, and A2 of Figures 8.6a, and 8.6b),
2. parent(s) but not children (Model B1 of Figure 8.6c),
3. no parents but children (Models C1, C2, and C3 of Figure 8.6d, 8.6e, and 8.6f),
and
4. parents as well as children (Model D1 of Figure 8.6g).

We generated 7 datasets having 100 – 5000 objects for each of these models except A2, for which we generated 6 datasets having 100 – 3000 objects. Because A2 contains a slot chain that involves a spatial ref. attribute, the probabilistic structure of this model for a given skeleton can have nodes with many children. For the dataset with 5000 objects, the number of children for some nodes was so high that it caused numerical underflow while computing full conditional distribution of a node (cf. Step 9 of Algorithm 9) because it involves the multiplication of probability values, which are always less than 1. A solution to handle this problem is to use logarithmic values. However, since this solution had not been implemented at the time of this experiment,

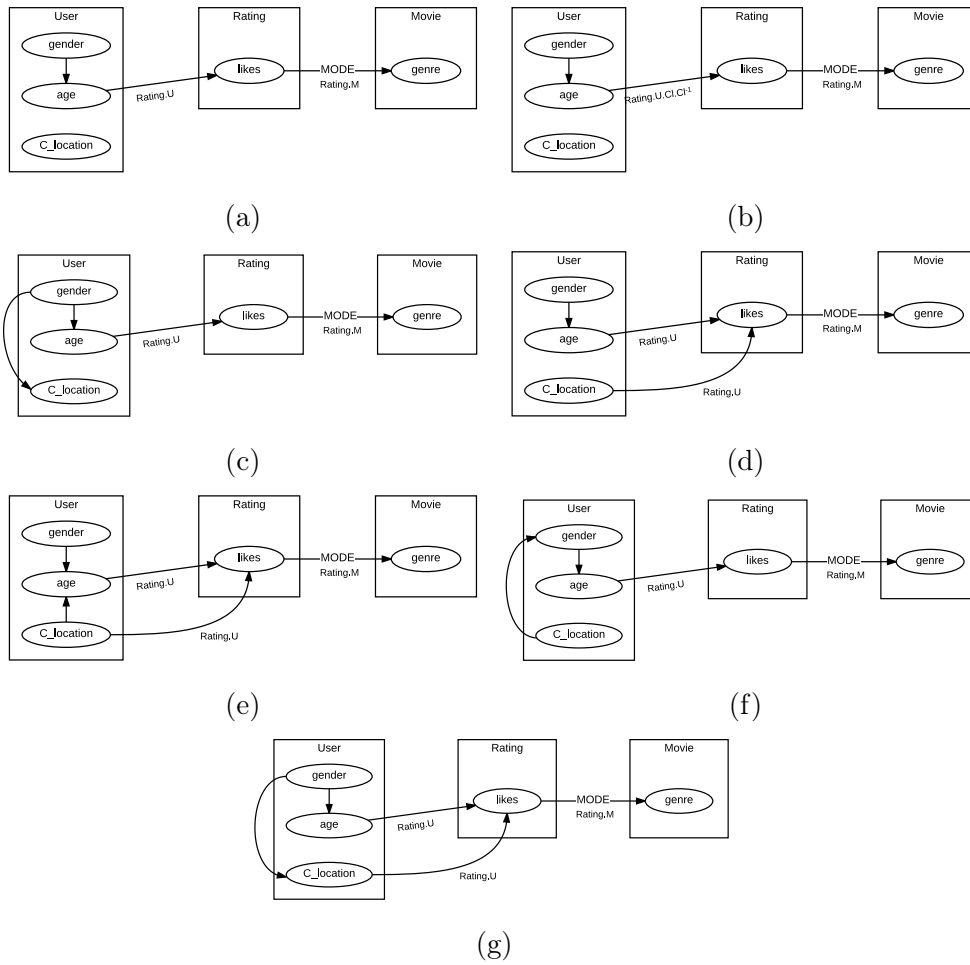


Figure 8.6 – PRMs-SA used in the experiments as gold standard models. We refer to these models by the following name: (a) A1, (b) A2, (c) B1, (d) C1, (e) C2, (f) C3, (g) D1

we continued our experiment with only 6 datasets for A2. Thus, we had altogether 48 datasets for 7 models.

We first applied the standard **RGS** algorithm on each of these datasets to learn a PRM-SA without adjusting the number of partitions during structure learning. For this, before applying **RGS** algorithm, we assigned each spatial object to a spatial partition by using **KMeans**. The number of partitions was chosen in such a way that the initial structure had the best score. The partitions were not modified during **RGS**. We refer to the models obtained in such a way as ‘Baseline’ models, and the corresponding algorithm as ‘Baseline’ algorithm. The objective was to check if adapting partitions while learning a PRM-SA is effective than learning a PRM-SA without adapting partitions. Then, we applied our proposed PRM-SA structure learning algorithms, which adapt partitions during learning, on all datasets. During the learning process, **KMeans** was used as the partition function to create spatial partition class objects. After learning PRMs-SA, we compared the learned models with the corresponding gold standard ones. Reconstruction quality was measured using hard and soft versions of precision, recall, and F-score proposed by **Ben Ishak [2015]**, given by Equations 2.4 in Section 2.7.1. As **PRMs-SA** do not overlap to **DAPERs**, we could not measure **RSHD** to compare **PRMs-SA**.

Soft precision and soft recall of Equations 2.4 consider reversed edges as irrelevant edges. In case of Bayesian networks, even though two **DAGs** have reversed edges,

they may encode the same probabilistic dependencies. Two DAGs are equivalent iff they have the same *skeleton*, and the same *v-structure*. However, this notion is not well-defined for PRMs yet. Not considering such reversed edges may result in lower precision and recall even when the learned PRM is equivalent to the gold PRM. Thus, in this experiment, we assess the quality of learned PRMs by comparing their skeleton with the that of corresponding gold PRMs. For this purpose, we define precision and recall for skeleton comparison in the following way.

Let \mathcal{S}'_{true} , and $\mathcal{S}'_{learned}$ be the skeleton of the gold PRM and the learned PRM respectively, which are obtained by ignoring the direction of all edges. Let Nb_{true} be the number of edges in \mathcal{S}'_{true} , and $Nb_{learned}$ be the number of edges in $\mathcal{S}'_{learned}$. Then, we define precision and recall for skeleton in the following way.

$$\text{Precision}_{skeleton} = \frac{\text{Number of matching edges in } \mathcal{S}'_{true} \text{ and } \mathcal{S}'_{learned}}{Nb_{learned}} \quad (8.3a)$$

$$\text{Recall}_{skeleton} = \frac{\text{Number of matching edges in } \mathcal{S}'_{true} \text{ and } \mathcal{S}'_{learned}}{Nb_{true}} \quad (8.3b)$$

Though these are not perfect metrics for comparing PRMs, they can give hints on the cause of low hard precision or hard recall values. They can be used to understand whether the low value is because of reversed edges or because the algorithm learned completely different edges. When learned models include reversed edges, these metrics for skeletons will be higher than the corresponding hard metrics. If low hard precision/recalls are caused by truly irrelevant edges, the metrics for skeletons must have similar values as hard ones.

The metrics discussed so far can assess the quality of the overall structure of PRMs but cannot identify if spatial attributes have affected the structure. Thus, we propose two types of metrics to understand if spatial attributes have contributed in the structure or not.

The first type of metrics compares the *Markov blanket* of spatial reference attributes in the gold PRM with that in the learned PRM. We propose hard and soft versions of precision and recall for this. Let M_{gold}^i , and $M_{learned}^i$ denote the dependency structure among i^{th} spatial ref. attribute and its Markov blanket in gold and learned PRMs-SA respectively. Then, we define hard precision for the Markov blanket of i^{th} spatial ref. attribute as the ratio of the number of edges present in both M_{gold}^i , and $M_{learned}^i$ (with correct direction, slot chain and aggregator) to the number of edges present in $M_{learned}^i$. The overall hard precision for all spatial attributes in a PRM-SA, denoted $\text{Hard Precision}_{spatial}$ is the average of the hard precision for the Markov blanket of each spatial ref. attribute. We define hard recall for all spatial attributes in a PRM-SA in the same manner as follows.

$$\text{Hard Precision}_{spatial} = \frac{1}{N} \sum_{i=1}^N \frac{\text{Number of edges present in both } M_{gold}^i \text{ and } M_{learned}^i}{\text{Number of edges in } M_{learned}^i} \quad (8.4a)$$

$$\text{Hard Recall}_{spatial} = \frac{1}{N} \sum_{i=1}^N \frac{\text{Number of edges present in both } M_{gold}^i \text{ and } M_{learned}^i}{\text{Number of edges in } M_{gold}^i} \quad (8.4b)$$

where N is the number of spatial attributes in the PRM-SA.

For soft version of these metrics, we consider only the skeleton of M_{gold}^i , and $M_{learned}^i$. Let these skeletons be denoted by $M'_{gold}{}^i$, and $M'_{learned}{}^i$ respectively. Then, soft precision and recall for spatial attributes in a PRM-SA is defined like in Equations 8.4.

$$\text{Soft Precision}_{spatial} = \frac{1}{N} \sum_{i=1}^N \frac{\text{Number of edges present in both } M'_{gold}{}^i \text{ and } M'_{learned}{}^i}{\text{Number of edges in } M'_{learned}{}^i} \quad (8.5a)$$

$$\text{Soft Recall}_{spatial} = \frac{1}{N} \sum_{i=1}^N \frac{\text{Number of edges present in both } M'_{gold}{}^i \text{ and } M'_{learned}{}^i}{\text{Number of edges in } M'_{gold}{}^i} \quad (8.5b)$$

Our second metric for spatial attributes is *Normalized Mutual Information (NMI)*, which compares the partitions learned by PRM-SA learning algorithms with the original partitions. Let C_l , and C_g be the partitions of a spatial attribute in the learned PRM-SA, and those in the gold PRM-SA respectively. Then, *NMI* between C_l and C_g is defined as

$$NMI(C_l, C_g) = \frac{I(C_l, C_g)}{\sqrt{H(C_l)H(C_g)}} \quad (8.6)$$

where $I(C_l, C_g)$ is the mutual information between C_l , and C_g , and $H(C)$ is the entropy associated with partitions C . They are defined in the following way.

$$H(C) = - \sum_{i=1}^k P(i) \log_2 P(i) \quad (8.7a)$$

$$I(C_l, C_g) = \sum_{i=1}^{k_l} \sum_{j=1}^{k_g} P(i, j) \log_2 \frac{P(i, j)}{P(i)P(j)} \quad (8.7b)$$

Here, $P(i)$ is the probability that a randomly picked object is in partition $C_i \in C$, $P(i, j)$ is the probability that a randomly picked object belongs to partition C_i in C_l and to partition C_j in C_g , and k_l and k_g are the number of partitions of the spatial ref. attribute in the learned PRM-SA and the gold PRM-SA respectively. $NMI(C_l, C_g) = 1$ when $C_l = C_g$, i.e. when the partitions match perfectly.

In addition to these metrics, we also score each of the learned models to study how well the models are able to describe the underlying data with respect to the gold models. As there is no general approach for comparing PRMs in terms of their score, we propose to check the absolute difference between scores of two models to see how closely they describe the data. In this experiment, we use Bayesian Dirichlet scoring function to score the learned models as well as the gold models.

8.5.2 Results and discussion

In the following, we will use the terms ‘Adaptative1’, ‘Adaptative2’, and ‘Adaptative3’ (‘Adap1’, ‘Adap2’, and ‘Adap3’ in short) for the versions 1, 2 and 3 of our

proposed adaptative approach for PRM-SA structure learning. Table 8.1 reports overall average precision, recall, and F-score along with standard deviations for each of the five algorithms under study when applied on the gold models of Figure 8.6. These metrics for each model are provided in Tables C.1 – C.9 in Appendix C. The best values are shown in bold face whereas the worst values are underlined.

As we are comparing five different algorithms on multiple datasets for each model, we perform Friedman test (Demšar [2006]) to check if there is any difference in the performance of these algorithms. The null hypothesis of this statistical test is that there is no difference in the performance of the algorithms. If the null hypothesis is rejected, we proceed with Nemenyi test to detect significant differences between algorithms. The results of Nemenyi test for overall precision, and recall are depicted in *critical distance diagrams* (CD-diagrams) of Figure 8.8. Detailed results of this test for the evaluation metrics which rejected Friedman test are presented in Appendix C. In these CD-diagrams, lower ranks are better, and the horizontal lines that connect different algorithms denote that the connected algorithms are not significantly different, whereas in the tables, blue values indicate that the performance of those algorithms in terms of the corresponding metrics are observed to be statistically significant from each other for the given model, and green values indicate that the algorithm significantly outperforms all other algorithms.

Tables C.1, C.2, and C.3 show average hard precision, hard recall and hard F-score respectively. We observed that Adaptative3 algorithm obtained the best precision for 4 out of 7 models. However, it produced the worst average recall for 5 models. At the same time, this algorithm had the best overall average precision but the worst overall average recall. On the contrary, Adaptive2 had the best recall (for 4 models) but the worst precision (for 3 models). Baseline, Naive and Adaptative1 algorithms had extreme results in fewer models. However, Baseline had the worst F-score, and Naive had the best F-score.

Friedman tests followed by Nemenyi tests on these metrics showed that Adaptive3 had significantly better precision than Adaptive2 in model A1. However, we could not come to such conclusion about Baseline, Naive and Adaptative1 algorithms for this model. The result of Nemenyi test on hard precision for model A1 is shown in CD diagram of Figure C.1a. In Figure C.1a, Adap3, which is at the rightmost side, is connected with Naive and Adap1 but not with Adap2, indicating that the performance of Adap2 and Adap3 in terms of hard precision is significantly different from each other. Similarly, it was found that Adaptive3 had significantly better precision than Adaptative1 for model A2, and Naive performed significantly better on model C2 than Baseline in terms of hard precision. For other models, Friedman test was not rejected. Overall, Adaptive2 was observed to have the worst precision but the best recall, and Adaptive3 had the best precision but the worst precision. This difference in performance between these two algorithms was statistically significant too, as seen in Figure 8.7a. However, we could not come to any conclusion regarding Naive and Adaptative1. Regarding overall hard F-score, the statistical tests could not detect any significant difference between the algorithms.

Average soft precision, soft recall and soft F-score of all algorithms observed for each model are presented in Tables C.4, C.5, and C.6 respectively. In all models except A2, these values were found to be the same as the hard version of these metrics because edges in gold and learned PRMs-SA differed by slot chains and/or aggregators

Table 8.1 – Average \pm standard deviation of metrics for four PRM-SA structure learning algorithms in the experiment.

Metrics	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
Hard Precision	0.495 \pm 0.191	0.550 \pm 0.228	0.489 \pm 0.252	<u>0.469</u> \pm 0.204	0.575 \pm 0.220
Hard Recall	0.396 \pm 0.177	0.430 \pm 0.188	0.419 \pm 0.233	0.469 \pm 0.173	<u>0.380</u> \pm 0.181
Hard F-score	<u>0.429</u> \pm 0.167	0.470 \pm 0.184	0.441 \pm 0.231	0.456 \pm 0.160	0.435 \pm 0.168
Soft Precision	0.497 \pm 0.186	0.552 \pm 0.223	0.492 \pm 0.248	<u>0.471</u> \pm 0.200	0.580 \pm 0.212
Soft Recall	0.398 \pm 0.172	0.433 \pm 0.183	0.421 \pm 0.229	0.471 \pm 0.167	<u>0.384</u> \pm 0.175
Soft F-score	0.431 \pm 0.162	0.473 \pm 0.179	0.444 \pm 0.227	0.458 \pm 0.155	<u>0.439</u> \pm 0.161
Precision_{skeleton}	0.827 \pm 0.211	0.830 \pm 0.214	0.809 \pm 0.226	<u>0.706</u> \pm 0.187	0.914 \pm 0.186
Recall_{skeleton}	0.672 \pm 0.247	0.664 \pm 0.242	0.701 \pm 0.251	0.732 \pm 0.226	<u>0.627</u> \pm 0.261
F-score_{skeleton}	0.723 \pm 0.205	0.719 \pm 0.203	0.734 \pm 0.221	<u>0.702</u> \pm 0.171	0.711 \pm 0.211

in few cases only. Even though model A2 had different average hard and soft precisions, recalls, and F-scores, there was no difference in the statistical test results. Thus, conclusions about the hard version of these metrics are valid for the soft version too.

It was observed that many of the learned PRMs-SA had the edge between the nodes *User.age* and *Users.gender* reversed. Because all models except C2 do not have any v-structure involving these nodes, and reversing this edge does not result in a v-structure in all models except C2, and C3, the edges *User.age* \rightarrow *Users.gender* and *User.age* \leftarrow *Users.gender* would be equivalent in GBNs of all models except C2 and C3. We cannot state this for all edges though. Such reversed edges that would have been equivalent to the original edges of gold models are not considered in hard and soft versions of precision, recall and F-score, and hence have affected the metrics adversely. Average hard precision and average recall of the best performing algorithms (i.e., 0.575 and 0.469 respectively) are not very high. To understand if it is the effect of reversed edges, we computed precision, recall and F-score (cf. Equations 8.3) comparing the skeleton of learned models with that of gold models. Average values of these metrics for skeletons are presented in Table 8.1 (Details in Tables C.7, C.8, and C.9). It is clear from this table that soft precision and recall for skeletons are higher than hard precision and hard recall (as well as soft precision and recall) for all algorithms (and for all models as seen in Tables C.1, and C.7 and also Tables C.2, and C.8). This indicates that learned models had reversed edges, which lowered the values of hard metrics. However, whether such reversed edges are equivalent to the corresponding edges in gold models is still an open issue. Again, Adaptative2, and Adaptative3 had respectively the best and the worst recall for skeletons, and also respectively the worst and the best precision for skeletons. Nemenyi test showed that the difference in the performance of these two algorithms is statistically significant (see Figures 8.7e and 8.7f). However, we could not conclude about the differences in performance of these algorithms with Baseline, Naive and Adaptative1.

The metrics discussed so far assess the overall structure of learned models without any specific regards to spatial nodes. Thus, to understand the effect of spatial attributes on the structure of learned PRMs-SA, we computed the metrics given by Equations 8.4, 8.5, and 8.6. These metrics are listed in Table 8.2 (Details in Tables C.10–??).

Hard precision, recall and consequently F-score for spatial attributes of learned

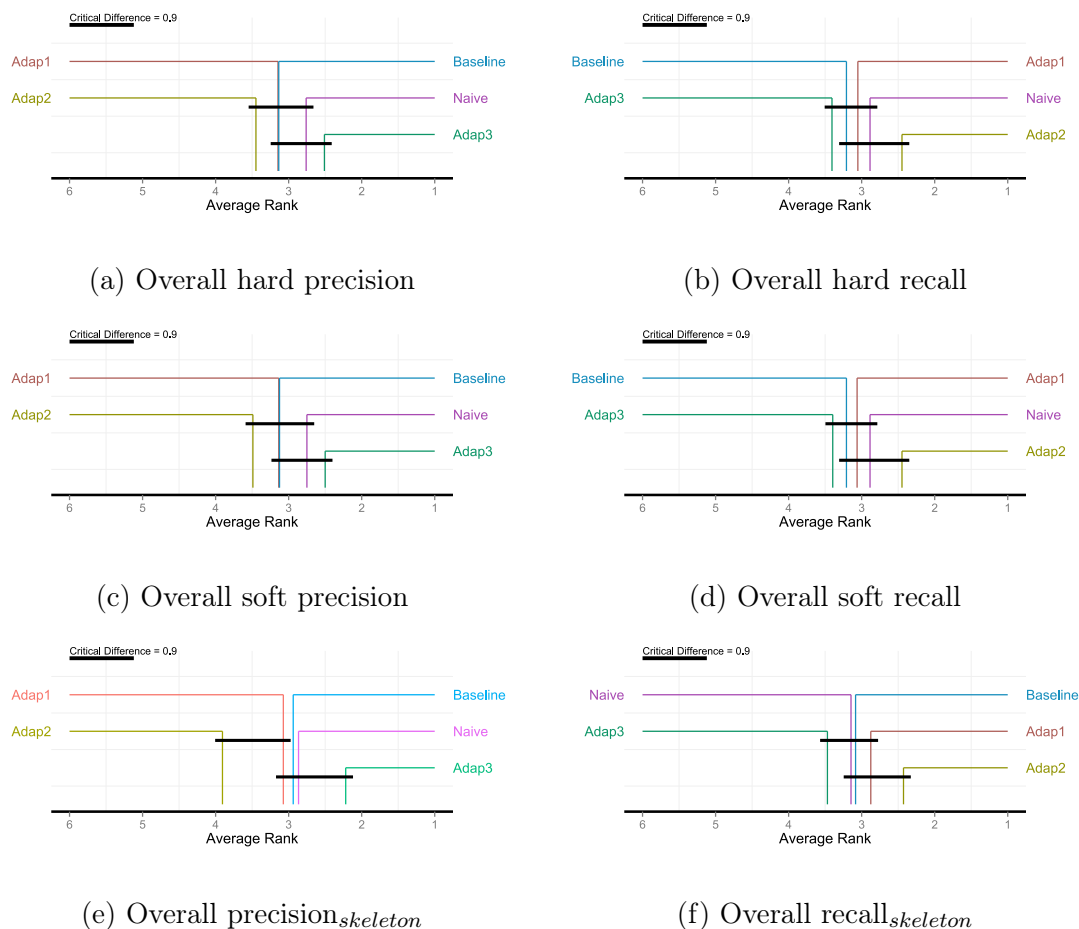


Figure 8.7 – Comparison of overall performance of PRM-SA learning algorithms with Nemenyi test

Table 8.2 – Average \pm standard deviation of metrics to measure spatial influence in the learned models

Metrics	Baseline	Naive	Adaptative 1	Adaptative 2	Adaptative 3
Hard $P_{spatial}$	0.250 \pm 0.438	0.229 \pm 0.425	0.167 \pm 0.377	0 \pm 0	0.688 \pm 0.468
Hard $R_{spatial}$	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449
Hard $F_{spatial}$	0.125 \pm 0.334	0.125 \pm 0.334	0.125 \pm 0.334	0 \pm 0	0.188 \pm 0.394
Soft $P_{spatial}$	0.365 \pm 0.470	0.432 \pm 0.480	0.397 \pm 0.446	<u>0.236</u> \pm 0.312	0.753 \pm 0.424
Soft $R_{spatial}$	<u>0.349</u> \pm 0.442	0.406 \pm 0.442	0.429 \pm 0.444	0.528 \pm 0.416	0.352 \pm 0.454
Soft $F_{spatial}$	<u>0.203</u> \pm 0.361	0.265 \pm 0.377	0.294 \pm 0.405	0.221 \pm 0.271	0.261 \pm 0.410
NMI	<u>0.800</u> \pm 0.061	0.806 \pm 0.062	0.806 \pm 0.062	0.806 \pm 0.062	0.851 \pm 0.128

models were not good for all algorithms but the soft version of these metrics were slightly better indicating the presence of reversed edges around the spatial reference attributes. Again, we observed the same pattern regarding the best and the worst hard as well as soft precision. Adaptative3 significantly outperformed all other algorithms in terms of hard precision_{spatial} (see Figure 8.8a). It also significantly outperformed all algorithms except Naive in terms of soft precision_{spatial} (see Figure 8.8b). Adaptative2 had the worst hard precision_{spatial}, and the worst hard F-score_{spatial}. These results for hard recall_{spatial} and hard F-score_{spatial}, however, were not statistically significant. Re-

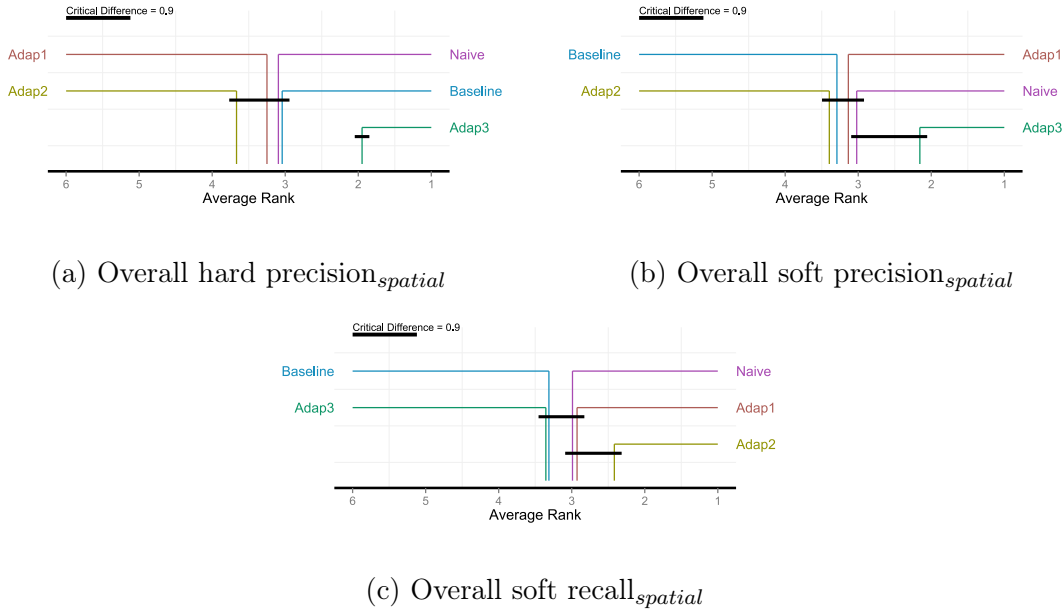


Figure 8.8 – Comparison of overall hard precision_{spatial}, soft precision_{spatial} and soft recall_{spatial} of PRM-SA learning algorithms with Nemenyi test

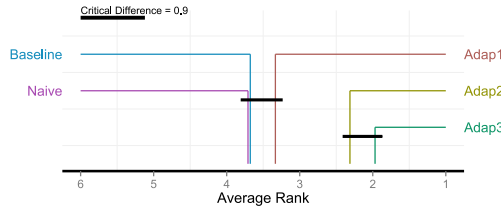


Figure 8.9 – Result of Nemenyi test for comparison of overall performance of PRM-SA learning algorithms in terms of the difference between the score of learned models and that of gold models.

garding soft recall_{spatial}, Adaptive2 performed significantly better than Adaptive3 and Baseline algorithms (see Figure 8.8c). Overall, Adaptive1 had the best soft F-score_{spatial}, and Baseline method had the worst soft F-score_{spatial}. However, statistical significance could not be established for these results. As we can see from Tables C.11 – C.15, these metrics are highly deviated. The result behind this could be the fact that our gold models have only one spatial attribute with small Markov blanket.

Next, we computed NMI between the original partitions of the spatial attribute *User.location* and the ones learned by these algorithms. Overall average of this metric is presented in Table 8.2, and average NMI for each model in Table C.16. From these tables, we can observe that Baseline had the worst NMI in many models, and Adaptive3 had the best NMI. When performed the Friedman test on these results, the null hypothesis of this test was not rejected.

To further study how well the learned models described the underlying data with respect to the gold models, we computed the absolute difference between Bayesian Dirichlet score of each learned model and that of the corresponding gold model. These differences are reported in Table C.17. As model scores are highly dependent on the

data size, the table lists this metric for each model and each dataset. As we can see in the table, the difference between the score of gold models and that of the models learned by Adap3 algorithm is the smallest in many of the cases (31 out of 48 models) whereas the largest differences were observed in many cases (24 out of 48 models) for the models learned by Naive algorithm and those learned by Baseline algorithm. Friedman test followed by Nemenyi test on these observations showed that models learned by Adap3 had significantly better performance compared to Naive, Adap1, and Baseline methods, as depicted in Figure 8.9. That means Adap3 learned models that describe the data as closely as the corresponding gold models do. Models learned by Naive and Baseline algorithm had the worst average rank. However, we could not find significant difference between Naive, Baseline and Adap1 algorithms. It should be noted that a big difference between the score of a learned model and that of a gold model does not necessarily mean that the learned model is worse because we are only measuring the closeness of the learned models with the gold models in the view of model score, and the big difference might be because the algorithm might have learned a model simpler than the corresponding gold model.

To conclude, we observed the following patterns in this experiment:

1. Adaptive3 had the best performance in terms of precision (both spatial as well as non-spatial) not only in simple models (like A1 where the spatial ref. attribute does not have probabilistic dependencies with any other attributes) but also in complex models (like A2, C3, and D1, which include v-structures, a slot chain involving spatial ref. attributes, and more number of dependencies) whereas Adaptive2 had the worst precision even in simple models (like A1, B1 and C1).
2. Recall (non-spatial) of Adaptive2 was significantly better than Adaptive3, which had the worst recall for most of the models.
3. Learned models had reversed edges not only among the spatial reference attribute and its Markov blanket but also in the overall structure.
4. Adapting partitions during PRM-SA learning as proposed in our algorithms (i.e., Naive, and three versions of adaptive algorithms) resulted in more accurate partitions than learning PRM-SA without adapting partitions (i.e., Baseline algorithm).
5. The performance of Baseline algorithm was similar to that of Naive in many cases.
6. Baseline algorithm had low soft precision_{spatial}, and the lowest soft recall_{spatial}, and soft F-score_{spatial}. This indicates that it could not properly identify dependencies of spatial reference attributes.
7. Bayesian Dirichlet score of models learned by Adaptive3 algorithm was significantly closer to that of respective gold models, indicating that those learned models describe the input data quite similarly to the gold models.

8.6 PRMs-SA in recommender systems

Like PRMs, PRMs-SA also have the potential to be deployed in recommender systems for helping users in finding interesting items, and eventually in making decisions. In this section, we will provide a hypothetical recommender system that makes use of

a PRM-SA. This example illustrates the use of PRMs-SA as a spatial recommender system that uses items' various features together with their location, and users' demographic information for making recommendations. It also shows that changing the length of slot chains of dependencies in the PRM-SA can produce different types of recommender systems, demonstrating the use of PRMs-SA as a flexible solution for recommending in the presence of spatial attributes.

Example 8.5 Restaurant recommendation

The PRM-SA of Figure 8.1c can be used for recommending restaurants to users. The idea is to recommend to a target user the restaurants for which the user would give high rating for the service and/or the user's satisfaction level would be high. We can then combine this PRM-SA with a PRM-EU to predict the link between a user and a restaurant. For this, we add a boolean attribute 'Exists' in class `Users_satisfaction`. This attribute will be dependent on the attributes `User_satisfaction.service_rating`, and `User_satisfaction.satisfaction_level` such that `User_satisfaction.Exists` will be true for a given pair of `Restaurant` and `User` objects when both `User_satisfaction.service_rating`, and `User_satisfaction.satisfaction_level` are high. The goal of this recommender system will be then to predict `User_satisfaction.Exists`, and recommend restaurants for which this attribute is true. Partitions for `Restaurant.location` can be defined in different ways as explained in Section 8.2. We can even construct a hierarchical partitioning, like the one proposed by Sarwat et al. [2014] (reviewed in Section 6.2), by further defining partitions for `P_restaurant_location.boundary`.

If there are no aggregators in the edges between the attributes of `Restaurant`, and `Users_satisfaction`, the resulting recommendation model would be simply a content-based filtering system that makes recommendations based on restaurants' features, and their their location. If those edges include slot chains that involve spatial reference attributes, we can achieve more interesting recommendations. For example, if we have the edge `Restaurant.parking_type` \rightarrow `MODE(U.R.C.C-1.service_rating)`, (note: only initial letters of classes and foreign keys are used to compress the slot chain), then it would mean that a user's rating for the service of a restaurant depends on the type of parking of most of the restaurants in the area (partition) where the restaurant is also located. A user might, for example, give better ratings to the restaurants whose most of the neighboring restaurants have underground parking.

8.7 Conclusion

In this chapter, we presented our novel approach to integrating spatial information into PRMs. Our model, which we call PRM-SA, extends standard PRMs, and provides a general solution to model spatial dependencies in PRMs. PRMs-SA consider the fact that spatial heterogeneity leads to some patterns in data. Thus, our basic idea is to extract such patterns through aggregation of spatial objects. Our model also opens a possibility to model spatial autocorrelation with the introduction of aggregated attributes on partition classes and a special constraint on the orientation of edges to avoid cycles. In this chapter, we also presented our approaches for learning the structure of PRM-SA from data, and evaluating these learning algorithms. Our proposed algorithms for learning PRMs-SA are based on standard relational greedy search (RGS) algorithm, which is a score-and-search method (cf. Section 2.2.2). The idea is to find the structure that best fits the data such that the spatial attributes are also well-partitioned. In other words, our proposed learning algorithms not only learns edges

between attributes but also adjust the partitions of spatial attributes while learning. In our naïve approach of **PRM-SA** learning, we apply two new operators *increase_k* and *decrease_k* together with standard *add*, *delete* and *revert edge* operations in **RGS** algorithm (cf. Algorithm 3) to find the neighborhood of a structure, whereas in our adaptative approach, we differentiate between the tasks of greedy search and finding the best number of partitions for spatial reference attributes. We have proposed three versions of adaptative algorithms applying different heuristics that involve different combinations of these two tasks. To evaluate these algorithms, we followed **Ben Ishak [2015]**'s approach of evaluating **PRM** learning approach, which involves the comparison of the model learned from a synthetic data with the model from which the synthetic data is generated. As their algorithm for generating synthetic data cannot generate spatial data, we have also proposed algorithms for generating random **PRMs-SA**, and simulated spatial datasets.

Through an experiment, we demonstrated that adjusting the partitions of spatial attributes during learning could be interesting. We evaluated on several synthetic datasets our four **PRM-SA** learning algorithms along with a static approach, where partitions are not adjusted during the learning process. Our experiments showed that version 3 of our adaptative approach (cf. Algorithm 16) had better precision but worse recall than version 2 (cf. Algorithm 15) but our experimental data was not sufficient to reach any conclusion about the static approach (which we call 'Baseline' algorithm), our naïve approach and version 1 of our adaptative approach. Metrics for skeletons indicated that the hard metrics were affected by reversed edges. We also examined how well these algorithms could learn spatial dependencies. We did not obtain good hard metrics for spatial attributes but soft metrics were better for spatial attributes, indicating that dependencies between spatial reference attributes and other attributes were detected by the algorithms but the direction of those edges were opposite compared to those in gold models. Adap3 was found to have significantly best soft $\text{precision}_{\text{spatial}}$, and soft $\text{recall}_{\text{spatial}}$, whereas Baseline algorithm had the worst performance. Besides, we also compared how similar the learned spatial partitions are to the original spatial partitions, and how close the learned models are to the gold models in the view of model scores. The results showed that Adap3 produced the most accurate partitions and learned the models that are the closest to the corresponding gold models.

From these observations, we could come to a conclusion that Adap3 could be an interesting algorithm for learning **PRMs-SA**. However, there are still open questions regarding the comparison of **PRMs**. Perhaps the most important question would be 'how to detect if two **PRM** are equivalent even though they have different structure?'. Unlike in Bayesian networks, the equivalence of **PRMs** is not well-established yet. The *relational causal discovery (RCD)* algorithm (**Maier et al. [2013]**), which is a relational adaptation of PC algorithm (**Spirites et al. [2000]**) and learns partially-directed causal relational models, could provide a solution towards defining equivalent structures in relational settings. However, their algorithm is specialized for **DAPERs**. Another direction for future research could be to extend **PRM-SA** learning algorithms to use more recent **PRM** learning algorithms, such as **RMMHC**, *Relational Max-Min Parents and Children (RMMPC)* etc. (**Ben Ishak [2015]**). Further, extending **PRMs-SA** to add support for spatial functions that work directly with the spatial attributes could be an interesting future prospect.

Implementations in PILGRIM

Contents

9.1	An Introduction to PILGRIM	136
9.2	Technical aspects	137
9.3	PILGRIM-Relational	137
9.3.1	Modules	139
9.3.2	Implementation of PRM-SA	144
9.3.3	Implementation of PRM benchmark generation	145
9.4	PILGRIM-Applications	148
9.5	Conclusion	150

9.1 An Introduction to PILGRIM

With the goal to provide a tool for working with probabilistic graphical models, *Data, User and Knowledge (DUKE)* research team at *Laboratoire d'Informatique de Nantes Atlantique (LINA)* is actively developing a software platform called PILGRIM (**P**robab**I**listic **G**Raph**I**cal **M**odels). Started in 2010 for modeling, learning and reasoning upon Bayesian networks, this project was extended to support **PRMs** in 2012. An application of **PRMs** on recommender systems was implemented in 2013. Currently, this project has the following sub-projects:

PILGRIM-General

It is the first project to be developed under the PILGRIM project. It is mainly concerned with modeling of standard Bayesian networks and *dynamic Bayesian networks*.

PILGRIM-Structure Learning

This project implements several algorithms to learn structure of standard Bayesian networks and dynamic Bayesian networks. Current version of this project has the implementation of the following structure learning algorithms: Greedy Search, **MMPC**, **MMHC**, $\overline{\text{MMPC}}$, Dynamic Greedy Search, *Dynamic Max-Min Parents and Children (DMMPC)*, *Dynamic Max-Min Hill Climbing (DMMHC)*, $\overline{\text{DMMPC}}$. It also implements several scoring functions such as **AIC**, **BDeu**, **BIC**, and **MDL**.

PILGRIM-Relational

This project aims at modeling **PRMs** and learning probabilistic models from relational data. It primarily implements **PRMs** and its extensions – **PRM-RU**, **PRM-CU**, and **PRM-SA**. This project consists of the following modules:

1. *Core module* provides basic data structures and functionalities for defining **PRMs**,
2. *PRM learning module* deals with the tasks of parameter estimation and structure learning of standard **PRMs**,
3. *PRM benchmark generation* provides algorithms for generating **PRM** benchmark datasets,
4. *PRM extensions module* implements various extensions of **PRMs**, such as **PRMs-SA**, **PRMs-RU**, and **PRMs-CU**.
5. *Utilities module* provides useful functionalities, such as visualization of **PRMs**, importing/exporting relational schema etc.

PILGRIM-Applications

This project was started with the goal to apply **PRMs** in some practical applications. The first application chosen for this purpose is recommender systems. [Huang et al. \[2004\]](#)'s recommender model has been implemented in this project.

PILGRIM is a joint effort of researchers at [LINA](#), and PhD and graduate students at the University of Nantes. This thesis has contributed only in PILGRIM-Relational and PILGRIM-Applications projects. Therefore, this chapter is dedicated to these two projects only. The main contributions of this thesis are in the modules **PRM** extensions, **PRM** benchmark generation and utilities of PILGRIM-Relational project, and in the implementation of recommender systems for PILGRIM-Applications project. [Table 9.1](#) lists the main contributors and their effort on different modules of these projects.

This chapter is organized as follows. We will begin with the technical aspects of PILGRIM in Section 9.2. Sections 9.3 and 9.4 will provide a detailed insight into PILGRIM-Relational and PILGRIM-Applications projects respectively.

9.2 Technical aspects

PILGRIM is developed in C++ and is compatible with Windows and Linux. It utilizes several libraries to provide a complete platform for working with probabilistic models. This section briefly presents such libraries, and relational data source supported by PILGRIM.

Library dependencies The projects PILGRIM-Relational and PILGRIM-Applications depend on the following libraries for various functionalities of the projects:

1. ProBT API¹ for various functionalities related to Bayesian networks, e.g., modeling/inference from a GBN,
2. Boost² for various useful libraries, such as Boost pointer, Boost UBLAS, Boost Graph, Boost Geometry etc.,
3. *Database Template Library (DTL)*³ for communicating with databases
4. Shark-ML⁴ for supervised and unsupervised machine learning algorithms,
5. Google Test⁵ for unit testing,
6. PugiXML⁶ for handling PRMs in XML format
7. Libboard⁷ for visualizing PRMs.

Relational data source Currently, PILGRIM-Relational and PILGRIM-Applications can work with spatial as well as non-spatial data stored in relational databases. To deal with spatial data, an additional extension, called PostGIS, is required.

9.3 PILGRIM-Relational

PILGRIM-Relational offers a platform for working with PRMs. It provides several functionalities for modeling PRMs and its extensions, learning such models from relational databases, and making inferences from such models. A summary of major functionalities implemented in PILGRIM-Relational are presented in Table 9.2. Usage of PILGRIM will be illustrated through some examples later in Appendix B. This section will provide implementation details of PILGRIM-Relational. Section 9.3.1 will present a general overview of PILGRIM-Relational modules. Section 9.3.2 will explain the implementation of PRM-SA. Section 9.3.3 will provide implementation details of PRM benchmark generation.

-
1. <http://www.probayes.com/~mazer/html/index.html>
 2. <http://www.boost.org>
 3. <http://dtemplatelib.sourceforge.net>
 4. <http://image.diku.dk/shark/>
 5. <https://github.com/google/googletest>
 6. <http://pugixml.org>
 7. <https://github.com/c-koi/libboard>

Table 9.1 – PILGRIM modules, and contributions made by the team members

Project manager: Philippe LERAY						
Projects	Members	Modules	Functionalities	Leaders	Participants	
Relational Applications	Philippe LERAY (PhD) Anthony COUTANT (AC) Mouna BEN ISHAK (MBI) Rajani CHULLYADYO (RC) Nasiba HUSEYNOVA (NH)	Core	Architecture	AC	MBI, RC	
			Parameter estimation	MBI	AC	
		PRM Learning	Relational Greedy Search	MBI	AC, RC	
			RMMHC , RMMPG, RMMPG	MBI		
		PRM benchmark generation	Random PRM generation	MBI	RC	
			Random PRM-SA generation	RC		
			Naïve skeleton generation	MBI	RC	
			k -partite skeleton generation	RC		
			Relational Forward Sampling	PhI, NH	RC	
			RBG Sampling	RC		
		PRM extensions	Evaluation metrics	MBI	RC	
			PRM-RU , PRM-CU	AC		
			PRM-SA	RC		
			PRM XML Serialization	AC	MBI	
Utilities	PRM export/import	RC	MBI			
	PRM Visualization	AC				
Applications	Rajani CHULLYADYO (RC)	Recommender System		RC		

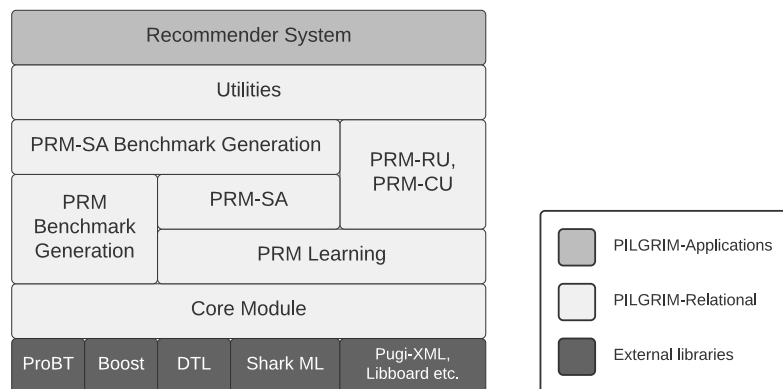


Figure 9.1 – Technological stack diagram

9.3.1 Modules

PILGRIM-Relational library is composed of five modules: core module, [PRM](#) learning, [PRM](#) benchmark generation, [PRM](#) extensions, and utilities. Figure 9.1 shows how these modules are built upon each other.

Core module

The core module provides basic data structures and functionalities for defining PRMs. All other modules are built on top of this module. The major functionalities of this module include [PRM](#) specification, database connectivity to deal with complete or partial instantiations of relational schemas, and instantiation of a [PRM](#) into a [GBN](#) for inference. Several classes are responsible for accomplishing these functionalities. In this section, we will present some of the main classes that constitute this module. In the following, texts in monospace typeface will indicate a C++ class.

PRM specification In PILGRIM, a [PRM](#) is defined through the `RBN` class. This is the most important class in this module. All other modules rely on this class for [PRM](#) specification. An `RBN` object is composed of a `RelationalSchema` object, a dependency structure described by a set of `IRBNVariable` objects, and the associated [CPDs](#) described by `RBNDistribution` objects.

Defining a relational schema A relational schema is specified through the class `RelationalSchema`. It stores a relational schema in the form of a graph such that classes are represented by vertices, and the reference slots (or foreign keys) by edges. Each vertex of this graph is an object of `Class`⁸.

`Class` represents a class of a relational schema. It consists of a set of attributes and an identifier (a primary key). An `Attribute` class represents an attribute of a class (i.e., a column of a table in the context of databases.) Each attribute is identified by a name, and can take a value from a finite set. This set of values is modeled by `Domain` class. It is an abstract class, and has the following implementations: `MultinomialDomain`, `ProBTDomain`, `ContainingNullDomain`, and `CompositeDomain`.

8. Here, `Class` in monospace typeface refers to a C++ class in PILGRIM whereas ‘class’ in plain font refers to a PRM class in general term.

Table 9.2 – Summary of major functionalities implemented in PILGRIM-Relational

Task	Main steps	Specialization	Implementation class
T1. Define a PRM	T1.1 Define a relational schema	PRM	RBN
	T1.2 Define a dependency structure	PRM-SA	RBN-SA
	T1.3 Define parameters	PRM-RU PRM-CU	RBN-RU RBN-CU
T1.1 Define a relational schema		Case I: Well-defined schema	RelationalSchema
		Case II: Import from database	SchemaUtility
		Case III: Generate a random schema	RelationalSchemaGenerator SpatialRelationalSchemaGenerator
T1.2 Define a dependency structure		Case I: Well-defined structure	RBN
		Case II: Learn structure from data	RGS
		Case III: Generate a random structure	RBNGenerateDependencies
T1.3 Define parameters		Case I: Well-defined parameters	RBN
		Case II: Learn parameters from data	RBN
		Case III: Generate random parameters	RBNGenerateParameters
T2. Make inference	Generate a GBN		RBN
T3. Generate a random dataset from a PRM	T3.1 Generate a relational skeleton		
	T3.2 Sample a PRM		
T3.1 Generate a relational skeleton (using one of the two strategies)		Non-spatial relational skeleton	RelationalSkeletonGenerator
		Strategy 1: Naïve skeleton	NaiveSkeletonGenerator
		Strategy 2: k -partite graph-based skeleton	KPartiteGraphGenerator
T3.2 Sample a PRM		Strategy 1: GBN-based sampling	GBNBasedSampling
		Strategy 2: Forward sampling	ForwardSampling
		Strategy 3: Relational Block Gibbs sampling	GibbsSampling

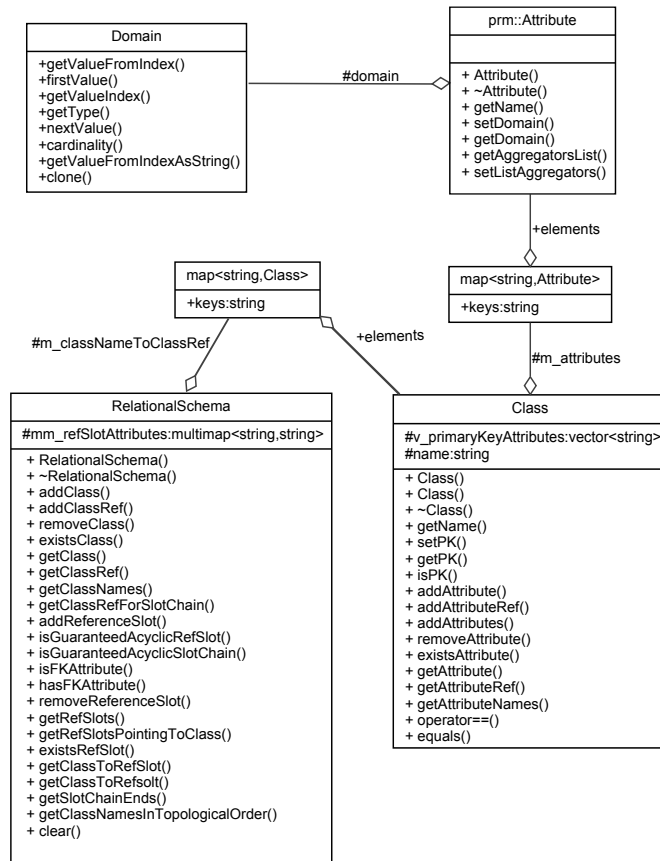


Figure 9.2 – Class diagram showing how RelationalSchema, Class, Attribute, and Domain are related to each other

Figure 9.2 shows how classes RelationalSchema, Class, Attribute, and Domain are related to each other.

Defining a dependency structure and CPDs The dependency structure in a PRM consists of random variables corresponding to class attributes and edges between these variables. PILGRIM-Relational makes a distinction between random variables and class attributes by modeling random variables in a PRM as `IRBNVariable`, and class attributes as `Attribute`. Two types of random variables have been realized in the current version of PILGRIM-Relational: `RBNSimpleVariable`, and `IRBNCompositeVariable`. The former type refers to a simple random variable with a slot chain (of type `SlotChain`) (which can be empty) and with or without an aggregator (of type `Aggregator`) whereas `IRBNCompositeVariable` refers to a random variable that is composed of more than one variable, such as a random variable obtained after performing multi-set operation (see Definition 5). An object of `RBNSimpleVariable` class contains an object of `IRBNCoreVariable`, which indicates the type of random variables, such as a variable created from a class attribute (modeled as `RBNAttributeVariable`), or a selector variable (modeled as `RBNSelectorVariable`) associated with a partition function in a PRM-RU and PRM-CU.

An RBN object consists of a set of `IRBNCoreVariable` objects. Note that an `IRBNVariable` object represents a relational attribute whereas an `IRBNCoreVariable` object represents a node in a PRM without a slot chain and an aggregator. Each node

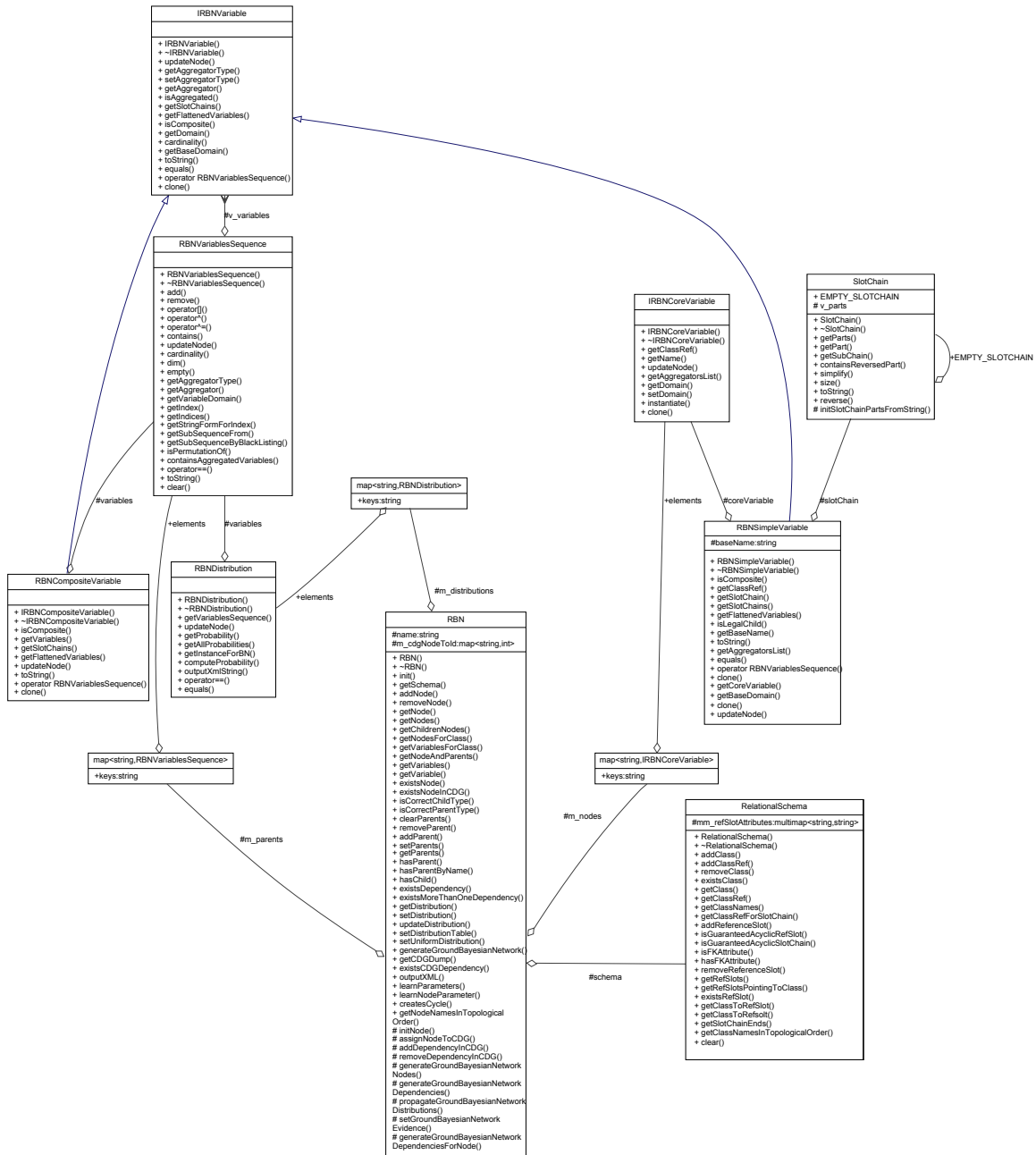


Figure 9.3 – Class diagram showing how RBN is related to other classes

is associated to a set of parents, which is an object of RBNVariablesSequence, and a distribution, represented by RBNDistribution class. The latter class is specialized into RBNDistributionUniform for uniform probability distributions, RBNConditionalDistribution for conditional probability distribution tables, and RBNDistributionProBT for distributions of types supported by ProBT. Figure 9.3 shows how the class RBN is associated with other classes.

Instantiations of a relational schema Instance is an abstract class that defines an interface for a complete or a partial instantiation of a relational schema. Currently, PILGRIM-Relational has two implementations of this class: (1) MockInstance, which is an in-memory storage of a relational schema instantiation in the form of a graph, and is basically implemented for testing purposes, and (2) DBInstance,



Figure 9.4 – Types of RBNDistribution

which is used to communicate with a relational schema instantiation stored as a relational database, and is specialized for PostgreSQL databases only. Connections to a relational database is established through `ConnectionManager` class.

Instantiating a PRM into a GBN A **GBN** is represented by `BayesianNetwork` class. It inherits `plBayesianNetwork` from ProBT library, and adds the concepts of `IRBNCOREVariable` for representing random variables. An RBN object provides the method `generateGroundBayesianNetwork()` to obtain a `BayesianNetwork` object by instantiating the PRM over an `Instance` object.

PRM learning module

This module deals with the tasks of parameter estimation and structure learning of (regular) PRMs. It offers statistical (MLE) as well as Bayesian (MAP, EAP, and Laplace) approaches to parameter learning. Current version of PILGRIM-Relational comes with the implementation of RGS, RMMHC, RMMPC, and $\overline{\text{RMMPC}}$ algorithms for learning the structure of regular PRMs, which are implemented in the classes `RGS`, `AlgoRMMHC`, `AlgoRMMPC` and `AlgoRMMPCBar` respectively (Ben Ishak [2015]). Scoring function for these score-based methods (explained in Section 2.6) is implemented in class `RBNDecomposableScore`.

PRM benchmark generation module

This module provides algorithms for generating random relational schemas, random PRMs, relational skeletons, and sampling PRMs. Initially implemented by Ben Ishak [2015], this module has undergone big modifications to overcome shortcomings mentioned in Section 2.7.3, to extend it with PRMs-SA and spatial dataset generation, and also to improve the quality of the original implementation. As this is one of our major

contributions in PILGRIM-Relational project, we will dedicate Section 9.3.3 for this module.

PRM extensions module

PRM-RU, PRM-CU, and PRM-SA are implemented in PILGRIM-Relational. All of them inherit RBN class, and add data structures and functionalities specific to them. As this thesis contributes to the implementation of PRM-SA, we will explain its implementation in detail in Section 9.3.2. The implementation of PRM-RU and PRM-CU is explained in detail in Coutant [2015]’s dissertation.

Utilities module

Besides the functionalities specific to PRMs, PILGRIM-Relational implements some utility methods to help developers/users in tasks like visualization of PRMs (through PRMDisplay), importing/exporting a relational schema from/to a database (through SchemaUtility), and serializing/deserializing PRMs (in RBNSerializeUnserialize). Some examples will be provided in Appendix B.

9.3.2 Implementation of PRM-SA

The implementation of regular PRM in PILGRIM-Relational does not have support for spatial attributes. Therefore, a new implementation was needed for PRM-SA. As PRMs-SA have many things in common with PRMs, the implementation of PRM-SA has been based on that of PRM.

PRM-SA specification

A PRM-SA is defined in the similar fashion as a PRM. That means, we need to define a relational schema, a dependency structure, and a set of parameters. However, the main things that distinguish PRMs-SA from PRMs are: the presence of spatial classes in the relational schema, the need for adapting a relational schema for spatial attributes, and the presence of partition functions in the PRM-SA. Because of these differences, the implementation of PRM has been extended in the core module.

Defining a spatial relational schema A spatial relational schema is defined through RelationalSchema (of Figure 9.2), which can now contain spatial as well as non-spatial classes thanks to the extension of Class into SpatialClass. SpatialClass holds spatial attributes (SpatialAttribute objects) as well as non-spatial attributes (Attribute). Unlike Attribute objects, SpatialAttribute objects are not associated with a Domain object but a geometry, which can be a point, line or polygon. For a spatial relational schema to be used with a PRM-SA, the schema needs to be adapted. This is done during the initialization of an RBNSA object, an extension of RBN.

Defining a PRM-SA A PRM-SA is implemented in class RBNSA. This class takes a spatial relational schema. During initialization of an RBNSA object, the associated spatial relational schema is adapted. That means, for each spatial attribute in the schema, a spatial partition class is created and is associated with a spatial ref. attribute. A spatial partition class is again a SpatialClass, and spatial ref. attributes are

modeled as an `Attribute` object. Then, partition objects are created according to the specified partition algorithms. Currently, only K-means algorithm has been implemented in PILGRIM-Relational. Once initialized, its dependency structure and parameters can be specified in the same way as for a [PRM](#).

Learning a PRM-SA

[PRM-SA](#) learning algorithms are based on `RGS` class. As explained in Chapter 2, standard [RGS](#) algorithm involves three graph operations: ‘`Add_edge`’, ‘`Delete_edge`’, and ‘`Revert_edge`’. Such operations are implemented as `GraphOperation`, and these three operations are specialized in `AddEdgeOperation`, `DeleteEdgeOperation`, and `RevertEdgeOperation` respectively. `RGS` takes a combination of these graph operations. It maintains a list of graph operation types that can be applied during the learning process. By default, `RGS` apply all of these operations to learn a regular [PRM](#). `RGS` requires a decomposable scoring function, an instance of class `RBNDecomposableScore`, to score candidate [PRMs](#). To avoid computing score of the same structure multiple times, `RGS` uses a `RBNCache`, which caches already computed scores. The class diagram of `RGS` of Figure 9.5 shows how `RGS` is associated with other classes.

Naïve approach For learning a [PRM-SA](#) in naïve way, we add two new types of `GraphOperation` for increasing or decreasing the number of partitions of spatial ref. attributes: `IncreasePartitionOperation`, and `DecreasePartitionOperation`. We then use these two operations together with the three standard operations (‘`Add_edge`’, ‘`Delete_edge`’, and ‘`Revert_edge`’) with `RGS` to learn a [PRM-SA](#).

Adaptative approach The first two versions of our proposed adaptative structure learning algorithms, Algorithms 14 and 15, are implemented in `AdaptativeRGS1` and `AdaptativeRGS2` respectively. These are based on `RGS`, and use the operations `IncreasePartitionOperation`, and `DecreasePartitionOperation` internally to find the best cardinality of spatial ref. attributes during the search process.

As the neighborhood generation process for the third version of our adaptative approach (Algorithm 16) involves the application of the three standard graph operations as well as the optimization of cardinality of spatial ref. attributes, we specialized ‘`Add_edge`’, ‘`Delete_edge`’, and ‘`Revert_edge`’ operations for spatial context in classes `SpatialAddEdgeOperation`, `SpatialDeleteEdgeOperation`, and `SpatialRevertEdgeOperation`. We thus use only these three operations with `RGS` for learning a [PRM-SA](#) using the third version of our adaptative approach.

Figure 9.6 shows the types of `GraphOperation` implemented in PILGRIM-Relational. Examples for naïve as well as adaptative approaches for learning a [PRM-SA](#) are provided in Section B.1.2.

9.3.3 Implementation of PRM benchmark generation

As explained in Sections 2.7.2 and 8.4.2, a [PRM\(-SA\)](#) benchmark is generated in the following sequence: (1) generation of a random (spatial or non-spatial) schema, (2) generation of a dependency structure, (3) generation of a [CPDs](#), (4) generation of a (spatial or non-spatial) relational skeleton, (5) sampling of the [PRM\(-SA\)](#) obtained

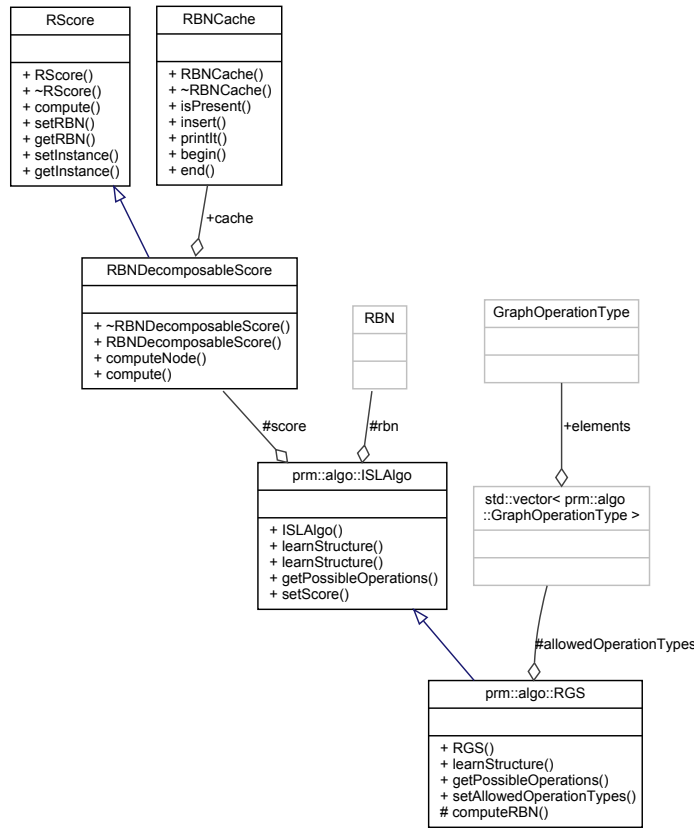


Figure 9.5 – Class diagram of RGS

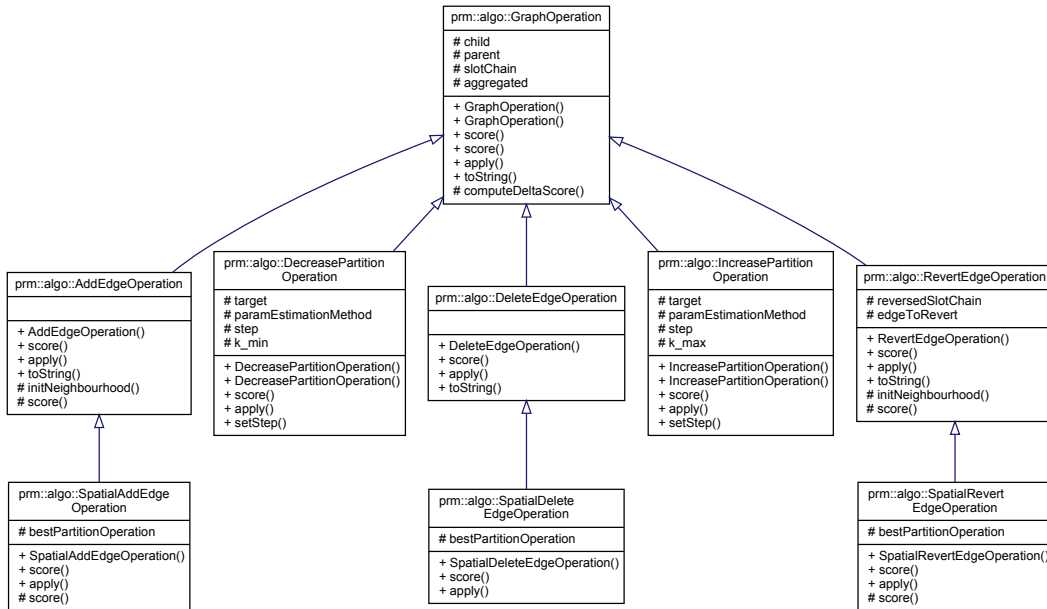


Figure 9.6 – Types of GraphOperation available in PILGRIM-Relational

at step (3) for the skeleton generated at step (4). PILGRIM-Relational provides a clear separation between these steps in the implementation and offers more than one implementation for steps (1), (4), and (5).

Generating a random relational schema RelationalSchemaGenerator is responsible for generating a random non-spatial relational schema. It creates a random

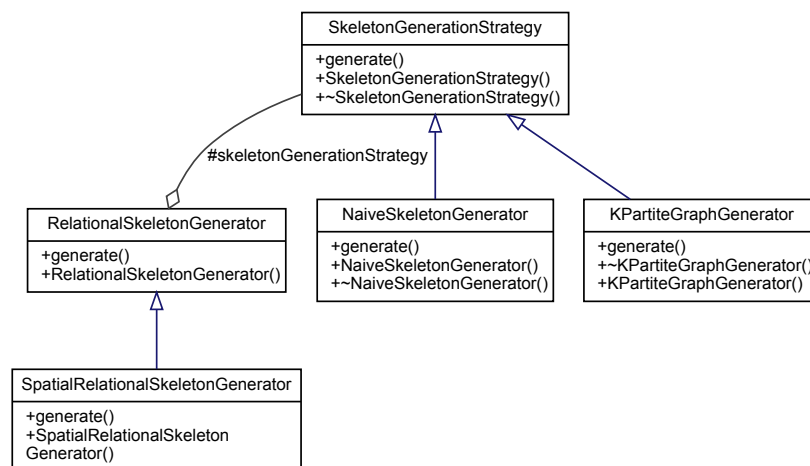


Figure 9.7 – Class diagram of relational skeleton generation strategies implemented in PILGRIM

DAG. Then, for each node in the **DAG**, it adds a class, and for each edge, it adds a foreign key in the class which corresponds to the start of the edge. Currently, two strategies for adding random edges (or generating reference slots) in a **DAG** are available: `RefSlotsGenerationFromPolytree`, and `RefSlotsGenerationWithPMMixed` (Ben Ishak [2015]). `SpatialRelationalSchemaGenerator` generates a random spatial relational schema, and is based on `RelationalSchemaGenerator`. The strategies for generating references slots are still valid for this class.

Generating a random dependency structure Given a relational schema, the class `RBNGenerateDependencies` creates a random **DAG** among the descriptive attributes present in the relational schema. It then assigns a slot chain to each of the edge in the **DAG** randomly. This results in a dependency structure of a random PRM.

Generating random CPDs Once the dependency structure is defined, we need to specify **CPDs** of each node in the dependency structure to get a complete PRM. This is done by `RBNGenerateParameters` class.

Generating a relational skeleton To obtain a benchmark dataset, we first need to define a (non-spatial or spatial) relational skeleton over which a PRM(-SA) can be instantiated to generate a complete dataset. `RelationalSkeletonGenerator` and `SpatialRelationalSkeletonGenerator` are available for generating a non-spatial and spatial skeleton respectively. These require an algorithm for generating a relation skeleton. We have implemented two algorithms for generating non-spatial relational skeletons: `NaiveSkeletonGenerator`, and `KPartiteSkeletonGenerator`. The former implements Ben Ishak [2015]’s algorithm, and the latter implements Algorithm 6 proposed in this thesis. Both inherit `SkeletonGenerationStrategy` class (Figure 9.7), which provides a common interface for these implementations so as to be able to replace one algorithm by another easily. Both `RelationalSkeletonGenerator` and `SpatialRelationalSkeletonGenerator` take one of these strategies but the difference between them is that the latter deals with the adapted version of a spatial relational schema and results in a spatial skeleton.

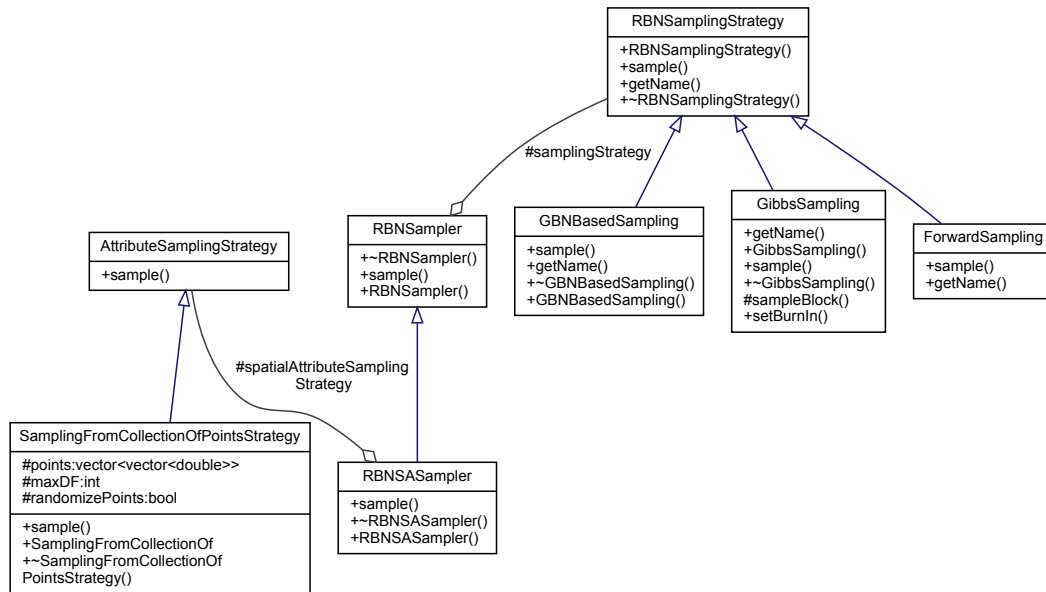


Figure 9.8 – Class diagram of sampling strategies implemented in PILGRIM

Sampling a PRM Next, we need to sample a PRM over a skeleton to obtain a complete dataset. We have provided an abstract class `RBNSamplingStrategy`, which defines an interface for a sampling algorithm. We have extended this class to implement three different algorithms for sampling a PRM, as shown in Figure 9.8. `GBNBasedSampling` implements Ben Ishak [2015]’s algorithm, where a `GBN` is generated by unrolling a PRM over a relational skeleton and forward sampling from ProBT library is applied on this `GBN`. As an improvement to this approach, `ForwardSampling` implements relational forward sampling of Algorithm 8, which works directly with databases avoiding `GBN` generation. To improve the sampling process further by supporting evidences (partially initialized databases or incomplete datasets), we have created `GibbsSampling`, which implements RBG of Algorithm 9. All three of these sampling algorithms are applicable for sampling a regular `PRM` to obtain a non-spatial dataset. However, only `GibbsSampling` can sample a `PRM-SA` because a spatial relational skeleton is partially initialized (cf. hypotheses in Section 8.4.2). To generate a spatial dataset, we also need to sample spatial attributes. `AttributeSamplingStrategy` is an interface for this. Among the three cases for sampling spatial attributes presented in Section 8.4.2, only the first case is implemented in `SamplingFromCollectionOfPointsStrategy`.

9.4 PILGRIM-Applications

This project aims at developing applications using PRMs. The first application realized in this direction is a recommender system. As our first attempt, we implemented Huang et al. [2004]’s recommendation model. Our personalized recommendation model proposed in Chapter 7 was developed independently and is planned to be merged into this sub-project. A summary of major functionalities implemented in PILGRIM-Applications are listed in Table 9.3.

In this project, a recommender system is built around a class called `Recommender`. This class consists of a recommendation model (`RecoModel`) and methods to learn the model, perform predictions on the model etc. Currently, Huang et al. [2004]’s

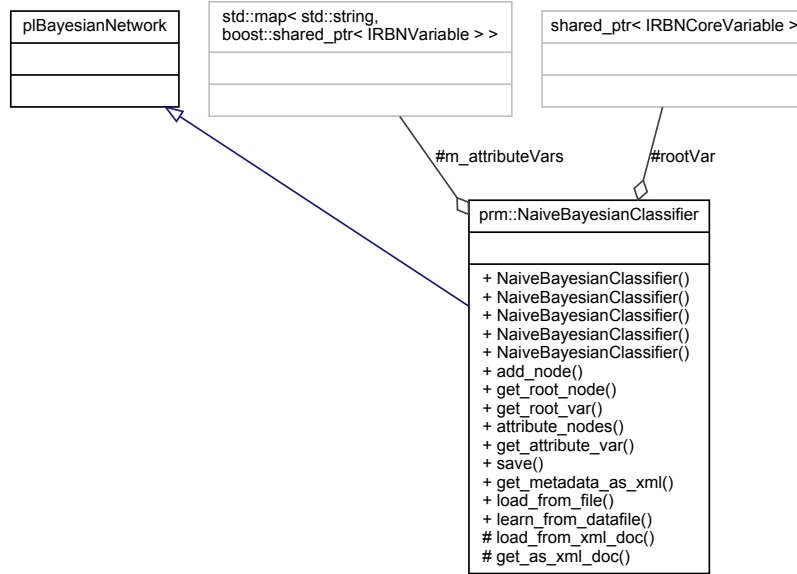
Table 9.3 – Summary of major functionalities implemented in PILGRIM-Recommender

Task	Main steps	Specialization	Implementation class
T1. Define a recommender system (Huang et al. [2004]’s model)		Case I: Well-defined structure	HuangsRecoModel, NaiveBayesianClassifier
	T1.1 Find Markov blanket	Case II: Learn from data	
	T1.2 Build naïve Bayesian classifier		
	T1.3 Learn parameters of the classifier		
T1.1 Find Markov blanket			MarkovBlanket
T1.2 Build naïve Bayesian classifier			HuangsRecoModel
T1.3 Learn parameters of the classifier			HuangsRecoModel
T2. Make top-N recommendations			HuangsRecoModel



Figure 9.9 – Class diagram of RecoModel

recommendation model has been implemented in the class `HuangRecoModel`, which extends `RecoModel` as shown in Figure 9.9. As mentioned in 2.4, this model builds a naïve Bayesian classifier among the target attribute (`Transaction.Exists` in Huang et al. [2004]’s proposal) and its Markov blanket such that the target attribute is the root (target) node of the classifier. Naïve Bayesian classifier is implemented in class `NaiveBayesianClassifier` by extending `plBayesianNetwork` as shown in Figure 9.10.

Figure 9.10 – Class diagram of `NaiveBayesianClassifier`

9.5 Conclusion

In this chapter, we introduced PILGRIM, a software for working with probabilistic graphical models. We presented a brief overview of different sub-projects of PILGRIM. We focused our discussion on two projects where this thesis has contributed the most: PILGRIM-Relational and PILGRIM-Applications. The former is a library that offers different functionalities to work with PRMs (including PRMs-SA). We have presented different modules in this library, and also covered the contributions made by this thesis in the implementation of PRM benchmark generation and PRMs-SA in detail. PILGRIM-Applications is aimed at using the PILGRIM-Relational library to develop useful applications. We have also presented our implementation of [Huang et al. \[2004\]](#)'s model for recommendation in this chapter. Major functionalities provided in these projects have been summarized in [Table 9.2](#). [Appendix B](#) will provide some examples on how to use PILGRIM.

Conclusion

In this thesis, we explored the potential for using probabilistic relational models ([PRMs](#)) in recommendation systems, and with spatial data. [PRMs](#), recommender systems, and spatial data are in fact quite related fields because [PRMs](#) model probabilistic models from relational data, and recommender systems and spatial data are common sources of relational data. However, their intersection is not much explored in machine learning community.

The first contribution of this thesis deals with the overlapping of [PRM](#) and recommender systems. We have proposed a [PRM](#)-based personalized recommender system to show that [PRMs](#) can be used in recommender systems.

Our proposed approach is capable of making personalized recommendations in a system

- where users do not have many interactions because items are less frequently purchased (probably due to high cost of the items),
- where users have preferences not only about items but also for items' characteristics,
- which lacks users' profiles and is in cold-start situation.

The basic idea behind our approach is to determine whether a relation can exist between a user's search session (instead of a user) and an item. Our approach involves two components:

1. A generic [PRM](#) (named [PRM-PrefReco](#)) that can determine whether a relation between a user's search session and an item could exist or not by evaluating how similar the search criteria and items' features are, and
2. A personalized Bayesian network obtained by instantiating the [PRM-PrefReco](#) with users' preferences.

The former is constructed offline using data available in the system and/or experts' knowledge, and the latter is built online, and generates personalized recommendations. With this approach, the same [PRM-PrefReco](#) can be instantiated into a content-based, collaborative filtering, or hybrid recommendation models only by changing the length of slot chains in the [PRM](#). This allows a system, which was once in cold-start situation,

to switch from a basic recommender to a collaborative filtering or hybrid recommender after the system has collected enough data. We performed a preliminary experiment on a real-estate search system which was in the cold-start situation. From the experiment, we could conclude that there is a possibility of using our approach for making personalized recommendations in cold-start situation. However, we could not test our solution in non cold-start situation due to the lack of bigger datasets in our experimental system.

Our second contribution addressed the missing theoretical work for the overlapping of [PRMs](#) and spatial data. We developed a general way to integrate spatial information into a [PRM](#). We proposed probabilistic relational models with spatial attributes ([PRMs-SA](#)), which extends regular [PRMs](#) to support spatial objects. This thesis is particularly concerned with geographical data, where a spatial object is associated with its geometry described by a sequence of pairs of coordinates (i.e., latitude and longitude). Because spatial data are easily obtainable in vector format, which can be easily modeled as an object compatible with [PRM](#) specification, we adopt the vector representation of spatial data. Because of the geometry (such as point, line or polygon) of spatial objects, [PRMs](#) cannot treat spatial objects in the same way as non spatial objects. [PRMs-SA](#) consider the fact that spatial heterogeneity leads to some patterns in data. Thus, our basic idea is to extract such patterns through aggregation of spatial objects. Our specification of [PRMs-SA](#) requires the adaptation of the underlying schema by adding a spatial partition class for each spatial attribute present in the schema. These spatial partition classes are, in fact, aggregation of spatial objects, and are associated with spatial partition functions, which can be standard spatial aggregators, domain knowledge, or spatial clustering algorithms. [PRMs-SA](#) then model spatial dependencies between attributes using the aggregation information.

We also proposed some algorithms to learn [PRMs-SA](#). The idea is to find the structure that best fits the data such that the spatial attributes are also well-partitioned. In other words, our [PRM-SA](#) structure learning algorithms not only learn the dependencies between attributes but also adjust partitions during learning so as to extract dependencies resulted by well-partitioned spatial attributes. Our naïve approach of learning a [PRM-SA](#) involves stepwise increment or decrement of the number of spatial partitions. Our adaptative approach attempts at finding the best number of partitions of spatial attributes at once, instead of performing stepwise increment or decrement, in particular steps during the learning process. We proposed three versions of adaptative approach by changing the steps at which they compute the best number of spatial partitions. In the first version, we find the optimal number of spatial partitions after a complete greedy search for the given slot chain length whereas in the second version, we find the best spatial partitions within each iteration of greedy search algorithm for the given slot chain. In the third version, it is done during the neighborhood generation process after performing add, delete and revert operations if these operations involve spatial reference attributes. We performed experimental evaluations of these algorithms along with a static approach (which we refer to as ‘Baseline’), where partitions are not adjusted during learning, on synthetic datasets. This experiment showed that the third version of our adaptative approach had significantly better precision but worse recall compared to the second version. However, we could not come to any conclusion about other algorithms. By comparing the standard hard and soft versions of evaluation metrics with the metrics for skeletons of the learned [PRMs-SA](#), we could reach the conclusion that the metrics are affected by reversed edges because the standard metrics do not identify equivalent [PRMs](#). Besides, we also observed that the third

version of our adaptative algorithm also resulted in more accurate partitions compared to other algorithms, and produced models that are closest to gold models in terms of the model score.

Besides these two major contributions, we also discussed on the limitations of [Ben Ishak \[2015\]](#)'s approach of evaluating [PRMs](#) learning algorithms. Since their approach had been the basis of evaluating [PRMs-SA](#) learning algorithms, we needed to overcome those limitations, which mainly include the unrealistic nature of skeletons generated by their relational skeleton generating algorithm, and sampling of [PRMs](#) with the presence of evidences. We proposed an algorithm for generating relational skeletons (referred to as [k-partite graph](#)-based skeletons) that resemble real-world skeletons. For sampling a [PRM](#) over a relational skeleton that includes evidences, we proposed relational block Gibbs ([RBG](#)) sampling algorithm, which is based on [Kaelin \[2011\]](#)'s lazy aggregation block Gibbs ([LABG](#)) algorithm for approximate inference. Our experimental study on this sampling algorithm along with relational forward sampling, and [Ben Ishak \[2015\]](#)'s [GBN](#)-based sampling algorithms showed that [RBG](#) produced better samples than other algorithms though it is very time-consuming when it comes to generating big datasets for complex relational skeletons.

Future work

During the research presented in this thesis, many potential directions for future work have opened up. In the following, we list some of them.

Improvement of our proposed recommendation model More [SRL](#) methods are available now. Exploring [SRL](#) models other than [PRMs](#) for recommender systems can be interesting. [Kouki et al. \[2015\]](#), and [Fakhraei et al. \[2015\]](#), for example, have used hinge-loss Markov random fields to build a recommendation system. Adding explanations to the recommendation made ([Bilgic and Mooney \[2005\]](#)) can be an interesting enhancement too. Causal reasoning on Bayesian networks can aid in justifying the recommendations.

Applying our proposed recommender system in diverse domains Though we have evaluated our recommender system in the domain of real estates, it is a generic solution and can be applied in other domains too. Some of the applicable domains are flight/hotel/restaurant recommendation, or other product recommendation scenarios.

Evaluation of all types of recommendation models Our recommender model was evaluated with a small dataset on a very new system, which was in cold-start situation. Because the number of transactions was very low, we could not evaluate all types of models proposed in Chapter 7. Thus, a future prospect is to evaluate all types of proposed models by applying our methodology on bigger datasets.

Improvement of [PRM-SA](#) structure learning algorithms with newer algorithms for [PRM](#) structure learning Our proposed algorithms for learning [PRM-SA](#) structure are based on the standard relational greedy search ([RGS](#)) algorithm for learning [PRMs](#). [Ben Ishak \[2015\]](#) has shown through experiments that [RGS](#) algorithm is less accurate compared to newer algorithms, such as [RMMHC](#), [RMMPC](#) etc. Using

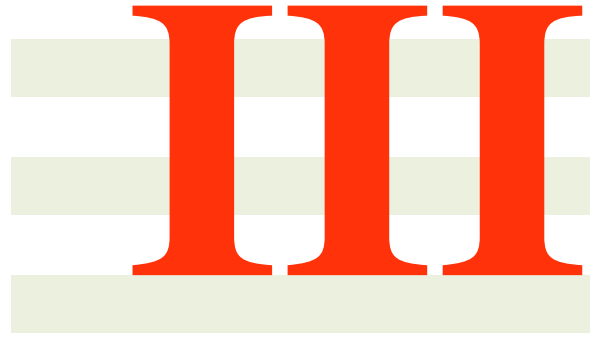
PRM-SA learning algorithms with these newer algorithms might improve the quality of the learned PRMs-SA.

Integration of spatial information and recommender systems Though we have discussed a method to use spatial information with PRMs through PRMs-SA, we haven't evaluated our model on real recommender datasets that deal with users and items. Using a PRM-SA, we can achieve a simple recommender system that is capable of predicting some attributes of interest (e.g., users' rating). We can also obtain a recommender system based on Huang et al. [2004]'s recommendation model. Such recommendation model, however, may be applicable only on systems where enough data exist, and may not address the situation for which we proposed a personalized recommender system (cf. Chapter 7). Thus, one direction for future research could be to merge PRM-SA with our personalized recommender system.

Equivalent PRMs Multiple equivalent Bayesian networks can be learned from the same dataset. In the same way, our PRM(-SA) learning algorithms can also result in equivalent PRMs(-SA). However, determining whether two PRMs are equivalent to each other is still an open issue. Though we have proposed to compare skeletons of PRMs in our experiments, skeleton comparison is not enough for detecting equivalence of PRMs. Consequently, the evaluation metrics of PRM(-SA) learning algorithms were affected. Maier et al. [2013] have proposed *relational causal discovery (RCD)* algorithm, which is a relational adaptation of PC algorithm (Spirtes et al. [2000]). Since PC algorithm can recover a BN structure equivalent to the true structure underlying the input data, Maier et al. [2013]'s RCD algorithm and also its improved version proposed by Lee and Honavar [2016] could provide a solution towards defining equivalent structures in the relational settings. However, these algorithms only learn DAPERs. Therefore, defining the notion of equivalent PRMs is an important direction for further research.

More experiments An area where we have proposed our theoretical approach but not yet performed empirical validation is the existence of multiple spatial attributes in PRMs-SA. Rigorous experiments with more complex (and random) PRMs-SA on bigger datasets are also needed to better benchmark PRM-SA structure learning algorithms.

Enhancement of PILGRIM Official release of PILGRIM software is already in the pipeline. One area where PILGRIM certainly needs an enhancement is the inference process because currently, reasoning using PILGRIM is expensive due to the use of GBN. Implementing Kaelin [2011]'s LABG algorithm for approximate inference could be a stepping stone towards efficient inference. As lifted inference is gaining popularity in recent years, we can also orient towards the implementation of lifted inference in PILGRIM.



Appendices



Empirical Study of PRM Sampling Algorithms

We carried out some experiments to study three algorithms for sampling PRMs – relational block Gibbs (RBG) sampling, relational forward sampling, and GBN-based sampling. The objectives of these experiments were to study RBG sampling, and to compare it with the two other sampling algorithms. We work with two types of relational skeletons in the experiments – one is generated using Ben Ishak [2015]’s algorithm, and another using our *k-partite graph* generation algorithm (see Algorithm 6). We refer to relational skeletons generated by the former algorithm as ‘*Naïve*’ skeletons because they have nearly equal number of objects of each class, and are less complex than the skeletons generated by our algorithm.

We divide the following section into two parts. Section A.1 presents the study of RBG sampling, and Section A.2 presents the comparison of the three algorithms.

A.1 Empirical study of relational block Gibbs sampling algorithm

The aim of this study is to understand how relational block Gibbs sampling performs on datasets of different type and size, how burn-in value affects the performance of the algorithm.

A.1.1 Methodology

We start with a PRM shown in Figure A.1a. The corresponding relational schema is shown as a DAG in Figure A.1b. We have chosen this PRM as our *gold standard* PRM because it has probabilistic dependencies with different length of slot chains and also includes aggregators. It has 6 dependencies – one with slot chain length = 0, four with slot chain length = 1, and one with slot chain length = 3. Two of them involve a MODE aggregator. Conforming to the relational schema (see Figure A.1b) of this PRM, we generate naïve and *k-partite graph* graph-based skeletons having approximately 100,

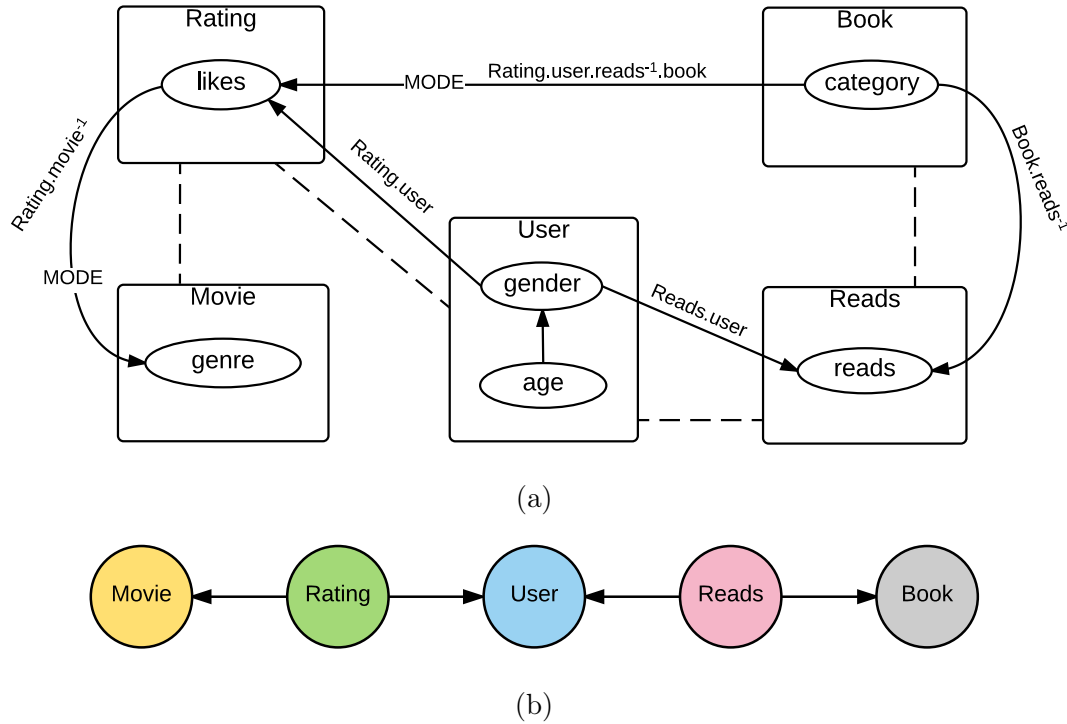


Figure A.1 – (a) The PRM used in the experiments, and (b) the underlying relational schema as a DAG

200, 500, 1000, 2000, 3000, and 5000 objects. So, we have altogether 14 relational skeletons. While generating k -partite skeletons, the choice of the scalar parameter α affects the structure of the skeleton to a great extent. Smaller values of α result in compact skeletons, i.e. many objects will have high in-degree. In-degree of objects, in fact, is determined by whether a referring object gets linked to an existing object or a new object, which in turn depends on the total number of objects generated so far (cf. Section 2.7.4). Thus, instead of picking a constant α for skeletons of different size, it should be chosen based on the size of the skeleton. In this study, we choose α to be the square root of the required number of objects in the skeleton. Also note that while generating these k -partite skeletons, we choose not to generate true scale-free graphs to avoid getting very complex skeletons for the experiments. Next, the PRM is sampled for each of these skeletons applying RBG sampling algorithm with the following burn-in values – 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 2500, and 3000. For each combination of dataset size, burn-in and skeleton type, the time taken to complete the algorithm is recorded, and parameters of the PRM are learned on the generated sample. Chi-square goodness-of-fit test with significance level of 0.05 is performed to compare the original parameters with the learned ones. Using this test, we can check how well the PRM nodes are sampled. Null hypothesis of this test is that the generated data for the given node are consistent with the original distributions used for generating the sample. The nodes that reject the test cannot be considered well-sampled. We count such nodes too.

Characteristics of the datasets

As mentioned previously, there are 14 relational skeletons – 7 naïve skeletons, and 7 k -partite skeletons. Note that in terms of the size of the dataset, we consider the

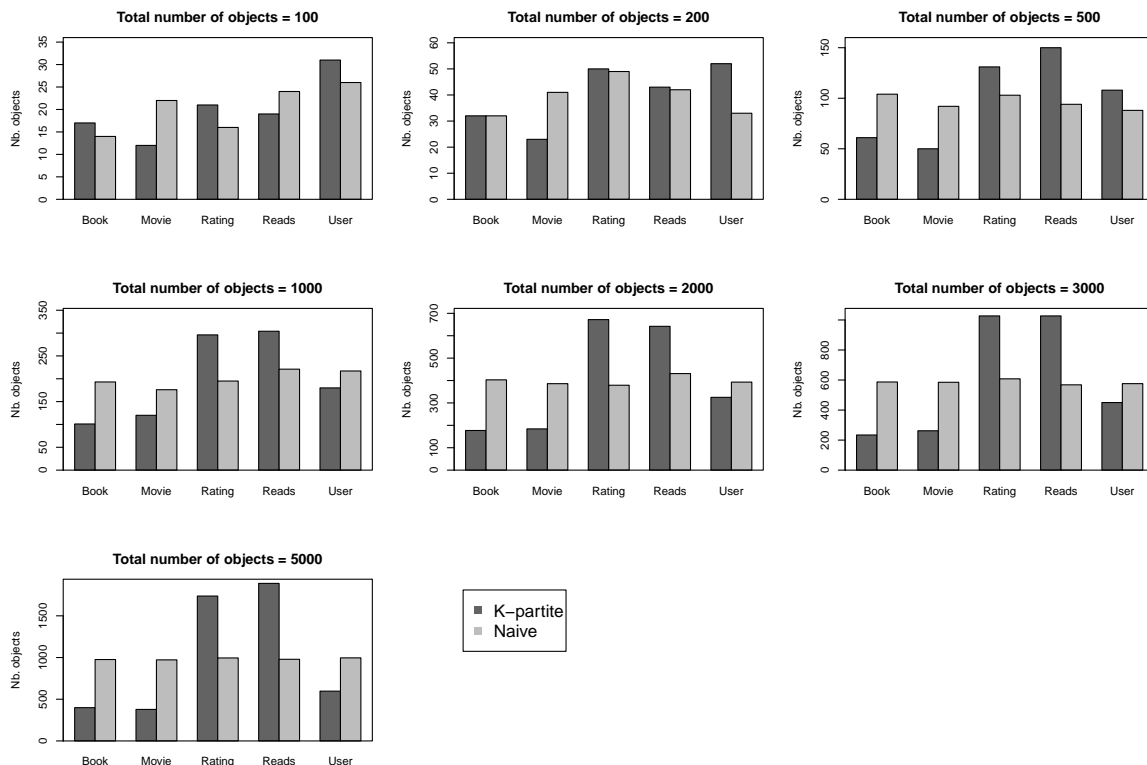


Figure A.2 – Distribution of objects in the relational skeletons used in the experiments

naïve skeleton with approximately 100 objects is comparable to the k -partite skeleton with approximately the same number of objects even though we cannot ensure that they have exactly the same number of objects.

In naïve skeletons, objects are almost uniformly distributed across classes as seen from Figure A.2. This is not true for k -partite skeletons. There are more *User* objects than *Book* or *Movie* objects in k -partite skeletons because *User* objects are referenced by *Rating* as well as *Reads* objects, and, hence, are likely to be generated more often than *Book* and *Movie* objects. The number of objects of relationship classes (i.e., *Rating*, and *Reads*) is higher than that of entity classes (i.e., *Book*, *Movie*, and *User*). However, this expected phenomenon is not observed in the datasets with 100 and 200 objects because the chosen scalar parameter might not have been enough to produce very compact datasets.

Figures A.3 and A.4a reveal that naïve skeletons are less complex than k -partite graph-based skeletons can be observed. The four charts in Figure A.3 correspond to the four edges present in the relational schema DAG shown in Figure A.1b. The charts show that the maximum in-degree of entity objects in naïve skeletons is almost always less than k -partite skeletons, and does not increase significantly with the size of data. Figure A.4a¹ shows the frequency of in-degree of *Book* and *Movie* objects for the references from *Rating* objects in the naïve and k -partite skeletons of different sizes. In this figure, we can see that there are more objects with high in-degree in k -partite skeletons compared to the corresponding naïve skeletons. From these figures, we can conclude that the experimental k -partite skeletons are complex than the naïve skeletons.

1. Distributions of in-degree for only the edges $\text{User} \leftarrow \text{Rating}$ and $\text{Movie} \leftarrow \text{Rating}$ are shown here because those for $\text{User} \leftarrow \text{Reads}$, and $\text{User} \leftarrow \text{Reads}$ are almost similar to the ones presented here.

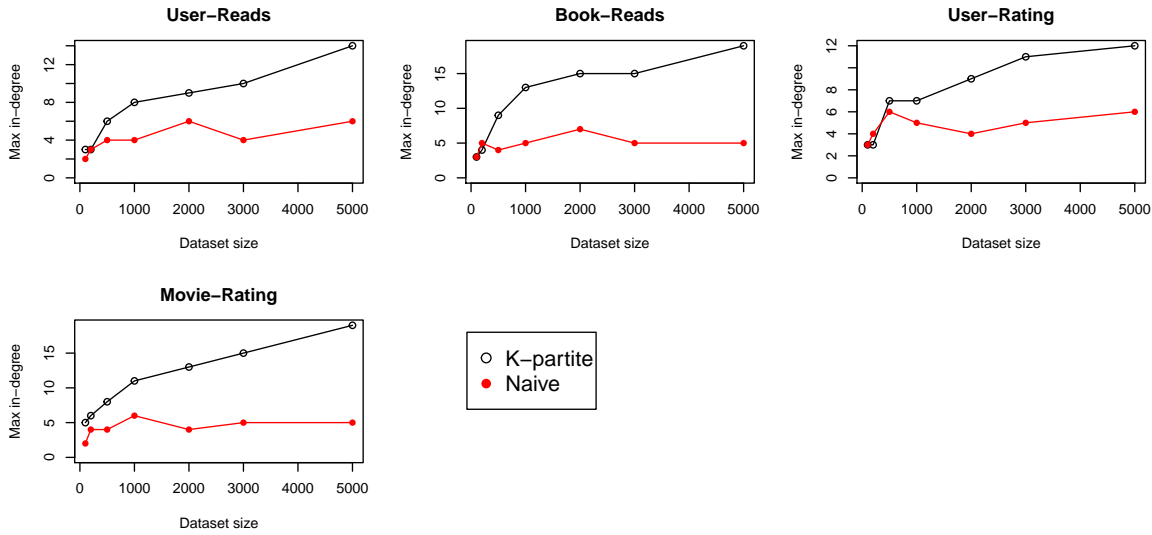


Figure A.3 – Max in-degree of entity objects in the relational skeletons. Each chart corresponds to one of the four edges in the relational schema DAG of Figure A.1b

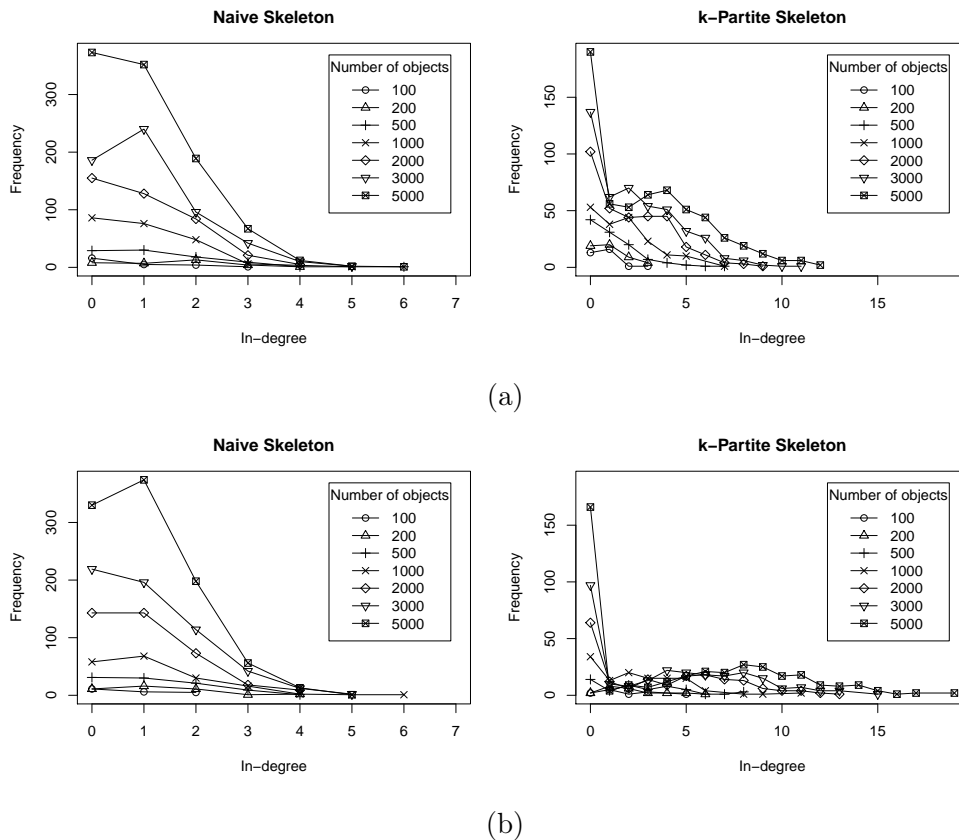


Figure A.4 – Distribution of in-degree in naïve and k -partite skeletons for the edges (a) to User from Rating objects, and (b) to Movie from Rating objects

A.1.2 Results and discussion

Figures A.5 and A.6 show the time taken by the sampling algorithm on each skeleton for different values of burn-in. From the first figure, we can see that it took longer to sample on k -partite skeletons than on naïve skeletons in most of the cases. Possible reason behind this is that naïve skeletons are generally simpler than k -partite skeletons

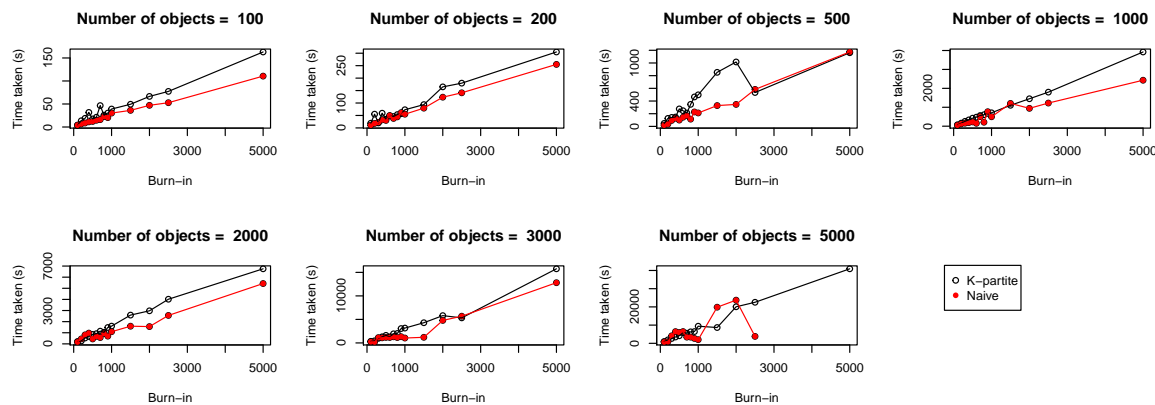


Figure A.5 – Burn-in vs time taken by RBG sampling algorithm on naïve and k -partite graph-based skeletons of different size.

because the latter ones tend to have nodes/objects that are referenced by many other nodes. Therefore, Markov blankets in k -partite skeletons tend to be much bigger than those in naïve skeletons, thereby increasing computation time for conditional probability distributions. Another observation we can make from the same figure is that the time taken by the algorithm increases almost linearly with the burn-in value on most of the skeletons. We can only expect to observe such linear relationship but cannot guarantee it because time taken for sampling actually depends on which attribute is selected, and how big its Markov blanket is. Because selecting an attribute for sampling and generating attribute values are done randomly, we cannot predict the exact behavior of the increase of burn-in value.

From Figure A.6, we can say that on k -partite skeletons, the time taken by the algorithm follows an increasing non-linear function of the number of objects whereas we cannot get such conclusion for naïve skeletons. This is because the shape of naïve skeletons is unpredictable whereas that of k -partite skeletons does not tend to change much with their size. On big k -partite skeletons as well as on small ones, only few nodes will have many incoming edges and many nodes will have few incoming edges. So, big k -partite skeletons are always more complex than small ones because of the presence of nodes with very high in-degree, but we cannot guarantee to observe such phenomenon on naïve skeletons.

Figures A.7, and A.8 present the number of nodes that rejected the null hypothesis of Chi-squared goodness-of-fit test for different burn-in values on skeletons of different size. No nodes must reject the null hypothesis in a well-sampled dataset. So, lower values are better in these charts. As seen in the figures, out of six nodes in the PRM, at most two nodes rejected the null hypothesis on both types of skeletons. As we can see in Figure A.8, k -partite skeleton having 100 objects resulted in good samples even for low as well as high burn-in values. Only 2 out of 14 samples for this skeleton had at most 1 node that was not well-sampled. As the size of k -partite skeletons was increased, more nodes (at most 2) rejected the null hypothesis. Similar pattern can be observed on naïve skeletons (Figure A.8). Maximum number of nodes that rejected the null hypothesis is 1 for smaller naïve skeletons and it is 2 for bigger ones. From these results and also from Figure A.7, we are not decisive about the size of skeletons and the value of burn-in to get perfect samples. We can only say that increasing burn-in can improve the quality of samples with the cost of time. We should note that even small values of burn-in could generate good samples for big datasets (e.g.,

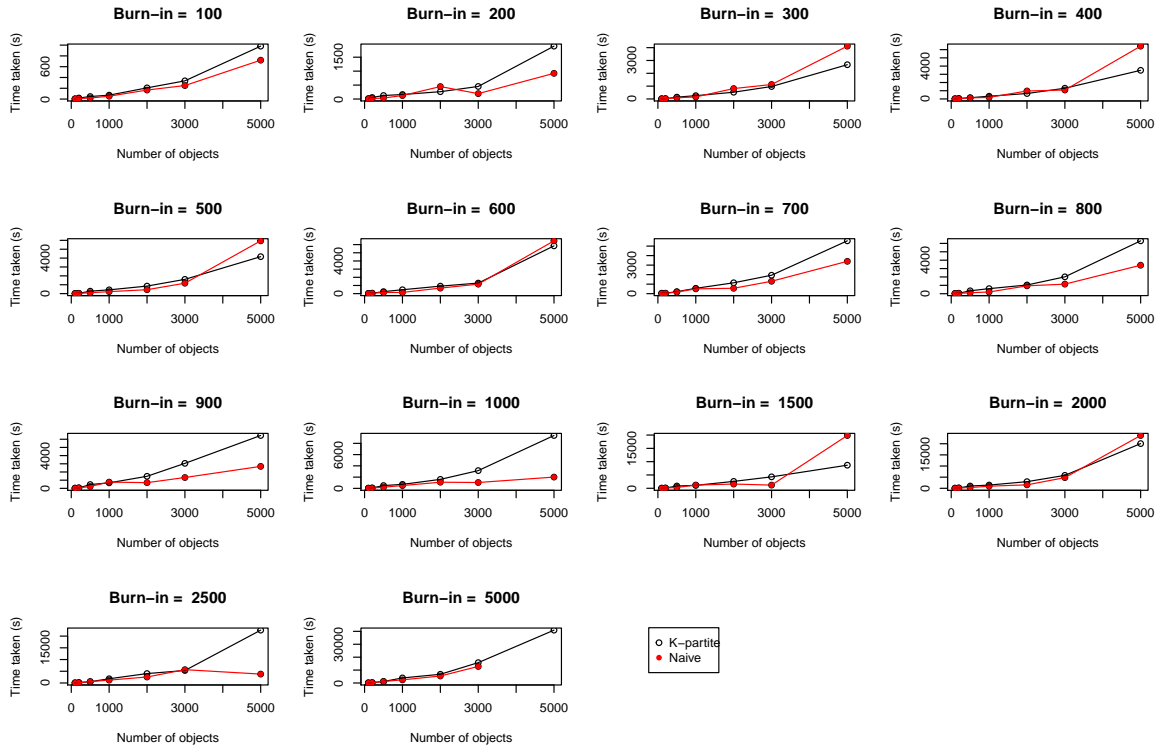


Figure A.6 – Skeleton size vs time taken by **RBG** sampling algorithm for different values of burn-in.

k -partite skeletons having 5000 objects with burn-in=300, 3000 objects with burn-in= 200, naïve skeletons having 5000 objects with burn-in=100 etc.). Thus, it would be better to generate big datasets using a small value of burn-in repeatedly until a well-sampled dataset is obtained instead of using high value of burn-in and trying to generate a good sample in one run.

A.2 Comparison of sampling algorithms

In this experiment, we assess the following three algorithms – relational forward sampling, **RBG** sampling, and **GBN**-based sampling. The primary objective is to verify that **RBG** sampling can replace **GBN**-based sampling. We also aim at comparing **RBG** sampling with relational forward sampling in terms of performance of the algorithms and quality of the samples generated by them.

A.2.1 Methodology

We work with the same PRM of Figure A.1a for this experiment. We first generate naïve and k -partite graph-based skeletons having 100, 200, 500, 1000, 2000, and 3000 objects. The three sampling algorithms are applied over these skeletons to sample the PRM. From the previous experiments (Section A.1), it was observed that high burn-in values are not necessary for obtaining good samples from big skeletons. That is why we use a medium value (600) of burn-in for **RBG** sampling in this experiment.

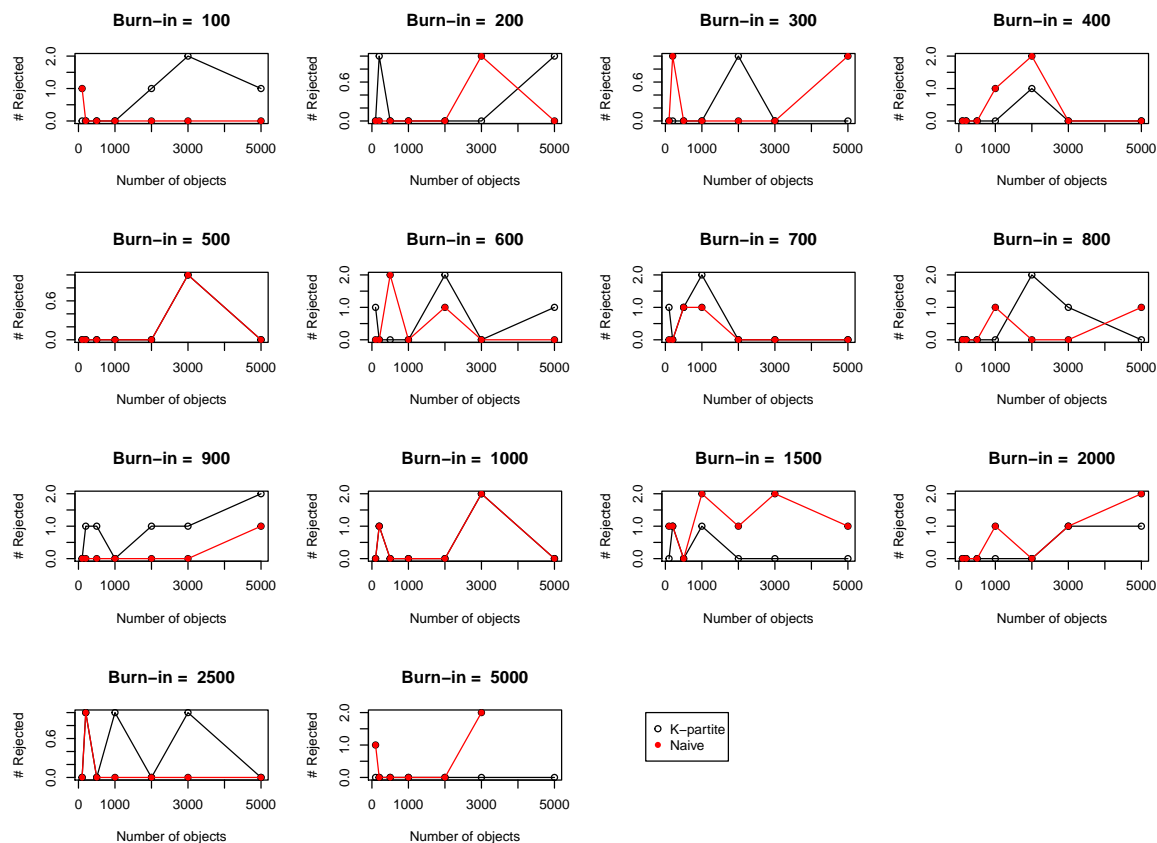


Figure A.7 – Skeleton size vs number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test for different values of burn-in. Lower values are better here.

A.2.2 Results and discussion

Figure A.9 shows that when the time taken to complete the sampling algorithms is considered, relational forward sampling outperforms the two other sampling algorithms. Even on big datasets, it took very less time compared to the two others. Time efficiency of this algorithm lies behind its non-iterative nature and the fact that GBN generation is not required for it. Unlike RBG sampling, it samples each attribute only once. The only time-consuming task in relational forward sampling is the communication with databases. Therefore, we can conclude that relational forward sampling is certainly a good solution if we need to generate very big datasets. One important thing to note here that it is difficult to determine the most time-efficient algorithm among RBG and GBN-based sampling because they are sensitive to the complexity of relational skeletons,. As seen in Figure A.9, RBG sampling took longer on k -partite skeletons but not on naïve ones. Moreover, RBG sampling also depends on the value of burn-in as well as on the time of execution. If we had chosen a smaller burn-in value, we might have obtained very different results. Because attributes are selected randomly for sampling at each step, no two executions of RBG sampling for the same burn-in value would give the same result.

We can observe in Figure A.10 that the number of the nodes rejecting the null hypothesis of the Chi-squared goodness-of-fit test is always lower for RBG sampling on both types of skeletons. All six nodes in the PRM were sampled well on k -partite skeletons by RBG sampling except in one case where only one node rejected the null hypothesis. Also on naïve skeletons, the best result was obtained with RBG sampling.

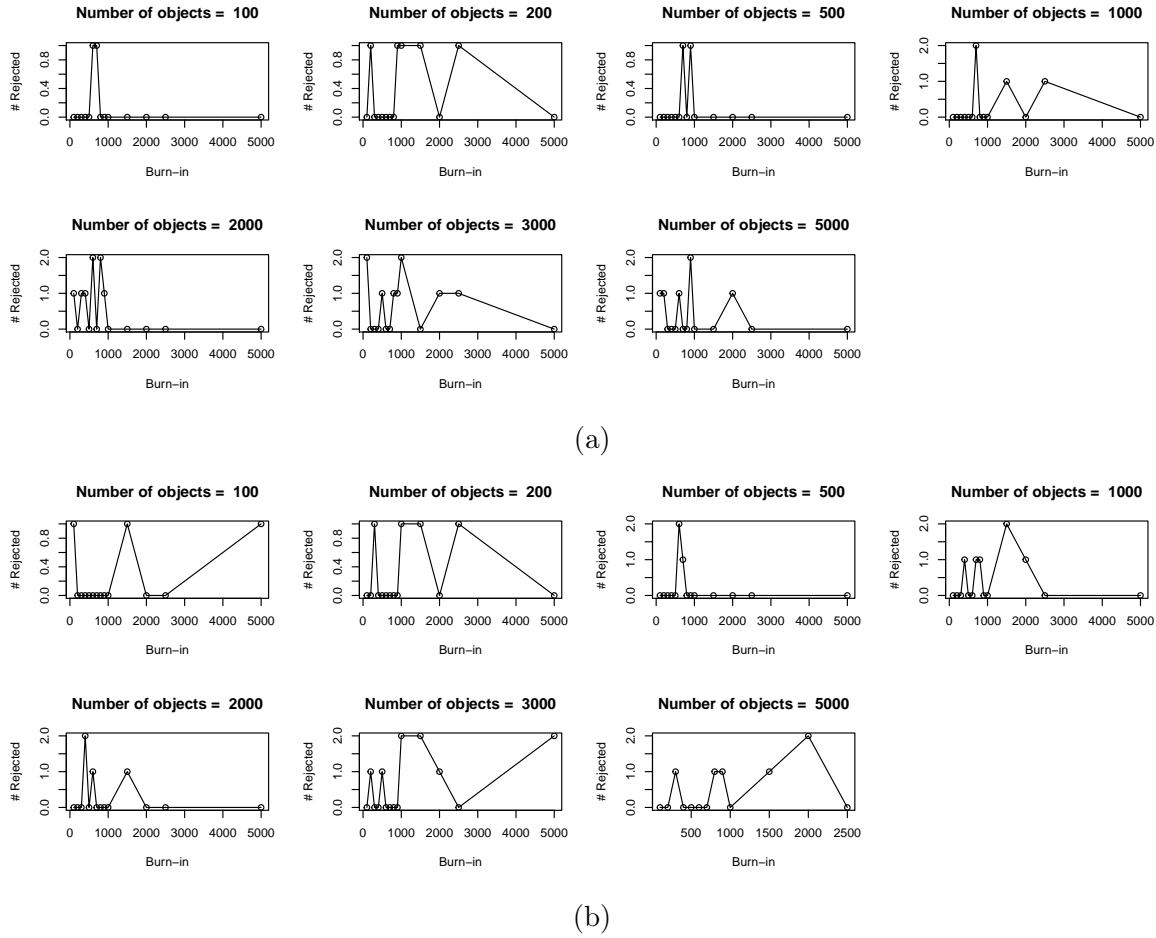


Figure A.8 – Burn-in vs number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test on (a) k -partite graph-based skeletons, and (b) naive skeletons of different size. Lower values are better here.

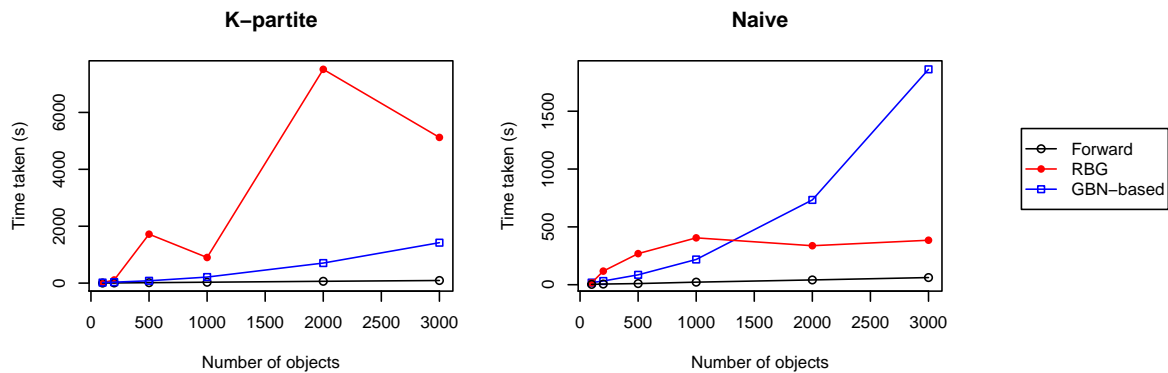


Figure A.9 – Time taken by relational forward sampling, RBG sampling, and GBN-based sampling algorithms on naive and k -partite graph-based skeletons of different size.

Though relational forward sampling was very efficient in terms of time, at least one node was not well sampled with this algorithm. From this, we can say that nodes are generally sampled well with RBG sampling.

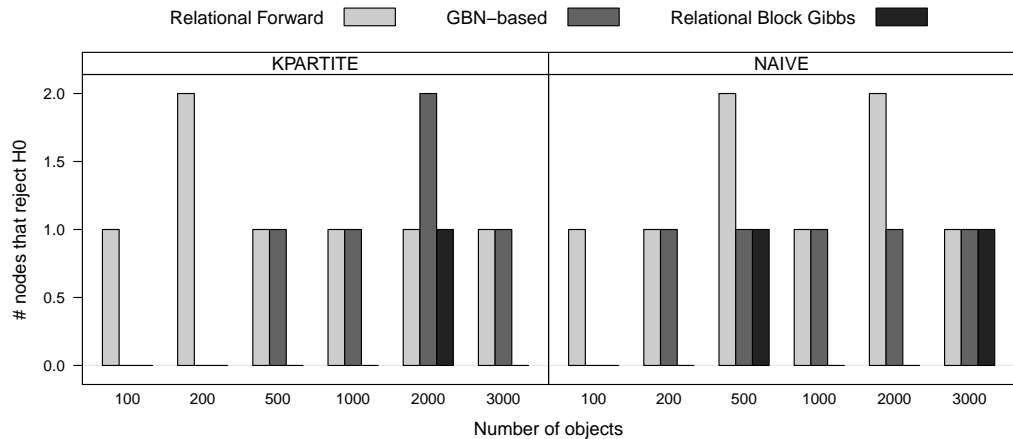


Figure A.10 – Number of nodes that rejected the null hypothesis of the Chi-square goodness-of-fit test on naïve and k -partite graph-based skeletons of different size. Lower values are better here.

A.3 Conclusion

From these experiments, we can conclude that relational forward sampling is very effective for generating very big datasets as it consumes way less amount of time compared to GBN-based sampling and RBG sampling algorithms. In terms of execution time, it is difficult to say whether RBG sampling performs better than GBN-based sampling because their performance highly depends on the complexity of relational skeletons. It is, however, possible to alter the performance of RBG sampling by adjusting the value of burn-in. Results show that higher burn-in values produce good samples but may take too long to complete the sampling process depending on the complexity of the skeleton. Thus, as a strategy to generate big datasets in less time using RBG sampling, we repeatedly generate big samples using a small value of burn-in until we obtain samples where no nodes reject the null hypothesis of Chi-squared goodness-of-fit test. We apply this strategy for obtaining big datasets in our next experiments.



Using PILGRIM

This appendix has been provided as a users' guide for using PILGRIM (Relational and Applications). In Section B.1, we will illustrate through code how to define a PRM. In Section B.2, we will provide an example of instantiating a PRM over a relational skeleton to make inferences. In Section B.3, we will show the usage of some utility methods. Generation of random datasets will be illustrated in Section B.4. An example of a recommender system will be shown in Section B.5.

B.1 Defining a PRM(-SA)

To construct a PRM/PRM-SA using PILGRIM-Relational, we need to define the following components: (1) the relational schema, (2) the dependency structure, and (3) the parameters. Thus, the first task is to get a `RelationalSchema` object. Once it is available, a PRM (or a PRM-SA) can be defined as a RBN (or RBNSA) object. We then need to construct the probabilistic dependency among the random variables present in the PRM, and assign a CPD to each random variable. PILGRIM-Relational offers more than one way to obtain each of the three components.

B.1.1 Defining a relational schema

PILGRIM-Relational implements three ways of defining a relational schema suitable for the following three different situations.

Case I When the schema is well-defined but does not exist physically as a database,

Case II When the schema exists as a database, and

Case III When a random schema is needed.

In the following, we will show examples for each of these cases.

Case I: Manual definition

To define a relational schema, we need classes, their attributes (descriptive attributes + one primary key¹), attribute domains, and reference slots. The following

1. Currently, PILGRIM-Relational supports single primary key only

example defines a schema containing three classes, *users*, *movie*, and *rating*. Each of them has a primary key 'id', and some attributes. The *users* class is a spatial class and has a spatial attribute 'location'. The *rating* is a relationship class, and has two reference slots: *movie_id*, and *user_id*, which refer to *movie*, and *users* respectively.

```
// Define a relational schema

// Classes
boost::shared_ptr<prm::spatial::SpatialClass> user(new prm::spatial::
    SpatialClass("users"));
boost::shared_ptr<prm::Class> movie("movie"), rating("rating");

// Primary key
std::vector<std::string> pks;
pks.push_back("id");

// Attribute domains
boost::shared_ptr<prm::Domain> genreDomain(new prm::MultinomialDomain(
    "Action, Comedy"));
boost::shared_ptr<prm::Domain> genderDomain(new prm::MultinomialDomain(
    "Male, Female"));
boost::shared_ptr<prm::Domain> ageDomain (new prm::MultinomialDomain("
    1, 2, 3"));
boost::shared_ptr<prm::Domain> likesDomain (new prm::MultinomialDomain(
    "Y, N"));
boost::shared_ptr<prm::Domain> userDomain (new prm::MultinomialDomain(
    ));
boost::shared_ptr<prm::Domain> movieDomain (new prm::MultinomialDomain(
    ));

// Non-spatial Attributes
prm::Attribute gender("gender", genderDomain);
prm::Attribute age("age", ageDomain);
prm::Attribute genre("genre", genreDomain);
prm::Attribute like("likes", likesDomain);

// Spatial attributes
prm::spatial::SpatialAttribute location("location");

// Reference slots
prm::Attribute userRef("user_id", userDomain);
prm::Attribute movieRef("movie_id", movieDomain);

// Add attributes to the respective classes
movie->addAttribute(genre);
movie->setPK(pks);

user->addAttribute(age);
user->addAttribute(gender);
user->addSpatialAttribute(location);
user->setPK(pks);

rating->addAttribute(like);
rating->addAttribute(userRef);
rating->addAttribute(movieRef);
rating->setPK(pks);

// Add classes to the schema
schema->addClassRef(movie);
```

```

schema->addClassRef(user);
schema->addClassRef(rating);

// Add reference slots
schema->addReferenceSlot("rating", "movie_id", "movie");
schema->addReferenceSlot("rating", "user_id", "users");

```

Case II: Importing a schema from a database

PILGRIM-Relational can automatically construct a `RelationalSchema` object by importing the schema from a PostgreSQL database. For this, we need to provide connection details, which include the name of ODBC data source, database user, password, and the name of the database that contains the schema to be imported. This is done as follows:

```

// Define DataSource object
DataSource dataSource;
dataSource.name = "DSN"; /* Name of the ODBC data source */
dataSource.user = "postgres_user";
dataSource.password = "password_of_postgres_user";
dataSource.database = "Name_of_the_database";

```

Now, a specific schema from the database mentioned in `dataSource` can be imported with the following two lines:

```

// Import schema

boost::shared_ptr<prm::RelationalSchema> schema(new prm::
    RelationalSchema());

// Import the complete schema
prm::utils::SchemaUtility<prm::DataSource>::importSchema(schema,
    dataSource);

```

The above method imports the complete schema. If class names to import are provided as follows, then partial import will be performed.

```

// Import selected classes only

/*
Assuming that the schema many classes and only the following three are
to be imported
*/
std::vector<std::string> classes;
classes.push_back("users");
classes.push_back("rating");
classes.push_back("movie");

boost::shared_ptr<prm::RelationalSchema> schemaSelectedClasses(new prm
    ::RelationalSchema());
prm::utils::SchemaUtility<prm::DataSource>::importSchema(
    schemaSelectedClasses, dataSource, classes);

```

Case III: Defining a random schema

In order to get a random relational schema, we need to first choose the strategy for generating reference slots. Currently, two strategies available for generating random reference slots in a schema [DAG: RefSlotsGenerationFromPolytree](#) and

RefSlotsGenerationWithPMMixed (Ben Ishak [2015]). Then, a random non-spatial relational schema can be generated with the help of RelationalSchemaGenerator for the selected RefSlotsGenerationStrategy whereas a random spatial relational schema can be obtained from SpatialRelationalSchemaGenerator. The following code is an example of generating a random spatial relational schema from the strategy RefSlotsGenerationFromPolytree.

```
// Define a random schema

/*
Choose a strategy for generating reference slots. There are two
strategies: RefSlotsGenerationFromPolytree and
RefSlotsGenerationWithPMMixed
*/
boost::shared_ptr<prm::RefSlotsGenerationStrategy> refSlotGenStrategy(
    new prm::RefSlotsGenerationFromPolytree(nbOfRefSlots));
/*
RefSlotsGenerationWithPMMixed can be created in the following way.
*/
// boost::shared_ptr<prm::RefSlotsGenerationWithPMMixed>
// refSlotGenStrategy(new prm::RefSlotsGenerationWithPMMixed(
//     maxInducedWidth, maxNodeDegree, maxNbEdges));

/*
Generate a random relational schema using the selected reference slot
generation strategy
*/
boost::shared_ptr<prm::RelationalSchema> schema = prm::
    SpatialRelationalSchemaGenerator::generateSchema(nbOfSpatialClasses
    , nbNonSpatialClasses, refSlotGenStrategy);
```

B.1.2 Defining a dependency structure

Before defining the dependency structure, we need to obtain an RBN (or RBNSA) object which represents a PRM (or a PRM-SA respectively). This object then needs to be initialized to construct the set of random variables present in the PRM. For a PRM-SA, we need to specify the partition algorithm for each spatial attribute in the relational schema. Only after that, dependencies among these random variables can be specified, and initialize spatial ref. attributes using these partition algorithms. In the following we illustrate how an RBNSA object can be constructed.

```
// Initialize a PRM

/* PRM-SA */
boost::shared_ptr<prm::spatial::RBNSA> rbnsa(new prm::spatial::RBNSA(
    schema));

/* Initialize PRM-SA */
rbnsa->init();

/* Assign a partition algorithm to each spatial attribute (Here we
have only one spatial attribute) */
auto users(boost::dynamic_pointer_cast<prm::spatial::SpatialClass>(
    schema->getClassRef("users")));

if (users)
{
```



```

boost::shared_ptr<prm::Attribute> spatialRefAttrib(users->
    getSpatialRefAttributeOf("location"));

/* Define partition algorithm */
unsigned int nbPartitions = 3;
boost::shared_ptr<prm::spatial::ISpatialPartitionAlgorithm>
    spatialPartitionAlgo(new prm::spatial::KMeans(users->
        getName(), "location", spatialRefAttrib->getName(),
        nbPartitions));
users->setPartitionAlgo(spatialRefAttrib->getName(),
    spatialPartitionAlgo);

/* Initialize spatial ref attributes */
rbnsa->initSpatialRefAttributes();
}

```

Similar to the relational schema definition, PILGRIM-Relational provides three ways of defining the dependency structure of a PRM for the following three different situations.

Case I When the dependency is well-defined or known in advance (e.g., through experts' knowledge),

Case II When the dependency is not known in advance and needs to be learned from data, and

Case III When a random dependency structure is needed.

Case I: Manual definition

To define a dependency structure manually, we only need to specify parent-child relations in the following way.

```

// Define probabilistic dependencies
rbnsa->setParents("rating.likes", "rating.user_id>users.age");
rbnsa->setParents("movie.genre", "MODE(~rating.movie_id>rating.likes)");
rbnsa->setParents("users.age", "users.gender");

```

Here, the method `setParents` takes two arguments. The first one is the name of the child node, and the second one is comma-separated list of parent nodes (relational attributes). Each node is named in the format `[ClassName].[AttributeName]`, and each parent relational attribute is specified in the following format.

`[Aggregator]([SlotChain]>[ParentNodeName]).`

`[SlotChain]` is expressed as a sequence of reference slots or inverse slots as follows.

`[ρ_1]>[ρ_2]>...>[ρ_n]`

Each slot ρ is in the format `[ReferringClassName].[ReferenceSlotName]`. If it is an inverse slot, then it is preceded by a tilde (\sim) symbol.

Case II: Learning structure from data

All PRM structure learning algorithms implemented in PILGRIM-Relational are based on relational greedy search algorithm, which is implemented in class RGS. Classes RGS, `AlgoRMMHC`, `AlgoRMMPC`, and `AlgoRMMPCBar` are available for learning regular

PRM, `AlgoRGSRU` for **PRM-RU**, and `AlgoRGSCU` for **PRM-CU**. **PILGRIM-Relational** implements four different algorithms to learn a **PRM-SA** (Algorithms 10, 14, 15, and 16). In this section, we will show examples for learning a **PRM-SA** only.

Before applying a **PRM-SA** learning algorithm, spatial ref. attributes need to be initialized in the database instance because initializing a **PRM-SA** adapts the underlying relational schema only but not the instance of the schema. Thus, we need to apply partition functions on all associated spatial ref. attributes. This is done as follows.

```

/*
Initialize the spatial ref. attribute in our database before applying
a PRM-SA learning algorithm
*/
boost::shared_ptr<prm::DBInstance> instance(new prm::DBInstance(schema
, "SchemaName"));
prm::spatial::SpatialPartitionFunction::partition(instance, "users", "
location", "C_location", "P_users_location");

```

After this initialization, we can proceed with one of the four **PRM-SA** structure learning algorithms.

Naïve approach In **PILGRIM-Relational**, the `RGS` class can be customized by injecting required operators. By default, it performs ‘Add_edge’, ‘Delete_edge’, and ‘Revert_edge’ operations to learn the structure of a **PRM**. We can add ‘Increase_k’ and ‘Decrease_k’ (cf. Algorithms 11 and 12 respectively) to learn a **PRM-SA** in naïve way (cf. Algorithm 10 in Section 8.3).

```

// Naive approach to learn a PRM-SA

// Required operations
std::vector<prm::algo::GraphOperationType> allowedOperationTypes
= boost::assign::list_of(prm::algo::ADD_EDGE)
                        (prm::algo::DELETE_EDGE)
                        (prm::algo::REVERT_EDGE)
                        (prm::algo::INCREASE_K)
                        (prm::algo::DECREASE_K);

// Cache for storing computed scores
boost::shared_ptr<RBNCache> cache(new RBNCache(10000000));

// Scoring function
boost::shared_ptr<RBNDecomposableScore> S(new RScoreBD(instance, rbnsa
, 1UL, cache.get()));

// Structure learning algorithm
prm::algo::RGS rgs;
rgs.setAllowedOperationTypes(allowedOperationTypes);
rgs.setScore(S);

unsigned int maxSlotChainLength = 3;
rgs.learnStructure(rbnsa, maxSlotChainLength);

```

Adaptative approach Three variations of adaptative approach of learning a **PRM-SA** are implemented in **PILGRIM-Relational**.

Version 1 (Algorithm 14) The first version, which finds the optimal cardinality of all spatial ref. attributes after a complete greedy search for the given slot chain

length, is implemented in class `AdaptativeRGS1`. It can be applied in the following way.

```

/* Adaptative approach for learning the structure of a PRM-SA (version
  1) */

// Cache for storing computed scores
boost::shared_ptr<RBNCache> cache(new RBNCache(10000000));

// Scoring function
boost::shared_ptr<RBNDecomposableScore> S(new RScoreBD(instance, rbnsa
  , 1UL, cache.get()));

// Structure learning algorithm
prm::algo::AdaptativeRGS1 rgs;
rgs.setScore(S);

unsigned int maxSlotChainLength = 3;
rgs.learnStructure(rbnsa, maxSlotChainLength);

```

Version 2 (Algorithm 15) This version finds the optimal cardinality of all spatial ref. attributes in each iteration of greedy search algorithm for the given slot chain. It is implemented in class `AdaptativeRGS2`, and can be applied in similar fashion as `AdaptativeRGS1`.

```

/* Adaptative approach for learning the structure of a PRM-SA (version
  2) */

// Cache for storing computed scores
boost::shared_ptr<RBNCache> cache(new RBNCache(10000000));

// Scoring function
boost::shared_ptr<RBNDecomposableScore> S(new RScoreBD(instance, rbnsa
  , 1UL, cache.get()));

// Structure learning algorithm
prm::algo::AdaptativeRGS2 rgs;
rgs.setScore(S);

unsigned int maxSlotChainLength = 3;
rgs.learnStructure(rbnsa, maxSlotChainLength);

```

Version 3 (Algorithm 16) For this version, we have specialized add, delete and revert operations for spatial context such that these operators perform optimization of the cardinality of spatial ref. attributes after performing standard add, delete or revert operations. Thus, to learn a PRM-SA using this version of adaptative approach, it is as simple as injecting these new specialized operators to RGS.

```

/* Adaptative approach for learning the structure of a PRM-SA (version
  3) */

// Required operations
std::vector<prm::algo::GraphOperationType> allowedOperationTypes
  = boost::assign::list_of(prm::algo::SPATIAL_ADD_EDGE)
    (prm::algo::SPATIAL_DELETE_EDGE)
    (prm::algo::SPATIAL_REVERT_EDGE);

```

```

// Cache for storing computed scores
boost::shared_ptr<RBNCache> cache(new RBNCache(10000000));

// Scoring function
boost::shared_ptr<RBNDecomposableScore> S(new RScoreBD(instance, rbnsa
    , 1UL, cache.get()));

// Structure learning algorithm
prm::algo::RGS rgs;
rgs.setAllowedOperationTypes(allowedOperationTypes);
rgs.setScore(S);

unsigned int maxSlotChainLength = 3;
rgs.learnStructure(rbnsa, maxSlotChainLength);

```

Case III: Random dependency structure

`RBNGenerateDependencies` is responsible for generating a random dependency structure.

```

// Generate random dependencies
RBNGenerateDependencies::generate(rbnsa, maxSlotChainLength,
    maxInducedWidth, maxParentsPerNode, maxNumOfDependencies);

```

B.1.3 Defining parameters

Again, PILGRIM-Relational addresses the following three scenarios for generating parameters for a [PRM](#) (or a [PRM-SA](#)).

Case I When the parameters are well-defined or known in advance (e.g., through experts' knowledge),

Case II When the parameters are not known in advance and need to be learned from data, and

Case III When random parameters are needed.

Case I: Manual definition

If the [CPDs](#) are known, they can be assigned to the corresponding nodes by calling the method `RBN::setDistributionTable()`, which takes three arguments: the node, its parents, and the conditional probability table as a vector of probability values. The following code assigns [CPDs](#) to the nodes of the structure defined manually in the previous section (Section [B.1.2](#)).

```

// Define CPDs

/* CPDs can be defined manually in the following way */
/* P(users.age | users.gender)
plProbValue probas_ua[] = {0.2, 0.7, 0.1, 0.2, 0.3, 0.5};
rbnsa->setDistributionTable("users.age", rbnsa->getParents("users.age
    "), initVector(probas_ua, 6));

/* P(users.gender) */
plProbValue probas_ug[] = {0.2, 0.8};
rbnsa->setDistributionTable("users.gender", rbnsa->getParents("users.
    gender"), initVector(probas_ug, 2));

```

```

/* P(rating.likes | rating.user_id>users.age) */
plProbValue probas_rl[] = {0.9, 0.1, 0.4, 0.6, 0.3, 0.7};
rbnsa->setDistributionTable("rating.likes", rbnsa->getParents("rating.
likes"), initVector(probas_rl, 6));

/* P(movie.genre | MODE(~rating.movie_id>rating.likes))*/
plProbValue probas_mg[] = {0.2, 0.8, 0.9, 0.1, 0.4, 0.6};
rbnsa->setDistributionTable("movie.genre", rbnsa->getParents("movie.
genre"), initVector(probas_mg, 6));

/* P(users.C_location) */
plProbValue probas_ucl[] = {0.1, 0.6, 0.3};
rbnsa->setDistributionTable("users.C_location", rbnsa->getParents("
users.C_location"), initVector(probas_ucl, 3));

```

Case II: Learning parameters from data

A parameter estimation method, and a complete instantiation of the relational schema are needed to learn parameters of a [PRM/PRM-SA](#) from data. Following parameter estimation methods have been implemented in PILGRIM-Relational: Laplace, MaximumAPosteriori, ExpectedAPosteriori, and MaximumLikelihood.

```

// Learn parameters from data

// Choose a parameter estimation method.
/* Currently available: Laplace, MaximumAPosteriori,
ExpectedAPosteriori and MaximumLikelihood */
boost::shared_ptr<prm::ParameterEstimationMethod>
paramEstimationMethod(new prm::Laplace());

// The source of data
boost::shared_ptr<prm::Instance> instance(new prm::Instance(schema));

rbnsa->learnParameters(instance, paramEstimationMethod.get());

```

Case III: Random parameters

Generating random parameters for all nodes in a [PRM/PRM-SA](#) is as simple as the following one-liner.

```

// Assign random CPDs to the nodes in a PRM
RBNGenerateParameters::computeAllCPTs(rbnsa);

```

B.2 Instantiating a PRM for making inference

RBN implements the method `generateGroundBayesianNetwork` for generating a [GBN](#) of type `BayesianNetwork`. This [GBN](#) can then be used for making inferences in the following way.

```

// Generate a ground Bayesian network
prm::BayesianNetwork gbn = rbnsa->generateGroundBayesianNetwork(
instance);

// Make inference : P(rating.like | users.age)
plSymbol user_l = gbn.get_node("l_rating.age");

```

```

plSymbol rating_1 = gbn.get_node("1_rating.like");
plValues evidence(rating_1);
evidence[rating_1] = 1;

plProbValue val = gbn.get_joint_distribution().ask(user_1, rating_1).
    instantiate(evidence).compute(1);

```

B.3 Utility methods

PILGRIM-Relational provides some useful methods, such as the ones for exporting/importing [PRMs](#) and relational schema.

B.3.1 Exporting/Importing a PRM

`RBNSerializeUnserialize` provides utility methods for exporting/importing a [PRM](#) into/from an *EXtensible Markup Language (XML)* file.

```

// Export a PRM (an RBN instance) into an XML file
RBNSerializeUnserialize::serialize(rbn, "Path/to/the/output/folder", "
    outputFilename.xml");

// Import a PRM from an XML file
boost::shared_ptr<prm::RBN> rbn = RBNSerializeUnserialize::serialize("
    Path/to/the/folder", "inputFilename.xml");

```

B.3.2 Exporting a relational schema into a database

`SchemaUtility` implements the utility method for exporting a `RelationalSchema` object into a database specified in a `DataSource` object. Currently, PILGRIM-Relational can export a relational schema into a PostgreSQL database only.

```

// Export a relational schema

/*
The exportSchema() method takes the following arguments:
schema = The RelationalSchema object to exported,
dataSource = The DataSource object specifying connection details and
    the database where the schema is to be exported,
replaceExisting = Whether to replace the old one if exists, and
isSpatialSchema = Whether it is a spatial schema or a non-spatial one.
*/
bool replaceExisting = true;
bool isSpatialSchema = true;
prm::utils::SchemaUtility<prm::DataSource>::exportSchema(schema,
    dataSource, replaceExisting, isSpatialSchema);

```

B.4 Generating datasets from a PRM

Using PILGRIM-Relational, we can generate spatial as well as non-spatial datasets. This is done by sampling a [PRM](#) over a relational skeleton as illustrated in [Section B.4.2](#). If a relational skeleton is not available, we can generate a random relational skeleton as shown in [Section B.4.1](#).

B.4.1 Generating a random skeleton

Two types of relational skeletons can be generated using PILGRIM-Relational: (1) naïve skeleton, and *k*-partite graph-based skeleton (see Section 2.7.2 in Chapter 2 for more details on these skeletons). In the following, we will show how to generate these types of skeletons for spatial as well as non-spatial databases. Current implementation stores the generated skeleton into a PostgreSQL database.

Generating a random relational skeleton

A random relational skeleton can be generated in the following way.

```
// Generate a non-spatial relational skeleton
/* Note: The corresponding relational schema must exist in the
   database. If it does not, export the schema first. */

/* Create an Instance object with the corresponding RelationalSchema
   object and the name of the database */
boost::shared_ptr<prm::Instance> instance(new prm::DBInstance(schema,
    databaseName));

/* Generate a K-partite graph-based skeleton */
boost::shared_ptr<prm::SkeletonGenerationStrategy>
    skeletonGenerationStrategy(new prm::KPartiteGraphGenerator(alpha));

// For a naive skeleton, use NaiveSkeletonGenerator
/*
boost::shared_ptr<prm::SkeletonGenerationStrategy>
    skeletonGenerationStrategy(new NaiveSkeletonGenerator());
*/

/* Generate a non-spatial relational skeleton using
   RelationalSkeletonGenerator */
boost::shared_ptr<prm::RelationalSkeletonGenerator> skeletonGenerator(
    new prm::RelationalSkeletonGenerator(skeletonGenerationStrategy));

// For a spatial relational skeleton, use
   SpatialRelationalSkeletonGenerator
/*
boost::shared_ptr<prm::SpatialRelationalSkeletonGenerator>
    skeletonGenerator(new prm::SpatialRelationalSkeletonGenerator(
    skeletonGenerationStrategy));
*/

/*
Note:
For k-partite graph-based skeleton, numberOfObjects = total number of
   objects in the dataset
For naive skeleton, numberOfObjects = approximate number of objects
   per class
*/
skeletonGenerator->generate(instance, numberOfObjects);
```

B.4.2 Sampling a PRM

To sample a PRM or a PRM-SA, we need to choose a sampling algorithm. Three sampling algorithms, explained in Section 2.7.2, have been implemented in PILGRIM-

Relational. We can then sample a [PRM](#) or a [PRM-SA](#) with the help of `RBNSampler` or `RBNSASampler` respectively.

Sampling a regular PRM

```
// Sample a (regular) PRM

/* First, choose the strategy for sampling. */
// For GBN-based sampling
boost::shared_ptr<prm::RBNSamplingStrategy> samplingStrategy(new prm::
    GBNBasedSampling());

// For forward sampling
/*
boost::shared_ptr<prm::RBNSamplingStrategy> samplingStrategy(new prm::
    ForwardSampling());
*/

// For relational Gibbs block sampling
/*
boost::shared_ptr<prm::RBNSamplingStrategy> samplingStrategy(new prm::
    GibbsSampling(burnIn));
*/

/* Initialize an RBNSampler with the chosen sampling strategy */
boost::shared_ptr<prm::RBNSampler> sampler(new prm::RBNSampler(
    samplingStrategy));

/* Sample the RBN object (rbn) for the given relational skeleton,
   which is an Instance object (instance) */
sampler->sample(rbn, instance);
```

Sampling a PRM-SA

A random spatial dataset can be generated in the similar way as its non-spatial counterpart. The only difference is that we need to specify how to generate spatial attributes. Among the three cases of sampling attributes presented in Section 8.4.2, only case II has been implemented in the current version. This strategy, which is implemented as `SamplingFromCollectionOfPointsStrategy`, needs a collection of points (latitude and longitude values), and the degree of freedom.

```
// Sample a PRM-SA

// A collection of points for sampling spatial attributes
std::vector<std::vector<double>> meanPoints;

// Latitude and longitude of Nantes
std::vector<double> nantes =
    boost::assign::list_of(-1.5545) (47.2185);

// Latitude and longitude of Paris
std::vector<double> paris =
    boost::assign::list_of(2.3475569) (48.8588589);

// Latitude and longitude of Berlin
std::vector<double> berlin=
    boost::assign::list_of(13.4251364) (52.5075419);
```



```

meanPoints.push_back(nantes);
meanPoints.push_back(paris);
meanPoints.push_back(berlin);

/* Maximum number of points to use from the collection of points */
// Choose a strategy for sampling spatial attributes
// Currently, only SamplingFromCollectionOfPointsStrategy is available
unsigned int maxNumPointsToUse = 2;
boost::shared_ptr<prm::AttributeSamplingStrategy> attSamplingStrategy(
    new prm::SamplingFromCollectionOfPointsStrategy(meanPoints,
        maxNumPointsToUse));

// Initialize RBNSASampler with the chosen sampling strategies
boost::shared_ptr<prm::RBNSASampler> rbnSASampler(new prm::
    RBNSASampler(attSamplingStrategy, rbnSamplingStrategy));

// Sample the RBNSA object
rbnSASampler->sample(rbnsa, instance);

```

B.5 Working with PILGRIM-Recommender

PILGRIM-Applications project currently implements [Huang et al. \[2004\]](#)'s recommendation method (see Sections 4.2.2 and 9.4 for details). In this section, we will illustrate how to deal with this recommendation method. We will show how to construct a recommendation model, and make predictions from it.

B.5.1 Defining a recommendation model

[Huang et al. \[2004\]](#)'s recommendation approach has been implemented in class `HuangRecoModel`. In the following example, we build a recommender system for a non-spatial version of the schema (movie-users-rating schema) manually created in Section B.1.1. To construct an object of `HuangRecoModel`, we first import our relational schema. We then initialize the `HuangRecoModel` object with the relational schema, and specify the name of the item class, the user class, and the target variable. In [Huang et al. \[2004\]](#)'s model, the target variable is `Transaction.exists`. However, in this example, we consider our target variable as `rating.rating`.

```

// Construct Huang's recommendation model

/* Import the input relational schema from the database */
prm::DataSource dataSource(prm::ConnectionManager::
    getDefaultDataSourceDetails());
boost::shared_ptr<prm::RelationalSchema> schema(new prm::
    RelationalSchema());
dataSource.database = databaseName;
prm::utils::SchemaUtility<prm::DataSource>::importSchema(schema,
    dataSource);

/* Initialize an RBN object for the input schema */
boost::shared_ptr<prm::RBN> rbn(new prm::RBN(schema));
rbn->init();

// Construct Huang's recommendation model
/* Initialize HuangRecoModel with the relational schema, the name of
    the item class, the user class and the target variable. */

```

```
boost::shared_ptr<prm::rs::model::HuangsRecoModel> recoModel(new prm::
    rs::model::HuangsRecoModel(schema, "movie", "users", "rating.rating
    "));
```

Huang et al. [2004] propose to use a naïve Bayesian classifier for recommendation. In PILGRIM-Recommender, this model can be either defined manually when the structure is known (e.g., when provided by experts) or learned from data. In the following, we will show examples for both situations.

Manual definition

To manually define a naïve Bayesian classifier for `HuangRecoModel`, we need to create a `NaiveBayesianClassifier` object in the following way.

```
// Define Huang's recommendation model manually

/* Target variable */
plSymbol rating = getSymbol("rating.rating");

/* Naive Bayesian classifier with rating.rating as the root variable
   */
boost::shared_ptr<prm::NaiveBayesianClassifier> classifier(new prm::
    NaiveBayesianClassifier("NBN", rating, rbn->getNode(rating.name()))
    );

/* Other attribute variables of the Naive BC */
plSymbol genre = getSymbol("movie.genre");
plSymbol age = getSymbol("users.age");

classifier->add_node(age, rbn->getVariable("rating.userid>users.age"))
    ;
classifier->add_node(genre, rbn->getVariable("rating.movieid>movie.
    genre"));

/* Set the model to the recommender */
recoModel->setModel(classifier);

// Learn the parameters of this model from data
boost::shared_ptr<prm::Instance> instance(schema, databaseName);
recoModel->setParameterEstimationMethod(new prm::Laplace());
recoModel->learnModelParameters(instance);
```

Learning a recommendation model from data

A naïve Bayesian classifier can be learned from data with the help of the method `learnModel()` of `HuangRecoModel`. This method identifies the Markov blanket of the target node and builds a naïve Bayesian classifier from the Markov blanket. It also learns the parameters from data too.

```
// Learn Huang's recommendation model

boost::shared_ptr<prm::Instance> instance(schema, databaseName);

recoModel->learnModel(instance);
```

B.5.2 Making recommendations

Top-N recommendations can be obtained from the method `topN` of `HuangRecoModel` in the following manner.

```
// Get top-N recommendations for a target user

/* Target user */
prm::Object targetUser("userId", schema, schema->getClassRef("users"))
;
targetUser->set("attribute1Name", "attribute1Value");
targetUser->set("attribute2Name", "attribute2Value");

/* Preferred labels, e.g., 4 and 5 in 5-scale rating, true in boolean
   rating */
std::set<std::string> preferredLabels;
preferredLabels.insert("preferredLabel1");
preferredLabels.insert("preferredLabel2");

/* Get top-N recommendations*/
std::vector<model::Relevance> topN = recoModelTest->getTopN(instance,
    targetUser, N, preferredLabels);
```




Detailed Results of PRM-SA Learning Algorithm Evaluation

In this chapter, we provide the detailed results of our experiment for evaluating PRM-SA structure learning algorithms (see Section 8.5 for details about the experiment). Tables C.1 – C.9 present average precision, recall, and F-score along with standard deviations for each of the five algorithms being evaluated on 7 models of Figure 8.6. Figures C.1 – C.4 show the results of Nemenyi test. Average precision, recall, and F-score for spatial attributes of each model are reported in Tables C.10 – C.15. Table C.16 shows average normalized mutual information (NMI) between the spatial partition learned during the experiment, and the original spatial partition for the spatial attribute (*User.location*) of each model. Finally, Table C.17 shows the absolute difference between Bayesian Dirichlet score of the learned models and that of the learned ones.

Table C.1 – Average \pm standard deviation of Hard Precision for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.619 \pm 0.209	0.619 \pm 0.209	0.619 \pm 0.209	<u>0.400</u> \pm 0.075	0.643 \pm 0.202
A2	0.361 \pm 0.195	0.361 \pm 0.195	<u>0.125</u> \pm 0.209	0.236 \pm 0.123	0.500 \pm 0.333
B1	0.633 \pm 0.178	0.633 \pm 0.178	0.595 \pm 0.189	<u>0.469</u> \pm 0.168	0.605 \pm 0.110
C1	0.476 \pm 0.224	0.452 \pm 0.230	0.548 \pm 0.263	<u>0.390</u> \pm 0.150	0.452 \pm 0.209
C2	<u>0.393</u> \pm 0.157	0.771 \pm 0.303	0.521 \pm 0.107	0.719 \pm 0.200	0.676 \pm 0.241
C3	<u>0.462</u> \pm 0.118	0.486 \pm 0.103	0.543 \pm 0.228	0.490 \pm 0.098	0.590 \pm 0.211
D1	0.500 \pm 0.136	0.500 \pm 0.136	<u>0.421</u> \pm 0.263	0.545 \pm 0.244	0.548 \pm 0.209
Overall	0.495 \pm 0.191	0.550 \pm 0.228	0.489 \pm 0.252	<u>0.469</u> \pm 0.204	0.575 \pm 0.220

Table C.2 – Average \pm standard deviation of Hard Recall for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.619 \pm 0.126	0.619 \pm 0.126	0.619 \pm <u>0.126</u>	<u>0.571</u> \pm 0.163	0.619 \pm 0.126
A2	0.278 \pm 0.136	0.278 \pm 0.136	<u>0.111</u> \pm 0.172	0.278 \pm 0.136	0.333 \pm 0.211
B1	0.536 \pm 0.094	0.536 \pm 0.094	<u>0.500</u> \pm 0.000	0.536 \pm 0.094	<u>0.500</u> \pm 0.000
C1	0.286 \pm 0.173	0.286 \pm 0.173	0.464 \pm 0.267	0.357 \pm 0.134	<u>0.250</u> \pm 0.144
C2	<u>0.286</u> \pm 0.107	0.486 \pm 0.195	0.400 \pm 0.163	0.543 \pm 0.151	<u>0.286</u> \pm 0.107
C3	<u>0.464</u> \pm 0.094	0.500 \pm 0.144	0.536 \pm 0.225	0.571 \pm 0.189	<u>0.464</u> \pm 0.094
D1	0.286 \pm 0.107	0.286 \pm 0.107	0.257 \pm 0.19	0.400 \pm 0.115	<u>0.200</u> \pm 0.000
Overall	0.396 \pm 0.177	0.430 \pm 0.188	0.419 \pm 0.233	0.469 \pm 0.173	<u>0.380</u> \pm 0.181

Table C.3 – Average \pm standard deviation of hard F-score for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.611 \pm 0.144	0.611 \pm 0.144	0.611 \pm 0.144	<u>0.465</u> \pm 0.100	0.624 \pm 0.145
A2	0.311 \pm 0.156	0.311 \pm 0.156	<u>0.114</u> \pm 0.181	0.254 \pm 0.127	0.383 \pm 0.221
B1	0.568 \pm 0.075	0.568 \pm 0.075	0.534 \pm 0.064	<u>0.494</u> \pm 0.127	0.543 \pm 0.051
C1	0.354 \pm 0.192	0.347 \pm 0.194	0.497 \pm 0.262	0.370 \pm 0.133	<u>0.320</u> \pm 0.167
C2	<u>0.323</u> \pm 0.112	0.587 \pm 0.224	0.439 \pm 0.136	0.612 \pm 0.158	0.377 \pm 0.092
C3	<u>0.457</u> \pm 0.093	0.486 \pm 0.102	0.535 \pm 0.223	0.523 \pm 0.136	0.505 \pm 0.109
D1	0.359 \pm 0.117	0.359 \pm 0.117	0.313 \pm 0.215	0.448 \pm 0.127	<u>0.287</u> \pm 0.024
Overall	<u>0.429</u> \pm 0.167	0.470 \pm 0.184	0.441 \pm 0.231	0.456 \pm 0.16	0.435 \pm 0.168

Table C.4 – Average \pm standard deviation of Soft Precision for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.619 \pm 0.209	0.619 \pm 0.209	0.619 \pm 0.209	<u>0.400</u> \pm 0.075	0.643 \pm 0.202
A2	0.380 \pm 0.155	0.380 \pm 0.155	<u>0.144</u> \pm 0.201	0.250 \pm 0.091	0.519 \pm 0.302
B1	0.633 \pm 0.178	0.633 \pm 0.178	0.595 \pm 0.189	<u>0.469</u> \pm 0.168	0.605 \pm 0.110
C1	0.476 \pm 0.224	0.452 \pm 0.230	0.548 \pm 0.263	<u>0.390</u> \pm 0.15	0.452 \pm 0.209
C2	<u>0.393</u> \pm 0.157	0.771 \pm 0.303	0.521 \pm 0.107	0.719 \pm 0.200	0.676 \pm 0.241
C3	<u>0.462</u> \pm 0.118	0.486 \pm 0.103	0.543 \pm 0.228	0.490 \pm 0.098	0.590 \pm 0.211
D1	0.500 \pm 0.136	0.500 \pm 0.136	<u>0.421</u> \pm 0.263	0.545 \pm 0.244	0.563 \pm 0.194
Overall	0.497 \pm 0.186	0.552 \pm 0.223	0.492 \pm 0.248	<u>0.471</u> \pm 0.200	0.580 \pm 0.212

Table C.5 – Average \pm standard deviation of Soft Recall for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	<u>0.619</u> \pm 0.126	<u>0.619</u> \pm 0.126	<u>0.619</u> \pm 0.126	0.571 \pm 0.163	<u>0.619</u> \pm 0.126
A2	0.296 \pm 0.091	0.296 \pm 0.091	<u>0.130</u> \pm 0.164	0.296 \pm 0.091	0.352 \pm 0.178
B1	0.536 \pm 0.094	0.536 \pm 0.094	<u>0.500</u> \pm 0.000	0.536 \pm 0.094	<u>0.500</u> \pm 0.000
C1	0.286 \pm 0.173	0.286 \pm 0.173	0.464 \pm 0.267	0.357 \pm 0.134	<u>0.250</u> \pm 0.144
C2	<u>0.286</u> \pm 0.107	0.486 \pm 0.195	0.400 \pm 0.163	0.543 \pm 0.151	<u>0.286</u> \pm 0.107
C3	<u>0.464</u> \pm 0.094	0.500 \pm 0.144	0.536 \pm 0.225	0.571 \pm 0.189	<u>0.464</u> \pm 0.094
D1	0.286 \pm 0.107	0.286 \pm 0.107	0.257 \pm 0.190	0.400 \pm 0.115	<u>0.210</u> \pm 0.025
Overall	0.398 \pm 0.172	0.433 \pm 0.183	0.421 \pm 0.229	0.471 \pm 0.167	<u>0.384</u> \pm 0.175

Table C.6 – Average \pm standard deviation of Soft F-score for PRM-SA structure learning algorithms for models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.611 \pm 0.144	0.611 \pm 0.144	0.611 \pm 0.144	<u>0.465</u> \pm 0.100	0.624 \pm 0.145
A2	0.330 \pm 0.112	0.330 \pm 0.112	<u>0.133</u> \pm 0.172	0.270 \pm 0.089	0.402 \pm 0.184
B1	0.568 \pm 0.075	0.568 \pm 0.075	0.534 \pm 0.064	<u>0.494</u> \pm 0.127	0.543 \pm 0.051
C1	0.354 \pm 0.192	0.347 \pm 0.194	<u>0.497</u> \pm 0.262	0.370 \pm 0.133	0.320 \pm 0.167
C2	<u>0.323</u> \pm 0.112	0.587 \pm 0.224	0.439 \pm 0.136	0.612 \pm 0.158	0.377 \pm 0.092
C3	<u>0.457</u> \pm 0.093	0.486 \pm 0.102	0.535 \pm 0.223	0.523 \pm 0.136	0.505 \pm 0.109
D1	0.359 \pm 0.117	0.359 \pm 0.117	0.313 \pm 0.215	0.448 \pm 0.127	<u>0.299</u> \pm 0.023
Overall	0.431 \pm 0.162	0.470 \pm 0.179	0.444 \pm 0.227	0.458 \pm 0.155	<u>0.439</u> \pm 0.161

Table C.7 – Average \pm standard deviation of Soft Precision_{skeleton} of models of Figure 8.6.

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.881 \pm 0.151	0.881 \pm 0.151	0.881 \pm 0.151	<u>0.648</u> \pm 0.089	0.917 \pm 0.144
A2	0.750 \pm 0.204	0.750 \pm 0.204	0.678 \pm 0.264	<u>0.500</u> \pm 0.105	0.889 \pm 0.172
B1	0.829 \pm 0.187	0.829 \pm 0.187	0.786 \pm 0.173	<u>0.714</u> \pm 0.183	0.888 \pm 0.145
C1	0.786 \pm 0.393	0.738 \pm 0.383	0.750 \pm 0.382	<u>0.574</u> \pm 0.140	0.786 \pm 0.393
C2	<u>0.907</u> \pm 0.117	0.971 \pm 0.076	0.948 \pm 0.09	0.943 \pm 0.098	0.971 \pm 0.076
C3	0.829 \pm 0.167	0.829 \pm 0.167	0.848 \pm 0.153	<u>0.736</u> \pm 0.204	0.943 \pm 0.098
D1	0.798 \pm 0.203	0.798 \pm 0.203	<u>0.752</u> \pm 0.239	0.795 \pm 0.102	1.000 \pm 0.000
Overall	0.827 \pm 0.211	0.830 \pm 0.214	0.809 \pm 0.226	<u>0.706</u> \pm 0.187	0.914 \pm 0.186

Table C.8 – Average \pm standard deviation of Soft Recall_{skeleton} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.905 \pm 0.163	0.905 \pm 0.163	0.905 \pm 0.163	0.905 \pm 0.163	0.905 \pm 0.163
A2	<u>0.611</u> \pm 0.136	<u>0.611</u> \pm 0.136	<u>0.611</u> \pm 0.136	<u>0.611</u> \pm 0.136	0.667 \pm 0.211
B1	0.714 \pm 0.173	0.714 \pm 0.173	<u>0.679</u> \pm 0.122	0.857 \pm 0.244	0.750 \pm 0.144
C1	0.464 \pm 0.267	0.464 \pm 0.267	0.643 \pm 0.378	0.536 \pm 0.173	<u>0.429</u> \pm 0.238
C2	0.686 \pm 0.195	0.629 \pm 0.138	0.714 \pm 0.227	0.714 \pm 0.107	<u>0.457</u> \pm 0.223
C3	0.857 \pm 0.244	0.857 \pm 0.244	0.857 \pm 0.244	0.857 \pm 0.283	<u>0.786</u> \pm 0.225
D1	0.457 \pm 0.151	0.457 \pm 0.151	0.486 \pm 0.195	0.629 \pm 0.180	<u>0.400</u> \pm 0.115
Overall	0.672 \pm 0.247	0.664 \pm 0.242	0.701 \pm 0.251	0.732 \pm 0.226	<u>0.627</u> \pm 0.261

Table C.9 – Average \pm standard deviation of Soft F-score_{skeleton} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.883 \pm 0.127	0.883 \pm 0.127	0.883 \pm 0.127	<u>0.745</u> \pm 0.085	0.903 \pm 0.133
A2	0.667 \pm 0.146	0.667 \pm 0.146	0.623 \pm 0.163	<u>0.548</u> \pm 0.112	0.739 \pm 0.169
B1	0.753 \pm 0.136	0.753 \pm 0.136	<u>0.718</u> \pm 0.111	0.771 \pm 0.195	0.806 \pm 0.112
C1	0.578 \pm 0.309	0.565 \pm 0.307	0.685 \pm 0.377	<u>0.549</u> \pm 0.147	0.551 \pm 0.288
C2	0.763 \pm 0.132	0.751 \pm 0.095	<u>0.788</u> \pm 0.152	0.804 \pm 0.062	<u>0.587</u> \pm 0.195
C3	0.833 \pm 0.194	0.833 \pm 0.194	0.846 \pm 0.194	<u>0.786</u> \pm 0.234	0.832 \pm 0.123
D1	0.575 \pm 0.166	0.575 \pm 0.166	0.577 \pm 0.198	0.686 \pm 0.121	<u>0.563</u> \pm 0.121
Overall	0.723 \pm 0.205	0.719 \pm 0.203	0.734 \pm 0.221	<u>0.702</u> \pm 0.171	0.711 \pm 0.211

Table C.10 – Average \pm standard deviation of Hard Precision_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.571 \pm 0.535	0.571 \pm 0.535	0.571 \pm 0.535	0 \pm 0	0.714 \pm 0.488
A2	0.333 \pm 0.516	0.333 \pm 0.516	0.333 \pm 0.516	0 \pm 0	0.667 \pm 0.516
B1	0.429 \pm 0.535	0.429 \pm 0.535	0.286 \pm 0.488	0 \pm 0	0.714 \pm 0.488
C1	0.429 \pm 0.535	0.286 \pm 0.488	0 \pm 0	0 \pm 0	0.571 \pm 0.535
C2	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0.714 \pm 0.488
C3	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0.571 \pm 0.535
D1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0.857 \pm 0.378
Overall	0.25 \pm 0.438	0.229 \pm 0.425	0.167 \pm 0.377	0 \pm 0	0.688 \pm 0.468

Table C.11 – Average \pm standard deviation of Hard Recall_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0
A2	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0
B1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C2	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C3	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
D1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
Overall	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449	0.271 \pm 0.449

Table C.12 – Average \pm standard deviation of hard F-score_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.571 \pm 0.535	0.571 \pm 0.535	0.571 \pm 0.535	0 \pm 0	0.714 \pm 0.488
A2	0.333 \pm 0.516	0.333 \pm 0.516	0.333 \pm 0.516	0 \pm 0	0.667 \pm 0.516
B1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C2	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
C3	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
D1	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0	0 \pm 0
Overall	0.125 \pm 0.334	0.125 \pm 0.334	0.125 \pm 0.334	0 \pm 0	0.188 \pm 0.394

Table C.13 – Average \pm standard deviation of Soft Precision_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.571 \pm 0.535	0.571 \pm 0.535	0.571 \pm 0.535	<u>0</u> \pm 0	0.714 \pm 0.488
A2	0.333 \pm 0.516	0.333 \pm 0.516	0.333 \pm 0.516	<u>0</u> \pm 0	0.667 \pm 0.516
B1	0.5 \pm 0.5	0.5 \pm 0.5	0.429 \pm 0.450	0.143 \pm 0.378	0.881 \pm 0.209
C1	0.714 \pm 0.488	0.571 \pm 0.535	0.643 \pm 0.476	0.226 \pm 0.229	0.714 \pm 0.488
C2	<u>0.071</u> \pm 0.189	0.643 \pm 0.476	0.271 \pm 0.274	0.619 \pm 0.267	0.714 \pm 0.488
C3	<u>0</u> \pm 0	0.036 \pm 0.094	0.19 \pm 0.378	0.179 \pm 0.170	0.571 \pm 0.535
D1	0.357 \pm 0.476	0.357 \pm 0.476	<u>0.333</u> \pm 0.471	0.452 \pm 0.326	1 \pm 0
Overall	0.365 \pm 0.470	0.432 \pm 0.480	0.397 \pm 0.446	0.236 \pm 0.312	0.753 \pm 0.424

Table C.14 – Average \pm standard deviation of Soft Recall_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0
A2	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0	1 \pm 0
B1	0.143 \pm 0.378	0.143 \pm 0.378	<u>0</u> \pm 0	0.143 \pm 0.378	0.286 \pm 0.488
C1	0.143 \pm 0.244	0.143 \pm 0.244	0.500 \pm 0.408	0.286 \pm 0.267	<u>0.071</u> \pm 0.189
C2	<u>0.107</u> \pm 0.134	0.357 \pm 0.244	0.250 \pm 0.204	0.429 \pm 0.189	<u>0.107</u> \pm 0.134
C3	<u>0</u> \pm 0	0.143 \pm 0.378	0.143 \pm 0.378	0.571 \pm 0.535	<u>0</u> \pm 0
D1	0.143 \pm 0.178	0.143 \pm 0.178	0.190 \pm 0.262	0.333 \pm 0.192	<u>0.095</u> \pm 0.163
Overall	0.349 \pm 0.442	0.406 \pm 0.442	0.429 \pm 0.444	0.528 \pm 0.416	0.352 \pm 0.454

Table C.15 – Average \pm standard deviation of Soft F-score_{spatial} of models of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	0.571 \pm 0.535	0.571 \pm 0.535	0.571 \pm 0.535	<u>0</u> \pm 0	0.714 \pm 0.488
A2	0.333 \pm 0.516	0.333 \pm 0.516	0.333 \pm 0.516	<u>0</u> \pm 0	0.667 \pm 0.516
B1	0.095 \pm 0.252	0.095 \pm 0.252	<u>0</u> \pm 0	0.143 \pm 0.378	0.210 \pm 0.360
C1	0.190 \pm 0.325	0.190 \pm 0.325	0.548 \pm 0.416	0.248 \pm 0.239	0.095 \pm 0.252
C2	<u>0.048</u> \pm 0.126	0.418 \pm 0.335	0.234 \pm 0.236	0.489 \pm 0.203	0.057 \pm 0.151
C3	<u>0</u> \pm 0	0.057 \pm 0.151	0.143 \pm 0.378	0.271 \pm 0.256	<u>0</u> \pm 0
D1	0.200 \pm 0.252	0.200 \pm 0.252	0.233 \pm 0.321	0.363 \pm 0.207	<u>0.143</u> \pm 0.244
Overall	<u>0.203</u> \pm 0.361	0.265 \pm 0.377	0.294 \pm 0.405	0.221 \pm 0.271	0.261 \pm 0.410

Table C.16 – Average \pm standard deviation of normalized mutual information(NMI) between the spatial partition learned during the experiment, and the original spatial partition for the spatial attribute (*User.location*) of each model of Figure 8.6

Model	Baseline	Naive	Adaptative1	Adaptative2	Adaptative3
A1	<u>0.745</u> \pm 0.091	0.759 \pm 0.091	0.759 \pm 0.091	0.759 \pm 0.091	0.770 \pm 0.137
A2	<u>0.794</u> \pm 0.030	0.827 \pm 0.052	0.827 \pm 0.052	0.827 \pm 0.052	0.822 \pm 0.130
B1	<u>0.816</u> \pm 0.081	<u>0.816</u> \pm 0.081	<u>0.816</u> \pm 0.081	<u>0.816</u> \pm 0.081	0.827 \pm 0.147
C1	<u>0.833</u> \pm 0.059	0.836 \pm 0.057	0.836 \pm 0.057	0.836 \pm 0.057	0.882 \pm 0.108
C2	<u>0.801</u> \pm 0.044	0.807 \pm 0.041	0.807 \pm 0.041	0.807 \pm 0.041	0.895 \pm 0.116
C3	0.779 \pm 0.032	<u>0.769</u> \pm 0.034	<u>0.769</u> \pm 0.034	<u>0.769</u> \pm 0.034	0.862 \pm 0.086
D1	<u>0.829</u> \pm 0.011	<u>0.829</u> \pm 0.011	<u>0.829</u> \pm 0.011	<u>0.829</u> \pm 0.011	0.898 \pm 0.159
Overall	<u>0.800</u> \pm 0.061	0.806 \pm 0.062	0.806 \pm 0.062	0.806 \pm 0.062	0.851 \pm 0.128

Table C.17 – Absolute difference between Bayesian Dirichlet score of the gold models and that of the learned models.

Model	$ \Delta S(\text{Gold}, \text{Adap1}) $	$ \Delta S(\text{Gold}, \text{Adap2}) $	$ \Delta S(\text{Gold}, \text{Adap3}) $	$ \Delta S(\text{Gold}, \text{Naive}) $	$ \Delta S(\text{Gold}, \text{Baseline}) $
A1	3.27	4.69	<u>8.21</u>	3.54	3.54
A1	14.71	11.59	8.83	14.71	<u>15.47</u>
A1	<u>33.25</u>	15.60	19.34	<u>33.25</u>	<u>33.25</u>
A1	<u>78.19</u>	76.10	22.17	<u>78.19</u>	61.07
A1	136.28	119.17	5.51	136.28	<u>137.08</u>
A1	<u>222.87</u>	219.70	105.33	<u>222.87</u>	200.24
A1	<u>323.33</u>	304.82	320.56	<u>323.33</u>	<u>323.33</u>
A2	16.52	14.71	<u>16.76</u>	16.52	15.39
A2	11.50	11.99	5.75	11.50	<u>13.48</u>
A2	19.08	17.89	9.67	27.55	<u>36.82</u>
A2	24.94	35.06	1.76	<u>36.84</u>	36.17
A2	78.22	78.65	<u>82.79</u>	<u>82.79</u>	81.76
A2	75.28	81.56	51.96	<u>82.57</u>	<u>82.57</u>
B1	<u>9.35</u>	3.40	3.03	<u>9.35</u>	<u>9.35</u>
B1	<u>32.87</u>	27.88	14.88	<u>32.87</u>	<u>32.87</u>
B1	23.88	20.08	19.51	<u>26.29</u>	<u>26.29</u>
B1	<u>75.15</u>	54.94	43.54	71.29	71.29
B1	<u>161.05</u>	140.33	30.82	161.05	<u>161.05</u>
B1	<u>246.48</u>	242.77	6.30	<u>246.48</u>	<u>246.48</u>
B1	345.57	340.52	8.81	<u>359.74</u>	<u>359.74</u>
C1	<u>20.44</u>	17.17	20.18	<u>20.44</u>	<u>20.44</u>
C1	16.06	13.20	19.97	19.97	19.97
C1	<u>39.91</u>	29.38	21.26	<u>39.91</u>	<u>39.91</u>
C1	<u>76.27</u>	73.60	<u>76.27</u>	<u>76.27</u>	<u>76.27</u>
C1	<u>96.56</u>	79.97	25.51	95.55	95.55
C1	<u>195.07</u>	177.76	113.17	191.27	155.69
C1	<u>272.07</u>	202.80	110.17	206.53	206.53
C2	<u>14.12</u>	13.80	11.33	13.80	12.93
C2	13.32	18.69	<u>16.77</u>	18.69	16.77
C2	30.78	27.06	23.33	<u>40.97</u>	<u>40.97</u>
C2	13.78	9.77	<u>61.19</u>	13.58	13.60
C2	7.69	11.08	<u>167.88</u>	11.08	11.08
C2	17.54	17.54	<u>266.75</u>	5.40	1.56
C2	4.39	11.99	<u>424.52</u>	11.99	11.99
C3	7.40	6.49	1.21	7.40	7.40
C3	19.06	12.28	2.38	<u>19.06</u>	18.29
C3	<u>39.80</u>	31.07	20.34	<u>39.80</u>	<u>39.80</u>
C3	62.70	62.70	17.09	62.70	<u>66.53</u>
C3	112.53	101.39	<u>122.98</u>	112.53	118.26
C3	127.18	127.18	124.31	127.18	<u>152.73</u>
C3	236.80	227.75	<u>539.97</u>	227.75	238.19
D1	<u>23.96</u>	22.89	13.84	<u>23.96</u>	<u>23.96</u>
D1	26.56	<u>29.34</u>	11.88	26.56	26.56
D1	<u>60.76</u>	57.24	13.45	<u>60.76</u>	<u>60.76</u>
D1	73.82	79.04	11.37	<u>85.29</u>	<u>85.29</u>
D1	80.35	104.45	93.15	<u>109.28</u>	<u>109.28</u>
D1	196.43	177.59	42.30	196.43	196.43
D1	<u>337.01</u>	308.73	79.54	329.60	329.60

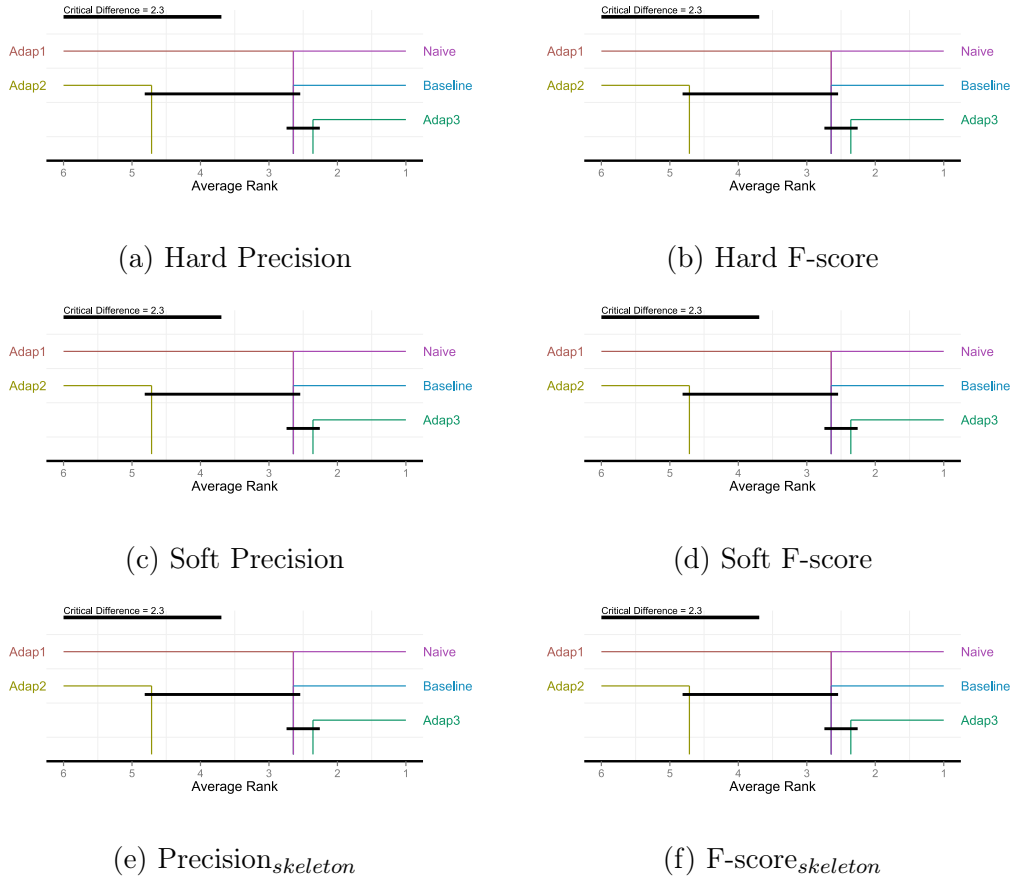


Figure C.1 – Comparison of PRM-SA learning algorithms with Nemenyi test for Model A1.

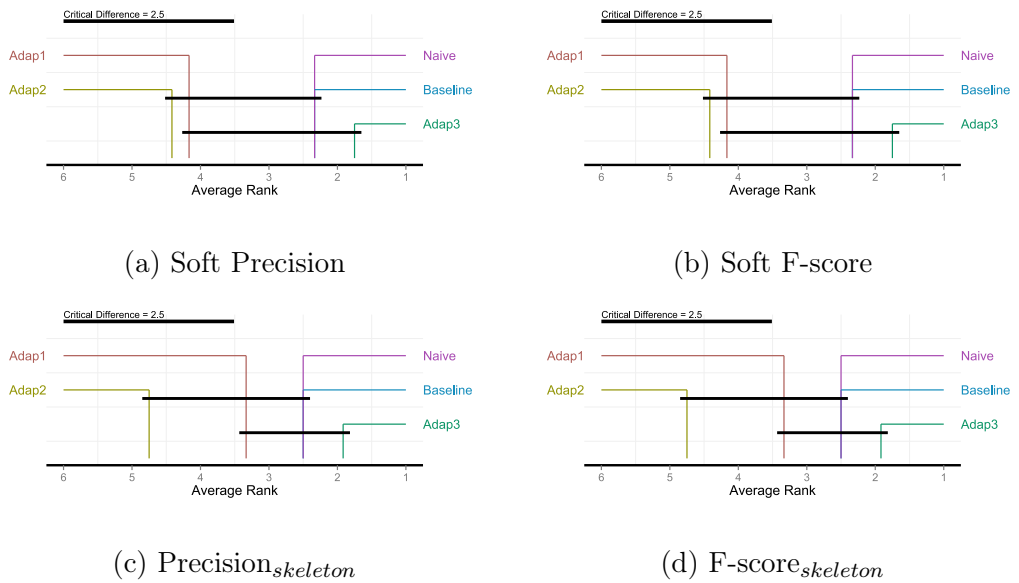
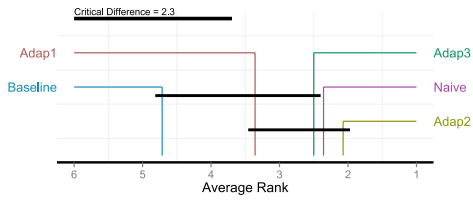
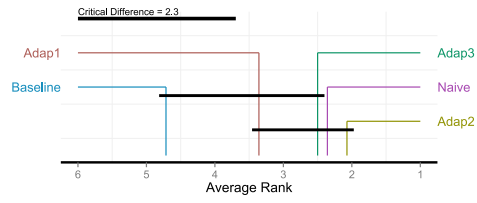


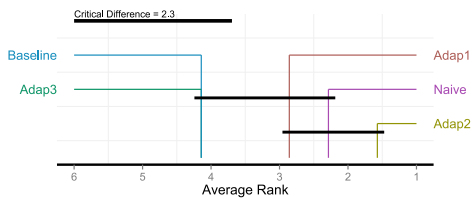
Figure C.2 – Comparison of PRM-SA learning algorithms with Nemenyi test for Model A2



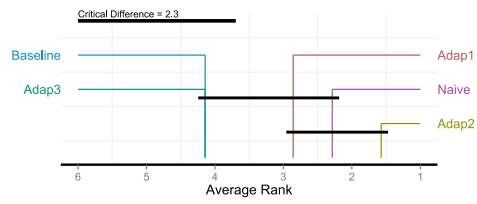
(a) Hard Precision



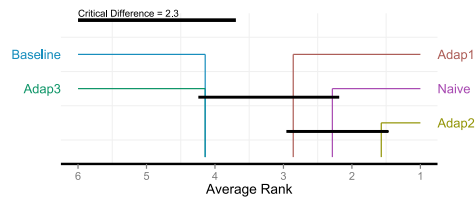
(b) Soft Precision



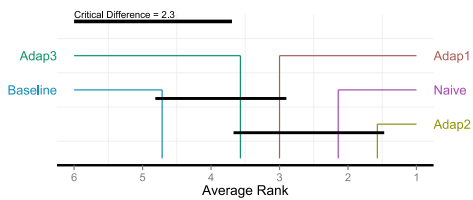
(c) Hard Recall



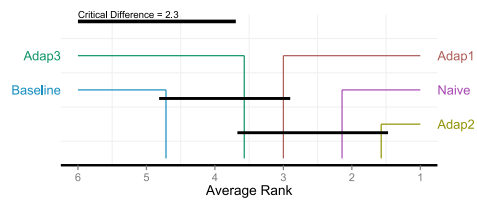
(d) Soft Recall



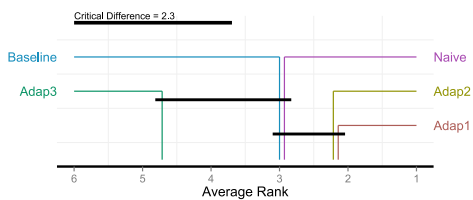
(e) Soft Recall_{spatial}



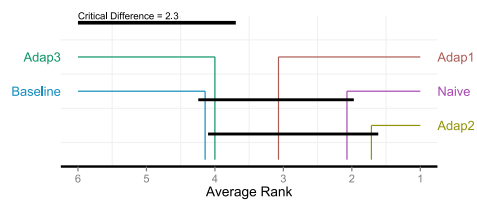
(f) Hard F-score



(g) Soft F-score



(h) F-score_{skeleton}



(i) Soft F-score_{spatial}

Figure C.3 – Comparison of PRM-SA learning algorithms with Nemenyi test for Model C2

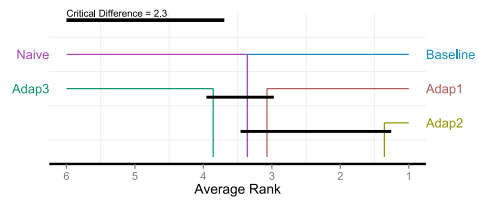
(a) $\text{Recall}_{\text{skeleton}}$

Figure C.4 – Comparison of PRM-SA learning algorithms with Nemenyi test for Model D1

Bibliography

- G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005. ISSN 1041-4347. [43](#), [45](#)
- G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, chapter 7, pages 217–253. Springer, 2011. [44](#)
- G. Adomavicius, N. Manouselis, and Y. Kwon. Multi-criteria recommender systems. In *Recommender systems handbook*, chapter 24, pages 769–803. Springer, 2011. [49](#)
- P. Aguilera, A. Fernández, R. Fernández, R. Rumí, and A. Salmerón. Bayesian networks in environmental modelling. *Environmental Modelling & Software*, 26(12):1376–1388, 2011. [80](#)
- A. Ahmed, L. Hong, and A. J. Smola. Hierarchical geographical modeling of user locations from social media posts. In *Proceedings of the 22nd international conference on World Wide Web*, pages 25–36. International World Wide Web Conferences Steering Committee, 2013. [84](#)
- H. J. Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008. [58](#)
- H. Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22(1):203–217, 1970. [19](#)
- C. F. Aliferis and I. Tsamardinos. Algorithms for large-scale local causal discovery and feature selection in the presence of limited sample or large causal neighbourhoods. Technical report, Technical report DSL-02-08, Department of Biomedical Informatics, Vanderbilt University, 2002. [20](#)
- S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck, and F. Jensen. Munin—an expert emg assistant. *Computer-aided electromyography and expert systems*, 21, 1989. [28](#)
- A. Azaria, A. Hassidim, S. Kraus, A. Eshkol, O. Weintraub, and I. Netanel. Movie recommender system for profit maximization. In Q. Yang, I. King, Q. Li, P. Pu, and G. Karypis, editors, *Seventh ACM Conference on Recommender Systems*, RecSys ’13, pages 121–128, Hong Kong, China, 2013. ACM. [55](#)
- R. Baeza-Yates, D. Jiang, F. Silvestri, and B. Harrison. Predicting the next app that you are going to use. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 285–294. ACM, 2015. [58](#)

- O. Bangsø and P. Wuillemin. Top-down construction and repetitive structures representation in bayesian networks. In J. N. Etheredge and B. Z. Manaris, editors, *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference, FLAIRS*, pages 282–286, Orlando, Florida, USA, 2000. AAAI Press. [3](#)
- J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12*, page 199, 2012. [84](#)
- J. Bao, Y. Zheng, D. Wilkie, and M. Mokbel. Recommendations in location-based social networks: a survey. *GeoInformatica*, 19(3):525–565, 2015. [80](#), [84](#)
- A. B. Barragáns-Martínez, E. Costa-Montenegro, J. C. Burguillo, M. Rey-López, F. A. Mikic-Fonte, and A. Peleteiro. A hybrid content-based and item-based collaborative filtering approach to recommend tv programs enhanced with singular value decomposition. *Information Sciences*, 180(22):4290–4311, 2010. [48](#)
- D. N. Barton, S. Kuikka, O. Varis, L. Uusitalo, H. J. Henriksen, M. Borsuk, A. de la Hera, R. Farmani, S. Johnson, and J. D. Linnell. Bayesian networks in environmental and resource management. *Integrated environmental assessment and management*, 8(3):418–429, 2012. [80](#)
- J. Beel, M. Genzmehr, S. Langer, A. Nürnberger, and B. Gipp. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In A. Bellogín, P. Castells, A. Said, and D. Tikk, editors, *Proceedings of the international workshop on reproducibility and replication in recommender systems evaluation*, RepSys 2013, pages 7–14, Hong Kong, China, 2013. ACM. [51](#)
- M. Ben Ishak. *Probabilistic relational models: learning and evaluation*. PhD thesis, Université de Nantes; Université de Tunis, Institut Supérieur de Gestion de Tunis, 2015. [ix](#), [xiii](#), [5](#), [6](#), [7](#), [26](#), [28](#), [29](#), [30](#), [31](#), [32](#), [37](#), [40](#), [117](#), [118](#), [119](#), [124](#), [133](#), [143](#), [147](#), [148](#), [153](#), [157](#), [170](#)
- M. Ben Ishak, N. Ben Amor, and P. Leray. A relational bayesian network-based recommender system architecture. In *Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO 2013)*, 2013. [iv](#), [ix](#), [61](#), [65](#), [66](#), [68](#), [69](#)
- M. Ben Ishak, R. Chulyadyo, A. Abdelwahab, M. Ramirez, P. Leray, and N. Ben Amor. Relational bayesian networks for recommender systems: review and comparative study. In *ENBIS-SFdS Spring Meeting on graphical causality models: Trees, Bayesian Networks and Big Data*, Paris, France, Apr. 2014. [7](#)
- M. Ben Ishak, R. Chulyadyo, and P. Leray. Probabilistic Relational Model Benchmark Generation. Technical report, LARODEC Laboratory, ISG, Université de Tunis, Tunisia ; DUKe research group, LINA Laboratory UMR 6241, University of Nantes, France ; DataForPeople, Nantes, France, Feb. 2016. [6](#)
- I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):1–36, 2007. [12](#)

- M. Bilgic and R. J. Mooney. Explaining recommendations: Satisfaction vs. promotion. In *Beyond Personalization Workshop, IUI*, volume 5, 2005. [153](#)
- D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *ICML*, volume 98, pages 46–54, 1998. [57](#)
- J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997. [28](#)
- J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. [12](#), [45](#)
- B. Bollobás, C. Borgs, J. Chayes, and O. Riordan. Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 132–139. Society for Industrial and Applied Mathematics, 2003. [33](#)
- R. R. Bouckaert. Probabilistic network construction using the minimum description length principle. In *Symbolic and quantitative approaches to reasoning and uncertainty*, pages 41–48. Springer, 1993. [19](#)
- L. Brozovsky and V. Petricek. Recommender system for online dating service. In *CoRR*, volume abs/cs/0703042, 2007. [56](#)
- A. Brun, A. Hamad, O. Buffet, and A. Boyer. Towards preference relations in recommender systems. In *Workshop on Preference Learning, European Conference on Machine Learning and Principle and Practice of Knowledge Discovery in Databases (ECML-PKDD 2010)*, 2010. [51](#)
- R. Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007. [ix](#), [48](#), [49](#)
- M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 328–334. American Association for Artificial Intelligence, 1999. [12](#)
- R. Cano, C. Sordo, and J. M. Gutiérrez. Applications of Bayesian networks in meteorology. *Advances in Bayesian networks*, pages 309–327, 2004. [106](#)
- I. Cantador, P. Brusilovsky, and T. Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM. [56](#)
- M. Chau and H. Chen. A machine learning approach to web page filtering using content and structure analysis. *Decision Support Systems*, 44(2):482–494, 2008. [12](#)
- Y. E. Chee, L. Wilkinson, A. E. Nicholson, P. F. Quintana-Ascencio, J. E. Fauth, D. Hall, K. J. Ponzio, and L. Rumpff. Modelling spatial and temporal changes with gis and spatial and dynamic bayesian networks. *Environmental Modelling & Software*, 82:108–120, 2016. [80](#)

- L. Chen and P. Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1-2):125–150, 2012. [50](#), [89](#)
- P. P.-S. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976. [15](#)
- R. Chulyadyo and P. Leray. Probabilistic Relational Models for Customer Preference Modelling and Recommendation. Technical report, Laboratoire d’Informatique de Nantes Atlantique - LINA, 2013. URL <http://hal.archives-ouvertes.fr/hal-00967044>. [7](#)
- R. Chulyadyo and P. Leray. A personalized recommender system from probabilistic relational model and users’ preferences. In *18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems*, KES 2014, pages 1063–1072, Gdynia, Poland, 2014. [6](#)
- R. Chulyadyo and P. Leray. Integrating spatial information into probabilistic relational models. In *IEEE International Conference on Data Science and Advanced Analytics*, DSAA’15, pages 1–8, Paris, France, Oct 2015. [6](#)
- E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. [13](#)
- G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. ISSN 0004-3702. [18](#)
- G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992. [19](#)
- A. Coutant. *Probabilistic Relational Models and Reference Uncertainty: Structure learning with clustering algorithms*. PhD thesis, Université de Nantes, Nov 2015. [24](#), [144](#)
- A. Coutant, P. Leray, and H. Le Capitaine. Probabilistic relational models with clustering uncertainty. In *International Joint Conference on Neural Networks*, IJCNN, pages 1–8. IEEE, 2015. [24](#)
- P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial intelligence*, 60(1):141–153, 1993. [18](#)
- R. Daly, Q. Shen, and S. Aitken. Review: learning bayesian networks: Approaches and issues. *The Knowledge Engineering Review*, 26(02):99–157, May 2011. ISSN 0269-8889. [16](#)
- L. M. De Campos, J. M. Fernández-Luna, J. F. Huete, and M. A. Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, 51(7):785–799, 2010. [57](#)
- M. de Jongh and M. J. Druzdzel. A comparison of structural distance measures for causal bayesian network models. *Recent Advances in Intelligent Information Systems, Challenging Problems of Science, Computer Science series*, pages 443–456, 2009. [29](#)

- L. De Raedt and K. Kersting. Probabilistic Inductive Logic Programming. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic Inductive Logic Programming – Theory and Applications*, chapter 1, pages 1–27. Springer, 2004. ISBN 978-3-540-78651-1. doi: 10.1007/978-3-540-78652-8. [13](#)
- V. Delcroix and A. Ben Mrad. Modéliser un critère par un réseau bayésien : V-structure et observations probabilistes fixes. In *Proceedings of the 8th Journées Francophones sur les Réseaux Bayésiens et les Modèles Graphiques Probabilistes*, JFRB 2016, pages 1–18, Clermont-Ferrand, France, 2016. [x](#), [98](#), [100](#), [101](#)
- J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. [127](#)
- S. Dooms, T. De Pessemier, and L. Martens. Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013*, 2013. [56](#)
- M. J. Egenhofer and J. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. *The*, 9:94–1, 1990. [76](#)
- M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011. [45](#), [46](#)
- S. Fakhraei, J. Foulds, M. Shashanka, and L. Getoor. Collective spammer detection in evolving multi-relational social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1769–1778. ACM, 2015. [153](#)
- I. Fernández-Tobías, I. Cantador, M. Kaminskas, and F. Ricci. Cross-domain recommender systems: A survey of the state of the art. In *Proceedings of the Second Spanish Conference on Information Retrieval (CERI 2012)*, 2012. [51](#)
- E. Fersini, E. Messina, and F. Archetti. Probabilistic relational models with relational uncertainty: An early study in web page classification. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, volume 3 of *WI-IAT'09*, pages 139–142. IET, 2009. [24](#)
- P. C. Fishburn. Additive utilities with incomplete product sets: Application to priorities and assignments. *Operations Research*, 15(3):537–542, 1967. [94](#)
- N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309. Lawrence Erlbaum Associates Ltd, 1999. [3](#), [12](#), [20](#), [26](#), [63](#), [112](#)
- Y. Gao, H. Qi, J. Liu, and D. Liu. A recommendation algorithm combining user grade-based collaborative filtering and probabilistic relational models. In *Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery - Volume 01*, FSKD '07, pages 67–71, Washington, DC, USA, 2007. IEEE Computer Society. [iv](#), [ix](#), [4](#), [48](#), [61](#), [65](#), [68](#), [69](#)

- M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010*, pages 257–260, Barcelona, Spain, 2010. ACM. [54](#)
- L. Getoor. *Learning statistical models from relational data*. PhD thesis, Stanford University, 2001. [3](#), [13](#), [21](#), [24](#), [26](#), [64](#), [67](#), [110](#)
- L. Getoor and M. Sahami. Using probabilistic relational models for collaborative filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, pages 1–6, 1999. [iv](#), [ix](#), [4](#), [61](#), [62](#), [63](#), [66](#), [68](#), [69](#)
- L. Getoor and B. Taskar. *Introduction to statistical relational learning*. MIT Press, 2007. [2](#), [12](#)
- L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, pages 170–177, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. [12](#), [20](#)
- L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar. Probabilistic relational models. In L. Getoor and B. Taskar, editors, *Introduction to statistical relational learning*, chapter 5, pages 129–174. The MIT press, 2007. [15](#), [21](#), [22](#), [28](#)
- K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001. [56](#)
- M. F. Goodchild. Twenty years of progress: GIScience in 2010. *Journal of Spatial Information Science*, 1(1):3–20, 2010. [106](#)
- D. A. Griffith. What is spatial autocorrelation? Reflections on the past 25 years of spatial statistics. *Espace géographique*, 21(3):265–280, 1992. [75](#), [106](#)
- G. Guo, J. Zhang, and D. Thalmann. Merging trust in collaborative filtering to alleviate data sparsity and cold start. *Knowledge-Based Systems*, 57:57–68, 2014. [57](#), [58](#)
- M. Gupta, R. Li, Z. Yin, and J. Han. Survey on social tagging techniques. *ACM Sigkdd Explorations Newsletter*, 12(1):58–72, 2010. [51](#)
- R. H. Güting. An introduction to spatial database systems. *The VLDB Journal – The International Journal on Very Large Data Bases*, 3(4):357–399, 1994. [76](#)
- J. Han, M. Kamber, and A. K. H. Tung. Spatial Clustering Methods in Data Mining: A Survey. In H. J. Miller and J. Han, editors, *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS*, pages 1–29. Taylor and Francis, 2001. [107](#)
- D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995. [29](#)
- D. Heckerman, C. Meek, and D. Koller. Probabilistic models for relational data. Technical report, MSR-TR-2004-30, Microsoft Research, 2004. [12](#)

- D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In L. Getoor and B. Taskar, editors, *Introduction to statistical relational learning*, chapter 7, pages 201–238. The MIT Press, 2007. [13](#)
- M. Henrion. Some practical issues in constructing belief networks. In *Proceedings of the Third Annual Conference on Uncertainty in Artificial Intelligence*, UAI'87, pages 161–174, Seattle, WA, USA, 1987. [94](#)
- M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In J. F. Lemmer and L. N. Kanal, editors, *Uncertainty in Artificial Intelligence*, volume 5 of *Machine Intelligence and Pattern Recognition*, pages 149 – 163. North-Holland, 1988. [18](#), [31](#), [37](#)
- J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004. [51](#), [52](#), [54](#), [55](#)
- T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 688–693, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. [62](#)
- W. Hong, S. Zheng, H. Wang, and J. Shi. A job recommender system based on user clustering. *Journal of Computers*, 8(8):1960–1967, 2013. [42](#), [51](#)
- R. Hu and P. Pu. A study on user perception of personality-based recommender systems. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 291–302. Springer, 2010. [50](#)
- J. Huang and Y. Yuan. Construction and Application of Bayesian Network Model for Spatial Data Mining. In *IEEE International Conference on Control and Automation, ICCA 2007*, pages 2802–2805, 2007. [106](#)
- S.-L. Huang. Designing utility-based recommender systems for e-commerce: Evaluation of preference-elicitation methods. *Electronic Commerce Research and Applications*, 10(4):398–407, 2011. [50](#)
- Y. Huang and L. Bian. A Bayesian network and analytic hierarchy process based personalized recommendations for tourist attractions over the Internet, 2009. [81](#), [83](#), [84](#)
- Z. Huang, D. Zeng, and H. Chen. A unified recommendation framework based on probabilistic relational models. In *Fourteenth Annual Workshop on Information Technologies and Systems (WITS)*, pages 8–13, 2004. [iv](#), [4](#), [7](#), [24](#), [61](#), [63](#), [66](#), [68](#), [69](#), [136](#), [148](#), [149](#), [150](#), [154](#), [179](#), [180](#)
- Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 141–142. ACM, 2005. [12](#)
- M. Jaeger. Relational bayesian networks. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 266–273. Morgan Kaufmann Publishers Inc., 1997. [3](#), [12](#)

- M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 397–406, Paris, France, 2009. ACM. 57
- D. Jannach and G. Friedrich. Tutorial: recommender systems. International Joint Conference on Artificial Intelligence, 2013. URL http://ijcai13.org/files/tutorial_slides/td3.pdf. ix, 48, 49
- D. Jannach, M. Zanker, M. Ge, and M. Gröning. Recommender systems in computer science and information systems – a landscape of research. In *Proceedings of the 13th International Conference on Electronic Commerce and Web Technologies*, volume 123 of *EC-Web 2012*, page 76, Vienna, Austria, 2012. Springer. 42, 52
- C. S. Jensen, U. Kjærulff, and A. Kong. Blocking Gibbs sampling in very large probabilistic expert systems. *International Journal of Human-Computer Studies*, 42(6): 647–666, 1995. 18
- F. V. Jensen, K. G. Olesen, and S. K. Andersen. An algebra of bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659, 1990. 18
- N. Jones, A. Brun, and A. Boyer. Comparisons instead of ratings: Towards more stable preferences. In *Proceedings of the 2011 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2011*, WI 2011, pages 451–456, Campus Scientifique de la Doua, Lyon, France, 2011. IEEE Computer Society. 51
- F. Kaelin. Approximate Inference in Probabilistic Relational Models. Technical report, McGill University, Montreal, Canada, 2011. xiii, 25, 39, 40, 118, 122, 153, 154
- R. Katarya and O. P. Verma. Recent developments in affective recommender systems. *Physica A: Statistical Mechanics and its Applications*, 461:182 – 190, 2016. 50
- K. Kersting and L. De Raedt. Bayesian logic programming: Theory and tool. In L. Getoor and B. Taskar, editors, *Introduction to statistical relational learning*, chapter 10, pages 291–321. The MIT Press, 2007. 12
- J. H. Kim and J. Pearl. A computational model for causal and diagnostic reasoning in inference systems. In A. Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*, IJCAI 83, pages 190–193, Karlsruhe, FRG, 1983. William Kaufmann. 18
- J. Kisynski and D. Poole. Lifted aggregation in directed first-order probabilistic models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI 2009, pages 1922–1929, Pasadena, California, USA, 2009. 25
- U. B. Kjærulff and A. L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer Science & Business Media, 2007. 92
- D. Koller and A. Pfeffer. Object-oriented bayesian networks. In D. Geiger and P. P. Shenoy, editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI '97, pages 302–313, Brown University, Providence, Rhode Island, USA, 1997. Morgan Kaufmann. 3

- Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009. [57](#)
- P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 99–106. ACM, 2015. [153](#)
- K. Kristensen and I. A. Rasmussen. The use of a bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33(3):197–217, 2002. [28](#)
- D. Landuyt, S. Broekx, R. D’hondt, G. Engelen, J. Aertsens, and P. L. Goethals. A review of bayesian belief networks in ecosystem service modelling. *Environmental Modelling & Software*, 46:1–11, 2013. [80](#)
- S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988. [18](#)
- S. Lee and V. Honavar. On learning causal models from relational data. In D. Schuurmans and M. P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3263–3270, Phoenix, Arizona, USA., 2016. AAAI Press. [154](#)
- S. K. Lee, Y. H. Cho, and S. H. Kim. Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. *Information Sciences*, 180(11): 2142–2155, 2010. [58](#)
- J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. LARS: A Location-Aware Recommender System. *2012 IEEE 28th International Conference on Data Engineering*, 1:450–461, apr 2012. [ix](#), [80](#), [81](#), [83](#), [84](#)
- L. Li, J. Wang, H. Leung, and C. Jiang. Assessment of catastrophic risk using bayesian network constructed from domain knowledge and spatial data. *Risk Analysis*, 30(7): 1157–1175, 2010. [106](#)
- B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073, 2014. [57](#), [58](#)
- N. N. Liu, M. Zhao, and Q. Yang. Probabilistic latent preference analysis for collaborative filtering. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009*, pages 759–766, Hong Kong, China, 2009. [51](#)
- P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, chapter 3, pages 73–105. Springer, 2011. [46](#)
- P. Lops, M. De Gemmis, G. Semeraro, C. Musto, and F. Narducci. Content-based and collaborative techniques for tag recommendation: an empirical evaluation. *Journal of Intelligent Information Systems*, 40(1):41–61, 2013. [51](#)

- H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM 2008, pages 931–940, Napa Valley, California, USA, 2008. 57
- S. A. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983, 2007. 12
- M. Maier, K. Marazopoulou, D. Arbour, and D. Jensen. A sound and complete algorithm for learning causal models from relational data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI 2013, pages 371–380, Bellevue, WA, USA, 2013. 29, 133, 154
- D. Malerba. A relational perspective on spatial data mining. *International Journal of Data Mining, Modelling and Management*, 1(1):103–118, 2008. 4, 73, 75, 106
- L. B. Marinho, T. Sandholm, C. de Souza Baptista, I. Nunes, C. Nóbrega, and J. Araújo. Extracting geospatial preferences using relational neighbors. *Journal of Information and Data Management*, 3(3):364–377, 2012. 81, 83, 84
- T. Martin, B. Ball, B. Karrer, and M. E. J. Newman. Coauthorship and citation in scientific publishing. *CoRR*, abs/1304.0473, 2013. 12
- J. Masthoff. Group recommender systems: Combining individual models. In *Recommender systems handbook*, chapter 21, pages 677–702. Springer, 2011. 51
- S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *Extended Abstracts Proceedings of the 2006 Conference on Human Factors in Computing Systems*, CHI 2006, pages 1097–1101, Montréal, Québec, Canada, 2006. ACM. 54
- P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 187–192, Edmonton, Alberta, Canada, 2002. AAAI Press / The MIT Press. 48, 57
- C. Mettouris and G. A. Papadopoulos. Ubiquitous recommender systems. *Computing*, 96(3):223–257, 2014. 51
- F. Meyer, F. Fessant, F. Clérot, and É. Gaussier. Toward a new protocol to evaluate recommender systems. In *Proceedings of the Workshop on Recommendation Utility Evaluation: Beyond RMSE*, RUE 2012, pages 9–14, Dublin, Ireland, 2012. 51
- B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, and L. P. Kaelbling. Lifted probabilistic inference with counting formulas. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, AAAI 2008, pages 1062–1068, Chicago, Illinois, USA, 2008. AAAI Press. 25
- B. Mobasher, R. Burke, and J. J. Sandvig. Model-based collaborative filtering as a defense against profile injection attacks. In *Proceedings of The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 1388–1393, Boston, Massachusetts, USA, 2006. AAAI Press. 57

- P. Närman, M. Buschle, J. König, and P. Johnson. Hybrid probabilistic relational models for system quality analysis. In *14th IEEE International on Enterprise Distributed Object Computing Conference*, EDOC, pages 57–66. IEEE, 2010. [24](#)
- J. Neville and D. Jensen. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, pages 77–91. Citeseer, 2003. [12](#), [112](#)
- J. Neville and D. D. Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, 2007. [3](#)
- M. E. J. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003. [32](#)
- J. Newton and R. Greiner. Hierarchical probabilistic relational models for collaborative filtering. In *Workshop on Statistical Relational Learning, 21st International Conference on Machine Learning*, 2004. [iv](#), [ix](#), [4](#), [24](#), [61](#), [64](#), [65](#), [67](#), [68](#), [69](#)
- M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning*, ICML-11, pages 809–816, 2011. [12](#)
- M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016. [13](#)
- M. Nilashi, O. bin Ibrahim, and N. Ithnin. Hybrid recommendation approaches for multi-criteria collaborative filtering. *Expert Systems with Applications*, 41(8):3879–3900, 2014. [49](#)
- M. Papagelis, D. Plexousakis, and T. Kutsuras. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *Trust management*, pages 224–239. Springer, 2005. [57](#)
- D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim. A literature review and classification of recommender systems research. *Expert Systems with Applications*, 39(11):10059–10072, 2012. [42](#)
- M.-h. Park, J.-h. Hong, and S.-b. Cho. Location-Based Recommendation System Using Bayesian User’s Preference Model in Mobile Devices. *Ubiquitous Intelligence and Computing*, pages 1130–1139, 2007. [106](#)
- M. J. Pazzani and D. Billsus. Content-based recommendation systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72078-2. [46](#)
- J. Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982. [18](#)
- J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 1558604790. [ix](#), [16](#), [18](#), [94](#)

- C. Perlich and Z. Huang. Relational learning for customer relationship management. In *Proceedings of international workshop on customer relationship management: data mining meets marketing*, 2005. [64](#)
- A. J. Pfeffer. *Probabilistic reasoning for complex systems*. PhD thesis, Stanford University, 2000. [25](#)
- J. Pitman. Combinatorial stochastic processes. *Lecture Notes for St. Flour Summer School*, 2002. [33](#)
- L. Pizzato, T. Rej, T. Chung, I. Koprinska, and J. Kay. Recon: a reciprocal recommender for online dating. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 207–214. ACM, 2010. [42](#), [51](#)
- A. Popescul and L. H. Ungar. Statistical relational learning for link prediction. In *IJCAI workshop on learning statistical models from relational data*, pages 81–90, New York, 2003. ACM Press. [12](#)
- P. Pu, L. Chen, and R. Hu. Evaluating recommender systems from the user’s perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4-5):317–355, 2012. [48](#), [49](#), [89](#)
- L. D. Raedt. *Logical and relational learning*. Cognitive Technologies. Springer, 2008. ISBN 978-3-540-20040-6. [2](#)
- F. Ricci. Recommender systems: Models and techniques. In *Encyclopedia of Social Network Analysis and Mining*, pages 1511–1522. Springer, 2014. [44](#)
- F. Ricci, L. Rokach, B. Shapira, and B. P. Kantor. *Recommender Systems Handbook*. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. [12](#), [45](#)
- F. Ricci, L. Rokach, and B. Shapira. Recommender systems: introduction and challenges. In *Recommender Systems Handbook*, chapter 1, pages 1–34. Springer, 2015. [49](#), [51](#), [58](#), [88](#)
- M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006. ISSN 1573-0565. [12](#), [13](#)
- R. Rossi and J. Neville. Modeling the evolution of discussion topics and communication to improve relational classification. In *Proceedings of the First Workshop on Social Media Analytics*, pages 89–97. ACM, 2010. [12](#)
- R. A. Rossi, L. K. McDowell, D. W. Aha, and J. Neville. Transforming graph data for statistical relational learning. *Journal of Artificial Intelligence Research*, 45(1):363–441, 2012. [13](#)
- S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952. [18](#)
- T. L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980. [95](#)
- T. L. Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008. [vii](#), [95](#)

- A. Said and A. Bellogín. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 129–136. ACM, 2014a. [51](#)
- A. Said and A. Bellogín. Rival: a toolkit to foster reproducibility in recommender system evaluation. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 371–372. ACM, 2014b. [57](#)
- L. Salwinski, C. S. Miller, A. J. Smith, F. K. Pettit, J. U. Bowie, and D. Eisenberg. The database of interacting proteins: 2004 update. *Nucleic acids research*, 32(suppl 1):D449–D451, 2004. [12](#)
- T. Sang, P. Beame, and H. A. Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005. [18](#)
- B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system – a case study. Technical report, DTIC Document, 2000. [57](#)
- B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. [46](#)
- M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Lars*: An efficient and scalable location-aware recommender system. *Knowledge and Data Engineering, IEEE Transactions on*, 26(6):1384–1399, 2014. [80](#), [83](#), [84](#), [132](#)
- A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002. [58](#)
- G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978. [19](#)
- S. Sen, J. Vig, and J. Riedl. Tagommenders: connecting users to items through tags. In *Proceedings of the 18th international conference on World wide web*, pages 671–680. ACM, 2009. [51](#)
- R. D. Shachter. Evaluating influence diagrams. *Operations research*, 34(6):871–882, 1986. [18](#)
- G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2(1-4):327–351, 1990. [18](#)
- G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and B. P. Kantor, editors, *Recommender systems handbook*, chapter 8, pages 257–297. Springer, 2011. [51](#), [55](#)
- S. Shearin and H. Lieberman. Intelligent profiling by example. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 145–151. ACM, 2001. [5](#), [88](#)

- S. Shekhar, M. R. Evans, J. M. Kang, and P. Mohan. Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):193–214, May 2011. [73](#), [75](#)
- N. Shibata, Y. Kajikawa, and I. Sakata. Link prediction in citation networks. *Journal of the American society for information science and technology*, 63(1):78–85, 2012. [12](#)
- P. Singla and P. M. Domingos. Lifted first-order belief propagation. In *AAAI*, volume 8, pages 1094–1099, 2008. [25](#)
- B. Smyth. Case-based recommendation. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, pages 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72079-9. [49](#), [88](#)
- B. Smyth and P. Cotter. A personalised tv listings service for the digital tv age. *Knowledge-Based Systems*, 13(2):53–59, 2000. [48](#)
- P. Spirtes, C. N. Glymour, and R. Scheines. *Causation, prediction, and search*. MIT press, 2nd edition, 2000. [19](#), [29](#), [133](#), [154](#)
- X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:1–20, 2009. [46](#)
- L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826. ACM, 2009. [12](#)
- B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In A. Darwiche and N. Friedman, editors, *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence*, UAI '02, pages 485–492, University of Alberta, Edmonton, Alberta, Canada, 2002. Morgan Kaufmann. [3](#)
- B. Taskar, P. Abbeel, M.-F. Wong, and D. Koller. Relational markov networks. In L. Getoor and B. Taskar, editors, *Introduction to statistical relational learning*, chapter 6, pages 175–199. The MIT press, 2007. [12](#)
- W. R. Tobler. A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography*, 46:234–240, 1970. [75](#)
- E. Triantaphyllou and S. H. Mann. An examination of the effectiveness of multi-dimensional decision-making methods: A decision-making paradox. *Decision Support Systems*, 5(3):303–312, Sept. 1989. ISSN 0167-9236. [94](#)
- I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale markov blanket discovery. In *Proceedings of the sixteenth international Florida artificial intelligence research society conference*, pages 376–381, 2003. [26](#)
- I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006. [20](#), [29](#)
- L. Ungar and D. P. Foster. A formal statistical approach to collaborative filtering. *CONALD'98*, pages 1–6, 1998. [62](#)

- S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the 5th ACM conference on Recommender systems*, pages 109–116. ACM, 2011. [58](#)
- B. Vargas-Govea, G. González-Serna, and R. Ponce-Medellín. Effects of relevant contextual features in the performance of a restaurant recommender system. *ACM RecSys*, 11:592, 2011. [56](#)
- D. Véras, T. Prota, A. Bispo, R. Prudêncio, and C. Ferraz. A literature review of recommender systems in the television domain. *Expert Systems with Applications*, 42(22):9046–9076, 2015. [42](#)
- T. S. Verma and J. Pearl. Equivalence and synthesis of causal models. In *Uncertainty in Artificial Intelligence 6*, pages 255–268. North-Holland, 1991. [19](#)
- P. Viappiani, B. Faltings, and P. Pu. The lookahead principle for preference elicitation: Experimental results. In H. L. Larsen, G. Pasi, D. O. Arroyo, T. Andreassen, and H. Christiansen, editors, *Proceedings of the 7th International Conference on Flexible Query Answering Systems*, volume 4027 of *FQAS*, pages 378–389, Milan, Italy, 2006. Springer, Springer. [88](#)
- P. Viappiani, P. Pu, and B. Faltings. Conversational recommenders with adaptive suggestions. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 89–96. ACM, 2007. [5](#), [88](#)
- M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008. [3](#), [12](#)
- A. R. Walker, B. Pham, and M. Moody. Spatial bayesian learning algorithms for geographic information retrieval. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 105–114. ACM, 2005. [84](#), [106](#)
- C. Wallace, K. B. Korb, and H. Dai. Causal discovery via mml. In *ICML*, volume 96, pages 516–524. Citeseer, 1996. [19](#)
- H. Wang, M. Terrovitis, and N. Mamoulis. Location Recommendation in Location-based Social Networks using User Check-in Data. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 374–383. ACM, Springer, 2013. [ix](#), [82](#), [83](#)
- Y. Weiss. Belief propagation and revision in networks with loops. Technical report, 1616, MIT AI lab, 1997. [18](#)
- L. Wilkinson, Y. E. Chee, A. E. Nicholson, and P. F. Quintana-Ascencio. An Object-oriented Spatial and Temporal Bayesian Network for Managing Willows in an American Heritage River Catchment. In *UAI Application Workshops*, pages 77–86, 2013. [106](#)
- P.-H. Wuillemin and L. Torti. Structured probabilistic inference. *International Journal of Approximate Reasoning*, 53(7):946–968, 2012. [25](#)

- M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. *Proc. of the 34th international ACM SIGIR conference on Research and development in Information*, page 325, 2011. [48](#), [81](#), [82](#), [83](#), [84](#)
- Z. Zhang, X. Zheng, and D. D. Zeng. A framework for diversifying recommendation lists by user interest expansion. *Knowledge-Based Systems*, 105:83–95, 2016. [58](#)
- C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005. [54](#), [56](#)

Thèse de Doctorat

Rajani CHULYADYO

Un nouvel horizon pour la recommandation

Intégration de la dimension spatiale dans l'aide à la décision

A new horizon for the recommendation

Integration of spatial dimensions to aid decision making

Résumé

De nos jours, il est très fréquent de représenter un système en termes de relations entre objets. Parmi les applications les plus courantes de telles données relationnelles, se situent les systèmes de recommandation (RS), qui traitent généralement des relations entre utilisateurs et items à recommander. Les modèles relationnels probabilistes (PRM) sont un bon choix pour la modélisation des dépendances probabilistes entre ces objets. Une tendance croissante dans les systèmes de recommandation est de rajouter une dimension spatiale à ces objets, que ce soient les utilisateurs, ou les items. Cette thèse porte sur l'intersection peu explorée de trois domaines connexes - modèles probabilistes relationnels (et comment apprendre les dépendances probabilistes entre attributs d'une base de données relationnelles), les données spatiales et les systèmes de recommandation. La première contribution de cette thèse porte sur le chevauchement des PRM et des systèmes de recommandation. Nous avons proposé un modèle de recommandation à base de PRM capable de faire des recommandations à partir des requêtes des utilisateurs, mais sans profils d'utilisateurs, traitant ainsi le problème du démarrage à froid. Notre deuxième contribution aborde le problème de l'intégration de l'information spatiale dans un PRM.

Mots clés

Apprentissage Relationnel, Modèles Relationnels Probabilistes, Systèmes de Recommandation, Données Spatiales, Préférences des utilisateurs.

Abstract

Nowadays it is very common to represent a system in terms of relationships between objects. One of the common applications of such relational data is Recommender System (RS), which usually deals with the relationships between users and items. Probabilistic Relational Models (PRMs) can be a good choice for modeling probabilistic dependencies between such objects. A growing trend in recommender systems is to add spatial dimensions to these objects, and make recommendations considering the location of users and/or items. This thesis deals with the (not much explored) intersection of three related fields – Probabilistic Relational Models (a method to learn probabilistic models from relational data), spatial data (often used in relational settings), and recommender systems (which deal with relational data). The first contribution of this thesis deals with the overlapping of PRM and recommender systems. We have proposed a PRM-based personalized recommender system that is capable of making recommendations from user queries in cold-start systems without user profiles. Our second contribution addresses the problem of integrating spatial information into a PRM.

Key Words

Relational Learning, Probabilistic Relational Models, Recommender Systems, Spatial data, User Preferences.

