Mach Learn (2012) 89:233–255
DOI 10.1007/s10994-012-5309-4

Relational networks of conditional preferences

Frédéric Koriche

Received: 30 January 2012 / Revised: 6 May 2012 / Accepted: 14 June 2012 /
Published online: 18 July 2012

Abstract Much like relational probabilistic models, the need for relational preference models naturally arises in real-world applications involving multiple, heterogeneous, and richly interconnected objects. On the one hand, relational preferences should be represented into statements which are natural for human users to express. On the other hand, relational preference models should be endowed with a structure that supports tractable forms of reasoning and learning. Based on these criteria, this paper introduces the framework of *relational conditional preference networks (RCP-nets)*, that maintains the spirit of the popular "CP-nets" by expressing relational preferences in a natural way using the *ceteris paribus* semantics. We show that *acyclic* RCP-nets support tractable inference for optimization and ranking tasks. In addition, we show that in the online learning model, *tree-structured* RCP-nets (with bipartite orderings) are efficiently learnable from both optimization tasks and ranking tasks, using linear loss functions. Our results are corroborated by experiments on a large-scale movie recommendation dataset.

Keywords Conditional preferences · Relational networks · Preference optimization · Preference ranking · Online learning

1 Introduction

A recurrent issue in AI is the development of rational agents capable of tailoring their actions and recommendations to the preferences of human users. The spectrum of applications that depend on this ability is extremely wide, ranging from adaptive interfaces and configuration software, to recommender systems and group decision-making (Brafman and Domshlak 2009). In a nutshell, the crucial ingredients for addressing this issue are *representation*, *reasoning* and *learning*. In complex domains, we need a representation that offers a compact encoding of preference relations defined over large outcome spaces. We also need to be

Editors: S. Muggleton and J. Chen.

F. Koriche (✉)
LIRMM, CNRS UMR 5506, Université Montpellier II, Montpellier Cedex 5, France
e-mail: Frederic.Koriche@lirmm.fr

able to use this representation effectively in order to answer a broad range of queries. And, since the performance of decision makers is dependent on their aptitude to reflect users' preferences, we need to be able to predict and extract such preferences in an automatic way.

Among the different preference models that have been devised in the literature, *conditional preference networks (CP-nets)* have attracted a lot of attention by providing a compact and intuitive representation of qualitative preferences (Boutilier et al. 2004b). By analogy with Bayesian networks, CP-nets are graphical models in which nodes describe variables of interest and edges capture preferential dependencies between variables. Each node is labeled with a table expressing the preference over alternative values of the node given different values of the parent nodes under a *ceteris paribus* ("all else being equal") assumption. For example, in a CP-net for movie recommendation, the rule:

$$\mathsf{Genre} : \mathsf{comedy} \succ \mathsf{drama} \mid \mathsf{Date} = \mathsf{fifties}$$

might state that, for a film released in the fifties I prefer a comedy to a drama, provided that all other properties are the same. The semantics of a CP-net is a preference ordering on outcomes derived from such reading of entries in the conditional preference tables.

Despite their popularity, CP-nets are intrinsically limited to "attribute-value" domains. Many applications, however, are richly structured, involving objects of multiple types that are related to each other through a network of different types of relations. Such applications pose new challenges for devising relational preference models endowed with expressive representations, efficient inference engines, and fast learning algorithms.

In this paper, we introduce the framework of *relational* conditional preference networks (RCP-nets) that extends ceteris paribus preferences to relational domains. Briefly, an RCP-net is a template over a relational schema which specifies a ground CP-net for each particular set of objects. Based on the ceteris paribus paradigm, the representations provided by RCP-nets are *transparent*, in that a human expert can easily capture their meaning. For example, in an RCP-net for movie recommendation, the entry:

$$\mathsf{Movie.Genre} : \mathsf{action} \succ \mathsf{drama} \mid \mathit{mode}(\mathsf{Movie.Audience.User.Age}) = \mathsf{teen}$$

might capture the stereotype that, all other things being equal, action movies are preferable to dramas if the majority of people in the audience are teenagers.

In essence, the interest of RCP-nets lies in their ability to compare and order *relational outcomes*. Semantically, a relational outcome consists in a set of objects interconnected by the functional dependencies of the database schema. Two outcomes are "comparable" if they are defined over the same set of objects, but differ in the values assigned to the attributes of objects. For example, in configuration software (Junker 2006), the overall goal of the decision maker is to assemble from an available catalog of generic objects a customized product that meets user's preferences. Customized products, such as computers, cars, insurance products and travel packages, can be described as relational outcomes. In recommender systems (Jannach et al. 2010), a fundamental task is to predict what degree of desire a user would give to a new, unrated, item. Modeling the interaction of users and items as relational outcomes can help the system in making a better personalized recommendation by incorporating relational information about the user, such as her community in social networks (Golbeck 2006), or about the item, such as the actors, directors and critics of a movie (Melville et al. 2002; Newton and Greiner 2004).

From a computational viewpoint, a key feature of our framework is that the class of *acyclic* RCP-nets supports efficient inference for two well-studied inference tasks in preference handling: *outcome optimization* and *outcome ranking*. In outcome optimization, the decision maker is given a partial outcome in which some object attributes are left unspecified; the task is to find a maximally preferred completion of this outcome. For an acyclic CPR-net, (unconstrained) outcome optimization can be solved in polynomial time using a greedy algorithm that finds a maximally preferred completion of a partial outcome according to some topological order induced by the preference network. In outcome ranking, the decision maker is given a set of outcomes defined over the same collection of objects, but which differ in the values assigned to the object attributes; the task is to rank these outcomes in some non-decreasing order of preference. Again, such a task can be solved in polynomial time for acyclic RCP-nets, by compiling the network into a utility function that assigns a score to each outcome under consideration.

The learnability of RCP-nets is analyzed within the *online learning* setting (Cesa-Bianchi and Lugosi 2006), a well-studied theoretical model for devising algorithms capable of making and updating recommendations in real-time. In this setting, the decision maker observes instances of a reasoning task in a sequential manner. On round t, after observing the tth instance, the decision maker attempts to predict the solution associated with this instance. The prediction is formed by a hypothesis chosen from a predefined class $\mathcal{N}$ of RCP-nets. The decision maker can use this information to choose another hypothesis from the class $\mathcal{N}$ before proceeding to the next round. As a common thread in online learning, we make no assumption regarding the sequence of instance-solution pairs. This setting is thus general enough to capture agnostic situations in which the "true" preference model is not necessarily an element of the predefined class $\mathcal{N}$.

To measure the performance of the decision maker, we consider two standard metrics. The first, called *regret*, measures the difference in cumulative loss between the decision maker and the optimal hypothesis in $\mathcal{N}$. The second metric is computational complexity, i.e. the amount of computer resources required to choose hypotheses and to predict solutions. Based on these metrics, we show that the class of tree-structured RCP-nets (with bipartite orderings) is *efficiently learnable* from both optimization tasks and ranking tasks, using linear loss functions. Our online learning algorithm is an extension of the *Hedge* algorithm (Freund and Schapire 1997) that exploits the Matrix-Tree Theorem (Tutte 1984) for generating directed spanning trees at random.

The paper is organized as follows. After introducing the necessary background in graph theory (Sect. 2), we examine the syntax and semantics of RCP-nets in Sect. 3. The theoretical results concerning reasoning with acyclic RCP-nets and learning with tree-structured RCP-nets are presented in Sects. 4 and 5, respectively. In Sect. 6 we illustrate the learning potential of our framework with experiments on a large dataset. In Sect. 7, we compare our framework with related work and, in Sect. 8, we conclude by mentioning some perspectives of further research.

2 Preliminaries

Before delving into the representation of relational preference networks, we review the basic concepts from graph theory used in this paper. A *digraph* is a pair $\mathcal{G} = (\mathcal{X}, \mathcal{E})$, where $\mathcal{X}$ is a nonempty, finite set, and $\mathcal{E}$ is a binary relation on $\mathcal{X}$. The elements of $\mathcal{X}$ are the *nodes* of $\mathcal{G}$, and the elements of $\mathcal{E}$ are the *(directed) edges* of $\mathcal{G}$. The *size* of $\mathcal{G}$, denoted $|\mathcal{G}|$, is given by the number of its edges. Undirected graphs are represented here as digraphs for which the binary relation on nodes is symmetric. Notably, the *underlying graph* of a digraph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ is the pair formed by $\mathcal{X}$ and the symmetric closure of $\mathcal{E}$.

For a digraph $\mathcal{G}$ and a pair of nodes X, Y, a *walk* of length k in $\mathcal{G}$ from X to Y is a sequence of nodes $(X_1, \ldots, X_{k+1})$ such that $X = X_1$, $Y = X_{k+1}$ and (X_i, X_{i+1}) is an edge in $\mathcal{G}$ for $1 \leq i \leq k$. The walk is a *path* if all nodes are distinct, and a *cycle* if $X_1 = X_{k+1}$ and all intermediate nodes are distinct. For any pair of nodes X, Y, if there is a path of length k from X to Y then Y is an *ancestor* of X, and X is a *descendant* of Y. In the specific case where $k = 1$, Y is called a *parent* of X, and X is called a *child* of Y. A *root* (or *source*) is a node with no parents, and dually, a *leaf* (or *sink*) is a node with no children. The *in-degree* (resp. *out-degree*) of a digraph $\mathcal{G}$ is the maximum number of parents (resp. children) per node in $\mathcal{G}$.

The *deletion* of an edge (X, Y) from a digraph $\mathcal{G}$ is the digraph obtained by simply removing (X, Y) from $\mathcal{G}$. The *contraction* of (X, Y) in $\mathcal{G}$ is the digraph obtained by merging (X, Y) with a new node Z and redefining any edge (X, X') (resp. (Y', Y)) to (Z, X') (resp. (Y', Z)).

A digraph is *acyclic* if it contains no cycles. An acyclic digraph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ is *complete bipartite* if $\mathcal{X}$ can be partitioned into two sets $\mathcal{X}_1$ and $\mathcal{X}_2$ such that $\mathcal{E} = \mathcal{X}_1 \times \mathcal{X}_2$. In the particular case where $|\mathcal{X}_1| = 1$, $\mathcal{G}$ is a *star*. A *forest* is an acyclic digraph of in-degree one, and a *tree* is a forest with exactly one root node. A *spanning tree* (resp. *spanning forest*) of a digraph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ is a tree (resp. forest) for which the node set is $\mathcal{X}$ and the edge set is contained in $\mathcal{E}$. Let $\mathcal{K}_n$ denote the complete digraph of order n. Then, by Cayley's formula, the number of spanning trees of $\mathcal{K}_n$ rooted at a fixed node is n^{n-2}, and hence, the number of spanning forests of $\mathcal{K}_n$ is $(n+1)^{n-1}$.

For a digraph $\mathcal{G}$ with node set $\{X_1, \ldots, X_n\}$, a *linear extension* of $\mathcal{G}$ is a permutation π over $\{1, \ldots, n\}$ such that, if there is an edge from X_i to X_j, then $\pi(i) < \pi(j)$. It is well-known that if $\mathcal{G}$ is acyclic, then a linear extension of $\mathcal{G}$ can be constructed in $\mathcal{O}(|\mathcal{G}|)$ time using a topological sort algorithm.

Finally, a *weighted digraph* is a triple $\mathcal{G} = (\mathcal{X}, \mathcal{E}, w)$, where $(\mathcal{X}, \mathcal{E})$ is a digraph and w is a map from $\mathcal{X} \times \mathcal{X}$ to the set of nonnegative reals, such that $w(X, Y) > 0$ if and only if $(X, Y) \in \mathcal{E}$. For any subgraph $\mathcal{G}'$ of $\mathcal{G}$, the *weight* of $\mathcal{G}'$ is given by the product of weights of its edges. The *Laplacian* of $\mathcal{G}$ is the real matrix $\boldsymbol{\Lambda}(\mathcal{G})$ over $\mathcal{X} \times \mathcal{X}$ for which each entry $\lambda(X, Y)$ is given by:

$$\lambda(X, Y) = \begin{cases} \sum_{Z \in \mathcal{X}} w(Z, Y) & \text{if } X = Y \\ -w(X, Y) & \text{otherwise} \end{cases}$$

Let $\boldsymbol{\Lambda}_X(\mathcal{G})$ be the matrix obtained by deleting the row of X and the column of X from $\boldsymbol{\Lambda}(\mathcal{G})$. By the *Matrix-Tree Theorem* (Tutte 1984, Theorem 6.27), the determinant of $\boldsymbol{\Lambda}_X(\mathcal{G})$ is equal to the sum of weights of all spanning trees of $\mathcal{G}$ rooted at X. Based on this property, various algorithms have been proposed in the literature for generating in polynomial time random spanning trees of digraphs (see e.g. Kulkarni 1990; Colbourn et al. 1996).

3 RCP-nets

On the surface, our representation for relational preferences is similar to a probabilistic relational model (Getoor et al. 2002): the representation is structured in a graphical way by exploiting conditional independencies. However, the nature of connections between nodes in the graph is different: whereas conditional probabilities are *quantitative* and specify a probability measure over the outcome space, conditional preferences are *qualitative* and specify a strict partial order between outcomes.

3.1 Language

The basic building block of our framework is a *relational schema* that specifies a database structure. In order to clarify dependencies among the attributes of interconnected objects, the schema is represented as a digraph $\mathcal{S}$.

The nodes of $\mathcal{S}$ are separated into *class names* and *attribute names*. Intuitively, a class name denotes a type of objects, and an attribute name captures an elementary property that can be attached to a class name. Each attribute name A is associated with two predefined components: a finite domain D_A and a finite set Γ_A of aggregate functions or *aggregators*. Each $\gamma_A \in \Gamma_A$ maps any vector of values in D_A into a single value of D_A. Common aggregators include the mode (most frequently occurring value), the median, maximum or minimum (if values are ordinals), and the mean (if values are cardinals). The *domain size* of $\mathcal{S}$ is given by the maximum of the sizes of its domains.

The edges of $\mathcal{S}$ capture functional constraints between nodes: they are separated into *attributes* and *references*. An attribute is an edge of the form (X, A), also denoted $X.A$, where X is a class name and A an attribute name. A reference is an edge of the form $X.Y$ where X and Y are class names.

There is a natural correspondence between our representation and that of relational databases. Each class name X is associated with a table and each of its adjacent nodes is associated with a column in the table. For an attribute $X.A$, the entries in the corresponding column are values in D_A, and for a reference $X.Y$, the entries are foreign keys, each identifying an object in Y. We note in passing that this representation does not prevent us from having complex relationships between entities: using a standard reification technique, each k-ary relationship can be captured by introducing a new class name associated with k references, in which each object corresponds to a row in the relationship. These notions are illustrated in the following example.

Example 1 Suppose we would like to design a movie recommender system that periodically suggests a list of movies to each subscriber. The relational schema, described in Fig. 1, is composed of movies, actors, directors, critics, and users. Each box specifies a class name with its adjacent attribute names (in roman style) and reference names (in italic style); the dotted lines indicate the types of objects referenced. For instance, the class name Cast specifies the rank of actors playing in a movie; this class is associated with the attribute Cast.Rank and the references Cast.Actor and Cast.Movie.

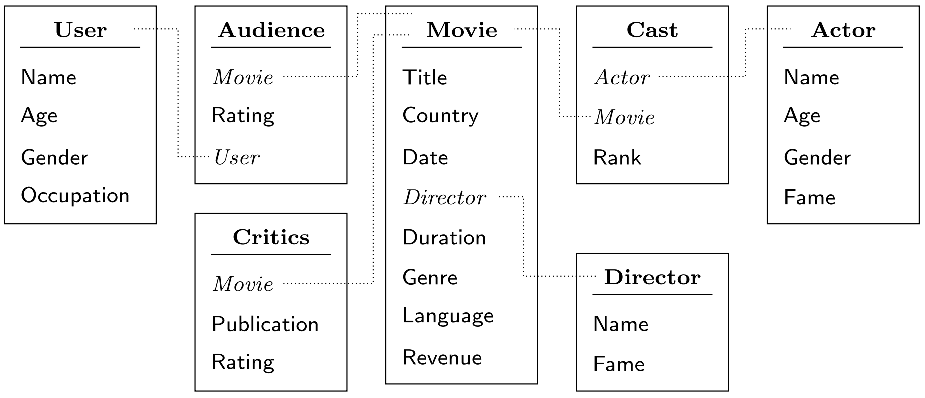


Fig. 1 A relational schema for the movie domain

A *chain* is a path in the underlying graph of $\mathcal{S}$ of the form $X.R$ where X is a class name and R is a (possibly empty) sequence of class names. A *slot* is an expression of the form $X.R.A$ where $X.R$ is a chain and A is an attribute name connected to the last class name of $X.R$. Intuitively, $X.R.A$ denotes a binary relation between objects of type X and values of type A. Note that because any slot is a path in the symmetric closure of $\mathcal{S}$, the relation captured by $X.R.A$ is *not* necessarily functional. For example, the slot User.Audience.Movie.Genre refers to all genres of movies watched by a user. A *term* is an aggregated slot, that is, an expression of the form $\gamma_A(X.R.A)$ where $X.R.A$ is a slot and γ_A is an aggregator in Γ_A. The set of all attributes occurring in $\mathcal{S}$ is denoted $\mathcal{A}(\mathcal{S})$ and the set of all possible terms that can be generated from $\mathcal{S}$ is denoted $\mathcal{T}(\mathcal{S})$.

In this study, preference relations are modeled as strict partial orders. Formally, given an arbitrary set $\mathcal{X}$, a *preference ordering* $\succ$ on $\mathcal{X}$ is a binary relation on $\mathcal{X}$ that is irreflexive, antisymmetric and transitive.

Definition 1 For a set $\mathcal{A} \subseteq \mathcal{A}(S)$ of attributes and a set $\mathcal{T} \subseteq \mathcal{T}(\mathcal{S})$ of terms, a *relational conditional preference network (RCP-net)* is a pair $N = (par, cpt)$ such that:

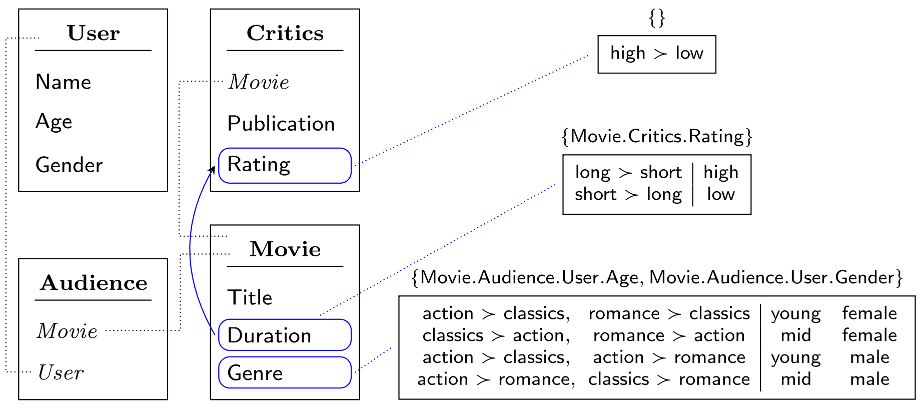
- *par* associates to each attribute $X.A \in \mathcal{A}$ a *parent set* $par(X.A)$, which is a collection $\{\gamma_{A_1}(X.R_1.A_1), \ldots, \gamma_{A_p}(X.R_p.A_p)\}$ of terms in $\mathcal{T}$ rooted at X.
- *cpt* associates to each attribute $X.A \in \mathcal{A}$ a *conditional preference table* $cpt_{X.A}$, which maps each vector $\boldsymbol{u}$ in $D_{A_1} \times \cdots \times D_{A_p}$ into a preference ordering $cpt_{X.A}(\boldsymbol{u})$ over D_A.

By $\mathcal{N}[\mathcal{A}, \mathcal{T}]$ we denote the class of all RCP-nets defined over the set $\mathcal{A}$ of attributes taking parents in the set $\mathcal{T}$ of terms. Each attribute in $\mathcal{A}$ is said to be *controllable*. For example, in a movie recommender system, it is legitimate to consider that movie attributes, including the genre, the release date and the duration of a film, are controllable. On the other hand, user attributes such as the age, the gender and the occupation of a person, are typically uncontrollable.

Given an RCP-net $N \in \mathcal{N}[\mathcal{A}, \mathcal{T}]$, the *dependency graph* of N is the digraph $\mathcal{G}(N)$ with node set $\mathcal{A}$ and such that there is an edge from $X.A$ to $X'.A'$ if and only if $X'.A'$ is the suffix of some slot in $par(X.A)$. Based on this notion, an RCP-net is *acyclic* if its dependency graph is acyclic, and *tree-structured* if its dependency graph is a forest. Finally, an RCP-net is *bipartite-ordered* (resp. *star-ordered*) if each of the entries of its conditional preference tables is a complete bipartite digraph (resp. a star).

The *parent size* of N is the maximum number p of parents per attribute in N. It is important to keep in mind that the parent size of an RCP-net does not necessarily coincide with the in-degree of its dependency graph. In particular, a tree-structured RCP-net N can have parent sets composed of multiple terms, provided that at most one suffix is in $\mathcal{A}$.

Example 2 Consider a restricted view of our movie recommendation domain, described in Fig. 2. The RCP-net N is defined over the set $\mathcal{A}$ of controllable attributes including Critics.Rating, Movie.Duration and Movie.Genre. We assume here that all attributes are associated with the *mode* aggregator. The dependency graph of N is depicted in the left part of the figure, while the parent sets and preference tables of N are presented in the right part. For instance, the first entry of the table associated to Movie.Duration states that a long movie is preferred over a short one if the aggregated reviews for this film are positive. Based on the above terminology, we can observe that N is both tree-structured and bipartite-ordered. In particular, the parents of Movie.Genre are defined over uncontrollable attributes.



User
Name
Age
Gender
Critics
Movie
Publication
Rating
Audience
Movie
User
Movie
Title
Duration
Genre
{}
high ≻ low
{Movie.Critics.Rating}
long ≻ short | high
short ≻ long | low
{Movie.Audience.User.Age, Movie.Audience.User.Gender}
action ≻ classics, romance ≻ classics | young female
classics ≻ action, romance ≻ action | mid female
action ≻ classics, action ≻ romance | young male
action ≻ romance, classics ≻ romance | mid male

Fig. 2 A tree-structured RCP-net for the movie domain

3.2 Semantics

Given a schema $\mathcal{S}$, a *skeleton* for $\mathcal{S}$ is a map κ that assigns to each class name X a finite set $[\![X]\!]_\kappa$ of *objects*, and to each reference $X.Y$ a function $[\![X.Y]\!]_\kappa$ from $[\![X]\!]_\kappa$ into $[\![Y]\!]_\kappa$. We assume that each object is associated to a unique class, i.e. $[\![X]\!]_\kappa \cap [\![Y]\!]_\kappa = \varnothing$ whenever $X \neq Y$. Based on the standard semantics of inverse and composition operations, a skeleton assigns a binary relation to any chain in the schema. A *ground chain* is an expression of the form $o.R$ where $X.R$ is a chain and o is an object in $[\![X]\!]_\kappa$. The notions of *ground attribute* and *ground term* are defined similarly. For a ground chain $o.R$, we denote by $[\![o.R]\!]_\kappa$ the set of objects o' such that $(o, o') \in [\![X.R]\!]_\kappa$.

Given a skeleton κ and a collection of attributes $\mathcal{A}$, we denote by $\mathcal{A}_\kappa$ the set of ground attributes $\{o.A : X.A \in \mathcal{A},\ o \in [\![X]\!]_\kappa\}$. For an RCP-net N, the *ground dependency graph* of N with respect to κ is the digraph $\mathcal{G}_\kappa(N)$ with node set $\mathcal{A}_\kappa$ and such that there is an edge from $o.A$ to $o'.A'$ if and only if there is an edge from $X.A$ to $X'.A$ in $\mathcal{G}(N)$, where X is the class of o and X' is the class of o in κ. Clearly, if $\mathcal{G}(N)$ is acyclic then $\mathcal{G}_\kappa(N)$ is acyclic.

A *relational outcome* or *interpretation* is a map I that extends a skeleton κ by assigning to each attribute $X.A$ a function $[\![X.A]\!]_I$ from $[\![X]\!]_\kappa$ into D_A. By $\mathcal{I}_\kappa$, we denote the space formed by all interpretations extending κ. Given a ground attribute $o.A$, the value assigned by I to $o.A$ is denoted $[\![o.A]\!]_I$. More generally, given a ground slot $o.R.A$ where $[\![o.R]\!]_\kappa = \{o_1, \ldots, o_n\}$, we denote by $[\![o.R.A]\!]_I$ the vector $\boldsymbol{v}$ in D_A^n such that $v_i = [\![o_i.A]\!]_I$ for $1 \leq i \leq n$.

With these notions in hand, we are now ready to examine the *ceteris paribus* semantics of relational preference networks. Consider an RCP-net N defined over a set of attributes $\mathcal{A}$, and a skeleton κ. A pair (I, J) of interpretations extending κ is called a *flip* on a ground attribute $o.A \in \mathcal{A}_\kappa$ if they are everywhere identical on $\mathcal{A}_\kappa$, excepted for $o.A$. Given an attribute $X.A$ with parent set $\{\gamma_{A_1}(X.R_1.A_1), \ldots, \gamma_{A_p}(X.R_p, A_p)\}$ and a flip (I, J) on $o.A$, we say that I *dominates* J in N if the value $[\![o.A]\!]_I$ is preferred to the value $[\![o.A]\!]_J$ in the entry of the table $cpt_{X.A}$ associated to the vector:

$$[\![par(o.A)]\!]_I = \big[\gamma_{A_1}\big([\![o.R_1.A_1]\!]_I\big), \ldots, \gamma_{A_p}\big([\![o.R_p.A_p]\!]_I\big)\big]$$

By extension, given any pair (I, J) of interpretations in $\mathcal{I}_\kappa$, we say that I *dominates* J in N, and write $I \succ_N J$, if there is a sequence $(I_1, \ldots, I_n)$ of flips such that $I_1 = I$, $I_n = J$, and I_i dominates I_{i+1} in N_κ for $1 \leq i < n$.

Table 1 Two relational outcomes for the movie domain

	User			Critics			Movie		
	Name	Age	Gender	Pub.	Title	Rating	Title	Dur.	Genre
I_1	Mary	mid	female	IMDb	Big Fish	high	Big Fish	long	romance
I_2					GoodFellas		GoodFellas		action

Example 3 Consider two relational outcomes I_1 and I_2 for the schema given in Fig. 2, specified in the Table 1. We remark that (I_1, I_2) is a flip on the movie genre. Thus, using the conditional preference table of Movie.Genre it follows that I_1 dominates I_2.

Finally, we say that an RCP-net N is *coherent* if, for any skeleton κ, the binary relation $\succ_N$ over $\mathcal{I}_\kappa$ is a preference ordering.

Theorem 1 *Any acyclic RCP-net is coherent.*

Proof Let N be an acyclic RCP-net defined over a set of attributes $\mathcal{A}$, let κ be a skeleton, and suppose that $I \succ_N I$ holds for some $I \in \mathcal{I}_\kappa$. By definition of the dominance relation, there is a sequence $(I_1, \ldots, I_n)$ of flips such that $I_1 = I = I_n$ and, $I_i \succ_N I_{i+1}$ for $1 \leq i < n$. Let S be the set of all ground attributes $o.A$ in $\mathcal{A}_\kappa$ such that $[\![o.A]\!]_{I_i} \neq [\![o.A]\!]_{I_{i+1}}$ for some flip (I_i, I_{i+1}). Consider any linear extension of S according to $\mathcal{G}_\kappa(N)$, and take the first element $o.A$ in the ordering. Let i be the first index in the sequence such that $[\![o.A]\!]_{I_i} \neq [\![o.A]\!]_{I_{i+1}}$. It follows that $[\![o.A]\!]_I \neq [\![o.A]\!]_{I_{i+1}}$. However, because $\mathcal{G}_\kappa(N)$ is acyclic, there is no parent of $o.A$ in S, and hence the ground condition $[\![par(o.A)]\!]_I$ remains unchanged during all the sequence of flips. It follows that $[\![o.A]\!]_{I_{i+1}} = [\![o.A]\!]_{I_{i+2}} = \cdots = [\![o.A]\!]_{I_n}$, and hence $[\![o.A]\!]_I \neq [\![o.A]\!]_{I_n}$, which contradicts the assumption that $I \succ_N I$. □

For the problems under consideration in the remaining sections, we will assume a prefixed and known schema $\mathcal{S}$ of domain size d, in which any aggregator can be implemented by a procedure that runs in $\mathcal{O}(n \log d)$ time, where n is the maximum number of objects per class name assigned by a skeleton. Because the size of preference tables grows exponentially with the number of parents, we will also assume that the parent size p of any RCP-net is *constant*.

4 Preference reasoning

For a class of RCP-nets, a *reasoning task* consists in a set of instances and a set of solutions. Of particular interest here is the class $\mathcal{N}_{\mathrm{acy}}[\mathcal{A}, \mathcal{T}]$ of acyclic RCP-nets. The size of $\mathcal{A}$ is denoted a, and the maximum of the sizes of terms in $\mathcal{T}$ is denoted k. The next lemma states that the parent values of an attribute can be retrieved in quasi-linear time using standard join and projection operations.

Lemma 1 *Let N be an RCP-net in $\mathcal{N}_{\mathrm{acy}}[\mathcal{A}, \mathcal{T}]$. Then, for any interpretation I and any ground attribute $o.A$, the vector of parent values $[\![par(o.A)]\!]_I$ can be constructed in $\mathcal{O}(t)$ time, where $t = kn \log_2(n) + n \log_2(d)$.*

Proof Consider any reference $X.Y$ in the schema $\mathcal{S}$. The inverse $[\![Y.X]\!]_I$ of the relation $[\![X.Y]\!]_I$ can be computed in $\mathcal{O}(n)$ time. In addition, the tuples in $[\![X.Y]\!]_I$ (or its inverse) can be ordered lexicographically, which requires $\mathcal{O}(n \log_2 n)$ steps. Based on this ordering, the composition $[\![X.Y]\!]_I \circ [\![Y.Z]\!]_I$ can be performed in $\mathcal{O}(n)$ time, in the following way: project $[\![X.Y]\!]_I$ onto the objects shared by $[\![Y.Z]\!]_I$ and prune any tuple in $[\![Y.Z]\!]_I$ that has no match in that projection. Thus, the ith value $\gamma_{A_i}([\![o.R_i.A_i]\!]_I)$ of the tuple $[\![par(o.A)]\!]_I$ can be found in $\mathcal{O}(kn \log_2 n + n \log_2 d)$ time using at most k join operations and one aggregate operation. Since the number of parents per attribute is constant, the result follows. □

4.1 Preference optimization

A *partial outcome* is an interpretation I that assigns the value "$*$" (unknown) to some ground attributes. A *completion* of I with respect to a set of attributes $\mathcal{A}$, is a map that extends I by replacing the unknown value of each attribute in $\mathcal{A}_\kappa$ with a value of appropriate type. J is *optimal* for an RCP-net N if there is no distinct completion J' of I such that $J' \succ_N J$.

Based on these considerations, an *outcome optimization task* for $\mathcal{N}[\mathcal{A}, \mathcal{T}]$ is a reasoning task in which instances are partial outcomes and solutions are outcome completions with respect to $\mathcal{A}$. Given an RCP-net N and an instance I, the task is to find a completion J of I that is optimal for N.

For acyclic RCP-nets, such a completion can be found in polynomial time using the *forward sweep* algorithm (Boutilier et al. 2004b), adapted to relational domains. Given an RCP-net N defined over an attribute set $\mathcal{A}$ and a partial outcome I, we first construct a topological ordering of $\mathcal{A}_\kappa$ according to the ground dependency graph $\mathcal{G}_\kappa(N)$, where κ is the skeleton of I. Then, starting from $J = I$, we instantiate each $o.A \in \mathcal{A}_\kappa$ in turn to a maximally preferred value in the preference ordering $cpt_{X.A}([\![par(o.A)]\!]_I)$.

Example 4 Suppose that Ann is a young woman. Based on the tree-structured RCP-net in Fig. 2, what would be her favorite romance movies? Starting from the partial outcome I in which only Ann's attributes are known, we can derive two optimal completions J_1 and J_2 of I. For both completions, the value of Critics.Rating is set to high, and the value of Movie.Duration is set to long. The value of Movie.Genre is action for J_1 and romance for J_2.

Theorem 2 *Let N be an RCP-net in $\mathcal{N}_{\text{ACY}}[\mathcal{A}, \mathcal{T}]$. Then, for any partial outcome I, finding an optimal completion of I for N can be done in $\mathcal{O}(ant)$ time.*

Proof We first establish the correctness of the forward sweep algorithm. Let I be a partial outcome, and J be the completion of I returned by the algorithm. Suppose that $J' \succ_N J$ for some distinct completion J' of I. In this case, there is a sequence of flips from J' to J in $\succ_N$. Let S be the set of ground attributes for which the value is switched in the sequence. Because $\mathcal{G}_\kappa(N)$ is acyclic, there is at least one $o.A \in S$ for which the values of all parents are fixed by I. Since J and J' are both extensions of I, we have $[\![par(o.A)]\!]_I = [\![par(o.A)]\!]_J = [\![par(o.A)]\!]_{J'}$. However, because $[\![o.A]\!]_J$ is optimal, the value of $o.A$ cannot be switched in the sequence, which contradicts the assumption that $J' \succ_N J$.

Now, consider the computational cost of the algorithm. A linear extension of $\mathcal{G}(N)$ can be found in $\mathcal{O}(a)$ time. Based on this ordering, a linear extension of $\mathcal{G}_\kappa(N)$ can be found in $\mathcal{O}(an)$ time, by replacing the attribute at position i with its n ground instances at positions $i_1, \ldots, i_n$, where $i_1 > i - 1$ and $i_n < i + 1$. Since by Lemma 1 the parent assignment of each attribute can be found in $\mathcal{O}(t)$ time, the algorithm returns a completion of I in $\mathcal{O}(ant)$ time. □

4.2 Preference ranking

For a skeleton κ, an *outcome set* is any finite collection $S = \{I_1, \dots, I_m\}$ of interpretations in $\mathcal{I}_\kappa$. Note that all interpretations in an outcome set are defined over the same set of objects, but differ in the values assigned to object attributes. A *ranking* of S is a permutation π over $[m] = \{1, \dots, m\}$. We say that π is *consistent* with an RCP-net N if, for any pair I_i, I_j in S, $I_i \succ_N I_j$ implies that I_i occurs before I_j in π, i.e. $\pi(i) < \pi(j)$. An *outcome ranking task* is a reasoning task for which any instance is an outcome set of size m and any solution is a permutation over $[m]$. Given an RCP-net N and an instance S, the problem is to find a ranking π of S that is consistent with N.

This problem can be solved in polynomial time for acyclic RCP-nets using a compilation technique inspired from Boutilier et al. (2001) and Brafman and Domshlak (2008). For a skeleton κ, a *utility function* is a map $\phi : \mathcal{I}_\kappa \to \mathbb{R}$. Any utility function ϕ induces a preference ordering $\succ_\phi$ over $\mathcal{I}_\kappa$ such that $I \succ_\phi J$ if and only if $\phi(I) > \phi(J)$. The function ϕ is *consistent* with an RCP-net N if $\succ_N$ implies $\succ_\phi$, that is, every linear extension of $\succ_\phi$ is a linear extension of $\succ_N$. Based on these notions, the overall idea of the compilation technique is to map any acyclic RCP-net N and any skeleton κ into a utility function ϕ over $\mathcal{I}_\kappa$ that is consistent with N. The function ϕ can then be exploited for solving multiple ranking tasks defined over κ.

We now embark on technical aspects. The basic ingredients of ϕ are two weight mappings f and g, where f is used to quantify local preferences in the tables of N, while g is used to quantify global dependencies between attributes. Consider an attribute $X.A$ with parent set $\{X_1.R_1.A_1, \dots, X_p.R_p.A_p\}$. Then, for each vector $\boldsymbol{u}$ of parent values in $D_{A_1} \times \cdots \times D_{A_p}$ and each value v in D_A, $f(\boldsymbol{u}, v)$ is the number of descendants of v in the preference ordering $cpt_{X.A}(\boldsymbol{u})$. Note that $f(\boldsymbol{u}, v) \leq |D_A| - 1$, and $f(\boldsymbol{u}, v) = 0$ if v is a leaf.

The weight mapping g is constructed in a recursive way using a topological ordering of $\mathcal{G}(N)$. If $X.A$ is the first attribute in the ordering, then we set $g(X.A) = 1$. Assuming by induction hypothesis that g is fixed for the first $k - 1$ elements in the ordering, if the kth element $X.A$ has no parents, then $g(X.A) = 1$. Otherwise,

$$g(X.A) = \frac{1}{|D_A|} \min_{i=1,\dots,p} \left(\frac{g(X_i.A_i)}{\sum_{X_j.A_j \in ch(X_i.A_i)} |[\![X_j]\!]_\kappa|} \right) \tag{1}$$

where $\{X_1.A_1, \dots, X_p.A_p\}$ is the set of all parents of $X.A$ in $\mathcal{G}(N)$, and $ch(X_i.A_i)$ is the set of all children of $X_i.A_i$ in $\mathcal{G}(N)$. The aim of g is thus to distribute the weight of an attribute evenly between its ground children.

With these ingredients in hand, the utility function ϕ is specified as a sum of sub-utilities $\phi_{X.A}$, each defined for a controllable attribute $X.A$ in N. Namely, if $X.A$ is an attribute with parent set $par(X.A) = \{X_1.R_1.A_1, \dots, X_p.R_p.A_p\}$, then $\phi_{X.A}$ associates to each vector $\boldsymbol{u}$ in $D_{A_1} \times \cdots \times D_{A_p}$ and to each value v in D_A the utility $\phi_{X.A}(\boldsymbol{u}, v) = g(X.A) f(\boldsymbol{u}, v)$. Based on these sub-utility functions, the value of any interpretation I is simply given by:

$$\phi(I) = \sum_{X.A \in \mathcal{A}} \sum_{o \in [\![X]\!]_\kappa} \phi_{X.A}\big([\![par(o.A)]\!]_I, [\![o.A]\!]_I\big) \tag{2}$$

Example 5 Consider again the RCP-net N in Fig. 2, and suppose that three long movies o_1, o_2 and o_3 are presented to John, a middle-aged male user. Specifically, o_1 and o_2 are romance movies, while o_3 is an action movie. The corresponding reviews, denoted r_1, r_2

and r_3, are positive for o_1 and o_3, but quite negative for o_2. We denote by I_1 (resp. I_2 and I_3) the interpretation associated to the entities John and $\{r_1, o_1\}$ (resp. $\{r_2, o_2\}$ and $\{r_3, o_3\}$). Now, according to our utility function ϕ, we have $\phi(I_1) = 1 + 1/2 + 0 = 3/2$, $\phi(I_2) = 1/2$ and $\phi(I_3) = 5/2$. Therefore, the ranking $(3, 1, 2)$ is consistent with N.

Theorem 3 *Let N be an RCP-net in $\mathcal{N}_{\text{acy}}[\mathcal{A}, \mathcal{T}]$ and κ be a skeleton. Then, compiling N into a consistent utility function ϕ on $\mathcal{I}_\kappa$ can be done in $\mathcal{O}(ad^{p+1})$ time. Based on this utility function ϕ, finding a consistent ranking of any outcome set $S \subseteq \mathcal{I}_\kappa$ of size m can be done in $\mathcal{O}(antm \log_2 m)$ time.*

Proof Let N_κ be the CP-net that associates to each instance $o.A$ of an attribute $X.A$ in N a parent set $par(o.A)$ and a preference table $cpt_{o.A}$. Specifically, if $X.A$ is an attribute with parent set $\{X.R_1.A_1, \ldots, X.R_p.A_p\}$, then:

- $par(o.A)$ is the set of all attributes $o_{ij}.A_i$, such that $1 \le i \le p$, $1 \le j \le n_i$, and $[\![o.R_i]\!]_\kappa = \{o_{i1}, \ldots, o_{in_i}\}$,
- $cpt_{o.A}$ is the map that assigns to each vector $\boldsymbol{w} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)$ in the space $D(A_1)^{n_1} \times \cdots \times D(A_p)^{n_p}$ the preference ordering $cpt_{o.A}(\boldsymbol{w}) = cpt_{X.A}(\boldsymbol{v})$, where $\boldsymbol{v}$ is the aggregated vector $(\gamma_{A_1}(\boldsymbol{v}_1), \ldots, \gamma_{A_p}(\boldsymbol{v}_p))$.

We can observe that the dependency graph of N_κ is $\mathcal{G}_\kappa(N)$. By a direct application of Brafman and Domshlak (2008, Theorem 3), the utility function ϕ defined above is consistent with the CP-net N_κ. Since $\succ_{N_\kappa} = \succ_N$, it follows that ϕ is consistent with the RCP-net N.

Now, consider the computational cost required for constructing ϕ. For each of the values of each preference ordering in N, the weight $f(\boldsymbol{u}, v)$ can be computed in $\mathcal{O}(d)$ time. Since there are at most a attributes and d^p table entries per attribute, the weight map f can be constructed in $\mathcal{O}(ad^{p+1})$ time. The weight map g can be obtained in $\mathcal{O}(a)$ time by simply constructing a topological ordering of $\mathcal{G}(N)$ and memorizing the number of ground children per attribute during the traversal.

Finally, the utility $\phi(I)$ of any interpretation I in the outcome set S can be computed in $O(ant)$ time, as specified by equality (2). Thus, any ranking of S can be found in $\mathcal{O}(antm \log_2 m)$ time by labeling each $I \in S$ with ϕ and breaking ties arbitrarily. □

5 Preference learning

By extending our previous considerations, a *prediction task* for a class of RCP-nets $\mathcal{N}$ is a triple $(\mathcal{X}, \mathcal{Y}, \ell)$, where $\mathcal{X}$ is a space of instances, $\mathcal{Y}$ is a space of solutions, and $\ell : \mathcal{N} \times \mathcal{X} \times \mathcal{Y} \to [0, \lambda]$ is a λ-bounded *loss function*.

In the online learning model, the decision maker is a learning algorithm that observes instances of a prediction task in a sequence of rounds. At trial t, the algorithm receives an instance $x_t \in \mathcal{X}$ and is required to predict a corresponding solution $N_t(x_t) \in \mathcal{Y}$ using its current hypothesis $N_t \in \mathcal{N}$. Once the algorithm has predicted, the true solution $y_t \in \mathcal{Y}$ is revealed and the algorithm incurs the loss $\ell(N_t; x_t, y_t)$ that measures the discrepancy between the predicted solution $N_t(x_t)$ and the correct response y_t. The ultimate goal of the decision maker is to minimize the cumulative loss it suffers along its run. To achieve this goal, the algorithm is allowed to choose a new hypothesis in $\mathcal{N}$ at the end of each trial, possibly using a randomized strategy.

As mentioned in the introduction, we make no assumptions regarding the sequence of examples. In this general setting, the performance of the decision maker is measured relatively

```
Parameters:
– a finite class $\mathcal{N}$ of RCP-nets with its component set $\mathcal{C}(\mathcal{N})$,
– a prediction task $(\mathcal{X}, \mathcal{Y}, \ell)$, such that $\ell$ is linear for $\mathcal{N}$

Initialization: Set $\theta_1(C) = 0$ for each $C \in \mathcal{C}(\mathcal{N})$

Trials: for each round $t = 1, 2, \dots$
(1) choose $N_t$ according to the distribution $\mathbb{P}_{\theta_t}(N) \sim \exp[\sum_{C \in N} \theta_t(C)]$
(2) receive instance $x_t \in \mathcal{X}$
(3) predict solution $N_t(x_t) \in \mathcal{Y}$
(4) receive response $y_t \in \mathcal{Y}$
(5) choose $\eta_t$ and set $\theta_{t+1}(C) = \theta_t(C) - \eta_t \ell(C; x_t, y_t)$ for each $C \in \mathcal{C}(\mathcal{N})$
```

Fig. 3 The expanded Hedge algorithm

to the performance of the best hypothesis in $\mathcal{N}$. Namely, the *regret* of an online learning algorithm L with respect to a sequence $\{(x_t, y_t)\}$ of T examples is given by the difference between the expected cumulative loss of the algorithm and the cumulative loss of the best hypothesis chosen with the benefit of hindsight:

$$\mathit{regret}\big(L, \{(x_t, y_t)\}\big) = \mathbb{E}\left[\sum_t \ell(N_t; x_t, y_t)\right] - \min_{N \in \mathcal{N}} \sum_t \ell(N; x_t, y_t)$$

A class of hypotheses $\mathcal{N}$ is *online learnable* with respect to a prediction task $(\mathcal{X}, \mathcal{Y}, \ell)$ if there exists an online learning algorithm L such that, for any sequence of T examples, the regret of L is *sublinear* as a function of T. This condition implies that "on average" the algorithm performs as good as the best fixed hypothesis in hindsight. If, in addition, the computational complexity of L is polynomial in the dimension parameters associated to $\mathcal{N}$, $\mathcal{X}$, and $\mathcal{Y}$, then $\mathcal{N}$ is *efficiently* learnable.

In this section, any RCP-net is viewed as a set of *components*, whose data structure will be clarified shortly. Let $\mathcal{C}(\mathcal{N})$ be the set of all distinct components generated from a class of hypotheses $\mathcal{N}$. A loss function ℓ is *linear* for $\mathcal{N}$ if $\ell(N; x, y) = \sum_{C \in N} \ell(C; x, y)$ for any $N \in \mathcal{N}$, where $\ell(C; x, y)$ denotes the loss incurred by the component C on the example (x, y).

Our learning algorithm is a variant of the Hedge algorithm (Freund and Schapire 1997) adapted to structured models. Following Koolen et al. (2010), we call this algorithm *Expanded Hedge*. As indicated in Fig. 3, the algorithm maintains a parameter vector θ_t over $\mathcal{C}(\mathcal{N})$. On round t, the algorithm predicts with a hypothesis N_t chosen at random according to the exponential distribution $\mathbb{P}_{\theta_t}$ induced by θ_t. Then, θ_t is updated using the rule $\theta_{t+1}(C) = \theta_t(C) - \eta_t \ell(C; x_t, y_t)$, where η_t is an adaptive learning rate.

The following result can be derived by a simple adaptation of Hedge's amortized analysis (Cesa-Bianchi and Lugosi 2006, Theorem 2.2).

Lemma 2 *Let $\mathcal{N}$ be a finite class of RCP-nets, and $(\mathcal{X}, \mathcal{Y}, \ell)$ be a predication task where ℓ is a λ-bounded linear loss function for $\mathcal{N}$. Then, for any sequence $\{(x_t, y_t)\}$ of T examples, the regret of the Expanded Hedge algorithm with adaptive learning rate $\eta_t = (2/\lambda)\sqrt{\ln|\mathcal{N}|/t}$ satisfies*:

$$\mathit{regret}\big(L, \{(x_t, y_t)\}\big) \le \lambda\sqrt{T \ln|\mathcal{N}|}$$

The rest of this section is devoted to the learnability issue of the class $\mathcal{N}_{\texttt{tree}}$ of tree-structured and bipartite-ordered RCP-nets. In order to obtain an efficient online learning algorithm for this class, we need a polynomial time procedure for generating hypotheses at random according to an exponential distribution. To this end, we first present a general procedure for generating random subsets; this procedure is the backbone for constructing RCP-nets. Next, we investigate the problem of learning bipartite orderings, and then, we turn to the problem of learning tree structures. We conclude by applying the results to optimization and ranking tasks.

5.1 Generating random subsets

Let $\mathcal{B}$ be an arbitrary non-empty set of subsets of $[b] = \{1, \dots, b\}$, and let w be a map that associates to each subset $B \in \mathcal{B}$ a positive real $w(B)$ that captures the weight of B. In this section, we present a simple algorithm due to Kulkarni (1990) that generates random subsets of $[b]$ which belong to $\mathcal{B}$ and such that the probability of generating a given $B \in \mathcal{B}$ is proportional to $w(B)$.

Before presenting the algorithm, we introduce the following notations. Let U and V be subsets of $[b]$, not necessarily in $\mathcal{B}$, such that $U \subseteq V$. By $[U, V]$, we denote the interval $\{B \in \mathcal{B} : U \subseteq B \subseteq V\}$. The weight of $[U, V]$, denoted $w[U, V]$, is defined as the sum of weights of sets in $[U, V]$, i.e.

$$w[U, V] = \sum_{B \in [U,V]} w(B) \tag{3}$$

The algorithm, called *Random Subset* and described in Fig. 4, starts from the largest interval $[U, V]$ with $U = \varnothing$ and $V = [b]$, and iteratively shrinks $[U, V]$ until it contains a unique set in $\mathcal{B}$. Namely, for each element $i \in [b]$, the algorithm inserts i in U with probability p and removes i from V with probability $1 - p$, where p is the ratio of $w[U \cup \{i\}, V]$ to $w[U, V]$. By a reformulation of Theorem 2.1. in Kulkarni (1990), we get the following result.

Lemma 3 *For any nonempty set $\mathcal{B}$ of subsets of $[b]$ and any positive weight function w over $\mathcal{B}$, the Random Subset algorithm produces a random variable U taking values in $\mathcal{B}$ with distribution proportional to w.*

```
Parameters:
  – a nonempty set $\mathcal{B}$ of subsets of $[b]$
  – a positive weight function $w : \mathcal{B} \to \mathbb{R}_+$
Initialization: Set $U = \varnothing$ and $V = [b]$

For $i = 1$ to $b$
  (1) let $p = \frac{w[U \cup \{i\}, V]}{w[U,V]}$
  (2) generate $q$ uniformly at random in the interval $[0, 1]$
  (3) if $q \leq p$ then $U = U \cup \{i\}$, else $V = V - \{i\}$
  (4) $i = i + 1$
Return $U$
```

Fig. 4 The random subset algorithm

5.2 Parameter learning

After a digression on generating random subsets, let us investigate the problem of learning the conditional preference tables of RCP-nets. More precisely, we examine the class $\mathcal{N}_{\texttt{tree}}[par]$ where the parent set par is prefixed and known. In this setting, any component of the preference network is specified as a triplet of the form $C = (X.A, \boldsymbol{u}, v)$ where $X.A$ is an attribute with parent set $\{X_1.R_1, A_1, \ldots, X_p, R_p.A_p\}$, $\boldsymbol{u}$ is a vector over $D_{A_1} \times \cdots \times D_{A_p}$ and v is a value in D_A. Based on this notation, $cpt_{X.A}(\boldsymbol{u})$ is encoded by the set of components $(X.A, \boldsymbol{u}, v)$, where v is maximally preferred value in D_A.

Given a schema of domain size d involving a attributes, the number of components is bounded by ad^{p+1}, and hence, remains polynomial in a and d whenever the parent size p is taken as constant. By contrast, the cardinality of $\mathcal{N}_{\texttt{tree}}[par]$ is bounded by $(2^d - 1)^{ad^p}$, where $(2^d - 1)$ is the number of complete bipartite digraphs over d values. Because this cardinality grows exponentially with a and d, a key computational issue is to generate hypotheses at random according to a given distribution over $\mathcal{N}_{\texttt{tree}}[par]$. Fortunately, this issue can be circumvented by exploiting the bipartite structure of preference orderings.

Lemma 4 *For the class $\mathcal{N}_{\texttt{tree}}[par]$, let θ be a parameter vector over the component set $\mathcal{C}(\mathcal{N}_{\texttt{tree}}[par])$. Then, generating a hypothesis $N \in \mathcal{N}_{\texttt{tree}}[par]$ at random according to $\mathbb{P}_\theta$ can be done in $\mathcal{O}(ad^{p+2})$ time.*

Proof Consider the map w that assigns to each $N \in \mathcal{N}_{\texttt{tree}}[par]$ the weight

$$w(N) = \prod_{C \in N} w(C) \quad \text{where } w(C) = e^{\theta(C)} \tag{4}$$

By construction, w is proportional to $\mathbb{P}_\theta$.

Now, consider any attribute $X.A$ with parents $\{X_1.R_1.A_1, \ldots, X_p.R_p.A_p\}$, and let $\boldsymbol{u}$ be a vector in $D_{A_1} \times \cdots \times D_{A_p}$. Without loss of generality, assume that $D_A = \{1, \ldots, d\}$, and let $\{C_1, \ldots, C_d\}$ be the set of components of the form $C_i = (X.A, \boldsymbol{u}, i)$. Recall that any bipartite preference ordering B over D_A is encoded by the set of components $(X.A, \boldsymbol{u}, i)$, where i is maximally preferred value in D_A. Based on equality (4), the weight of B is given by $w(B) = \prod_{i \in B} w_i$, where $w_i = w(C_i)$. Let $[U, V]$ be the set of all nonempty subsets B of D_A such that $U \subseteq B \subseteq V$. Based on the definition of $w[U, V]$ given in (3) and using the specification of w in (4), we get that:

$$w[U, V] = \begin{cases} -1 + \prod_{j \in V}(1 + w_j) & \text{if } U = \varnothing \\ \prod_{i \in U} w_i \prod_{j \in V \setminus U}(1 + w_j) & \text{otherwise} \end{cases} \tag{5}$$

The first equality in (5) is obtained by taking the sum of weights of all subsets of V and removing the weight of the empty subset. The second equality follows by taking the sum of weights of all subsets of V which include U. Thus, using the Random Subset algorithm, we can generate in $\mathcal{O}(d^2)$ time a random bipartite ordering for the entry $cpt_{X.A}(\boldsymbol{u})$. Since there are at most a attributes and d^p tuples of values per attribute, the result follows. □

5.3 Structure learning

Let us turn to the more general glass $\mathcal{N}_{\texttt{tree}}[\mathcal{A}, \mathcal{T}]$ of tree-structured RCP-nets with attribute set $\mathcal{A}$ and term set $\mathcal{T}$. Since the parent structure is unknown, any component is now defined as a quadruplet $C = (X.A, par(X.A), \boldsymbol{u}, v)$, where $X.A$ is an attribute, $par(X.A)$ is a parent set, $\boldsymbol{u}$ is a vector over the domain of $par(X.A)$ and v is a value in D_A. As specified above, any entry $cpt_{X.A}(\boldsymbol{u})$ is encoded by the set of components $(X.A, par(X.A), \boldsymbol{u}, v)$, where v is maximally preferred value in D_A.

For a schema of domain size d, the number of components is bounded by at^pd^{p+1}, where $|\mathcal{A}| = a$ and $|\mathcal{T}| = t$. By contrast, the size of $\mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$ is bounded by $(a+1)^{a-1}(t^p)^a(2^d-1)^{ad^p}$, where $(a+1)^{a-1}$ is the number of spanning forests of the complete digraph of order a. Again, this combinatorial barrier can be handled by exploiting the Matrix-Tree Theorem.

Lemma 5 *For the class $\mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$, let θ be a parameter vector over the component set $\mathcal{C}(\mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}])$. Then, generating a hypothesis $N \in \mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$ at random according to $\mathbb{P}_\theta$ can be done in $\mathcal{O}(a^5 + a^2t^pd^p + ad^{p+2})$ time.*

Proof By analogy with the proof of Lemma 4, let w be the map that assigns to each $N \in \mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$ the weight $w(N)$ specified by equality (4). Again, w is by construction proportional to $\mathbb{P}_\theta$.

Let $X.A.$ be an attribute with parent set $par(X.A)$, and $\boldsymbol{u}$ be a vector in the domain of $par(X.A) = \{X_1.R_1, A_1, \ldots, X_p, R_p.A_p\}$. We also assume without loss of generality that $D_A = \{1, \ldots, d\}$. Let $\{C_1, \ldots, C_d\}$ be the set of components of the form $C_i = (X.A, par(X.A), \boldsymbol{u}, i)$, each associated with its weight $w_i = w(C_i)$. By $w[X.A, par(X.A), \boldsymbol{u}]$, we denote the sum of weights of all bipartite orderings over D_A. From equality (5), we get that

$$w\big[X.A, par(X.A), \boldsymbol{u}\big] = -1 + \prod_{i=1}^{d}(1 + w_i) \tag{6}$$

By extension, let $w[X.A, par(X.A)]$ be the sum of weights of all possible conditional preference tables defined over the attribute $X.A$ with parent set $par(X.A)$. Applying the distributive property,

$$w\big[X.A, par(X.A)\big] = \prod_{\boldsymbol{u} \in D_{A_1} \times \cdots \times D_{A_p}} w\big[X.A, par(X.A), \boldsymbol{u}\big] \tag{7}$$

Now, let $\mathcal{G}$ be the weighted digraph with node set $\mathcal{A} \cup \{\top\}$, where $\top$ is a "dummy" node denoting the absence of parent. The edge set is formed by all pairs of distinct attributes in $\mathcal{A}$, together with all pairs of the form $(X.A, \top)$ where $X.A \in \mathcal{A}$. If e is an edge of the form $(X.A, X'.A)$, then $w(e)$ is given by the sum of weights $w[X.A, par(X.A)]$ of all possible parent sets $par(X.A)$ including one term in $\mathcal{T}$ with suffix $X'.A'$ and most $p-1$ terms in $\mathcal{T}$ with uncontrollable suffix. Analogously, if e is of the form $(X.A, \top)$, then $w(e)$ is the sum of weights $w[X.A, par(X.A)]$ of all parent sets $par(X.A)$ formed by at most p terms in $\mathcal{T}$ with uncontrollable suffix.

Based on these specifications, the weighted digraph $\mathcal{G}$ can be constructed in $\mathcal{O}(a^2t^pd^p)$ time. Namely, there are $2\binom{a}{2} + a$ edges in the graph, and the weight of each edge is the sum of at most t^p weights of the form $w[X.A, par(X.A)]$, each being computed in $\mathcal{O}(d^p)$ time using equalities (6) and (7).

Finally, let $\operatorname{span}(\mathcal{G})$ denote the set of all spanning trees of $\mathcal{G}$ rooted at $\top$. For any spanning tree S in $\operatorname{span}(\mathcal{G})$, let $w(S)$ be the product of weights of its edges. By construction, $w(S)$ is equal to the sum of weights $w(N)$ of all RCP-nets $N \in \mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$ with parent structure S.

Therefore,

$$\mathbb{P}_\theta(N) = \frac{w(N)}{\sum_{S \in \mathrm{span}(\mathcal{G})} w(S)}$$

With this property in hand, let U, V be two subsets of edges of $\mathcal{G}$ such that $U \subseteq V$, and let $w[U, V]$ be the sum of weights of all spanning trees $S \in \mathrm{span}(\mathcal{G})$ for which $U \subseteq S \subseteq V$. By $\mathcal{G}[U, V]$ we denote the weighted digraph obtained from $\mathcal{G}$ by deleting the edges which are not in V, and contracting the edges which are in U. By Lemma 5.4 in Kulkarni (1990),

$$w[U, V] = \left(\prod_{e \in U} w(e)\right) \det\bigl(\boldsymbol{\Lambda}_\top\bigl(\mathcal{G}[U, V]\bigr)\bigr)$$

Thus, based on the Random Subset algorithm, we can generate at random a spanning tree S of $\mathcal{G}$ rooted at $\top$ in $\mathcal{O}(a^5)$ time. This, together with the fact that any RCP-net over S can be generated in $\mathcal{O}(ad^{p+2})$ time, yields the result. □

5.4 Applications

We now have all ingredients in hand to examine the learnability of tree-structured RCP-nets.

For optimization tasks, each example supplied to the decision maker consists in a partial outcome x and a completion y of x. Here, $\ell_{opt}(C, x, y)$ indicates whether C has made a prediction mistake on (x, y). Specifically, a component C of the form $(X.A, par(X.A), \boldsymbol{u}, v)$ is charged one mistake on (x, y) if there is at least one object o of type X for which the value of $o.A$ is incorrectly predicted, i.e. $[\![par(o.A)]\!]_y = \boldsymbol{u}$, $[\![o.A]\!]_x = *$, and $[\![o.A]\!]_y \neq v$. Based on this notion, $\ell_{opt}(N; x, y)$ is equal to the number of components C in N which have made a mistake on (x, y). Thus, ℓ_{opt} is an upper bound on the number of mistakes made by the decision maker. Together with the fact that ℓ_{opt} is bounded by $\lambda = a$, the composition of Lemmas 2 and 5 yields the following result.

Theorem 4 *The class $\mathcal{N}_{\mathtt{tree}}[\mathcal{A}, \mathcal{T}]$ is efficiently online learnable from outcome optimization tasks with regret bound*

$$a^{3/2}\sqrt{\bigl(\ln(a+1) + p \ln t + d^{p+1} \ln 2\bigr)T}$$

For ranking tasks, each example consists in an outcome set $x = \{I_1, \ldots, I_m\}$ and a permutation y over $[m] = \{1, \ldots, m\}$. Here, the loss $\ell_{\mathtt{rank}}(C, x, y)$ is recursively defined as follows. For $m = 2$, any component C of the form $(X.A, par(X.A), \boldsymbol{u}, v)$ is charged one mistake on $(\{I_1, I_2\}, (2, 1))$ if there is at least one object o of type X such that $[\![par(o.A)]\!]_{I_1} = [\![par(o.A)]\!]_{I_2} = \boldsymbol{u}$, $[\![o.A]\!]_{I_1} = v$ and $[\![o.A]\!]_{I_2} \neq v$. Now, for $m \geq 2$, $\ell_{\mathtt{rank}}(C; x, y)$ is the number of pairs in x for which C has made a prediction mistake:

$$\ell_{\mathtt{rank}}(C; x, y) = \sum_{i, j \in [m] : y(j) < y(i)} \ell_{\mathtt{rank}}\bigl(C; \{I_i, I_j\}, (j, i)\bigr)$$

Based on this definition, $\ell_{\mathtt{rank}}(N; x, y)$ is an upper bound on the *Kendall's tau distance* (1938) that calculates the number of pairs in x for which the permutations $N(x)$ and y have opposite orderings (called discordant pairs). Since $\ell_{\mathtt{rank}}$ is bounded by $\lambda = a\binom{m}{2}$, we get the following result.

Theorem 5 *The class $\mathcal{N}_{\texttt{tree}}[\mathcal{A}, \mathcal{T}]$ is efficiently online learnable from outcome ranking tasks with regret bound*

$$a^{3/2}\binom{m}{2}\sqrt{\left(\ln(a+1) + p\ln t + d^{p+1}\ln 2\right)T}$$

We note in passing that both regret bounds are independent of the potentially large number n of objects in the database.

6 Experiments

The experiments reported in the section are not meant to give a rigorous empirical evaluation of our theoretical framework. Instead, they are intended as an illustration of the typical behavior of Expanded Hedge for learning tree-structured RCP-nets, comparing the type of prediction task (optimization vs. ranking), the number p of parents per attributes, and the type of preference ordering over domain values (bipartite graph vs. star).

Our experiments are based on a recent benchmark in movie recommendation (Cantador et al. 2011) for which the schema is described in Fig. 1. As an integration of the MovieLens, IMDb and Rotten Tomatoes systems, the database includes 10198 movies, 7742 actors, 4053 directors, and 6040 anonymous users, where each user rated at least 20 movies.

In our experiments, outcomes are defined according to 4 objects: a user, a movie picked in her watchlist, the actress or actor with leading role in the movie, and the director of the movie. Each outcome involves 3 uncontrollable attributes, specified by the user's age, gender and occupation, and 25 controllable attributes including the starring actor's fame and gender, the director's fame, and the film's country, genres, release date, revenue and critics rating. Notably, because a movie can have multiple genres, the set-valued attribute Movie.Genre was split into 18 binary attributes, each referring to a particular category in IMDb. Movie ratings were formatted according to a five-star scale. Other numeric attributes were discretized using the standard "equal frequency discretization" technique with 5 intervals.

Each experiment was conducted by selecting a group of 50 users generated at random from 4 known occupations. The experiment lasts for 1000 training rounds and, after each series of 5 rounds, we measured the algorithm's accuracy on 100 test examples generated from our user group. The final results are obtained by averaging the algorithm's accuracy on 10 experiments.

Learning to optimize In movie recommendation, a natural optimization task is to predict the type of movie users would like to see based solely on their profile. In this setting, each example at round t was generated by selecting a user and a highest-rated movie in her watchlist, together with the movie's starring actor and director. By denoting y_t the resulting interpretation, the partial outcome x_t was obtained from y_t by removing the value of every controllable attribute.

In order to evaluate the performance of the decision maker, we must ensure that its predicted completion $N_t(x_t)$ covers at least one movie that has been seen by the user. To this end, the forward sweep algorithm was slightly modified as follows: for each attribute $o.A$ iteratively processed according to the dependency graph of N_t, generate a linear extension of the preference ordering corresponding to $[\![par(o.A)]\!]_{x_t}$, and instantiate $o.A$ to the first value in the linear extension that matches at least one movie in the user's watchlist. A mistake is made if the resulting completion $N_t(x_t)$ does not cover any highest-rated movie. In this case, the completion y_t is returned.

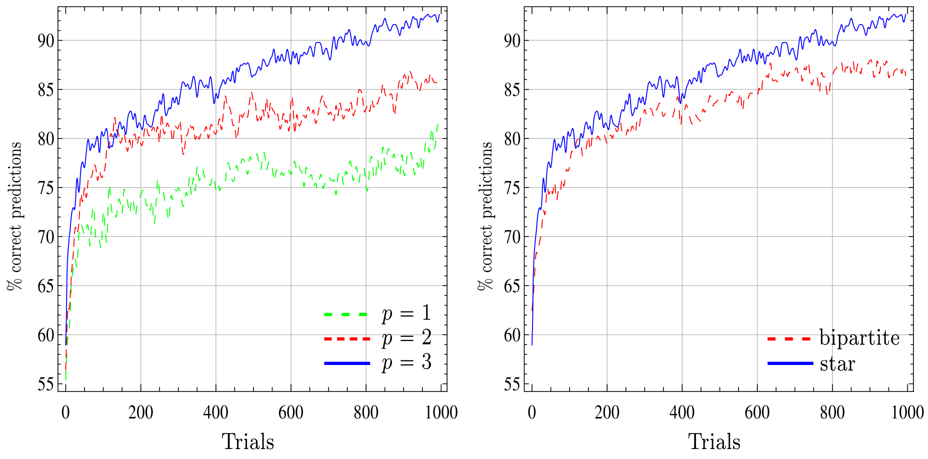


Fig. 5 Accuracy results on optimization tasks, comparing the parent sizes (*left part* with star orderings) and the type of preference orderings (*right part* with $p = 3$)

The algorithm's accuracy was measured by counting the number of correct predictions. The left part of Fig. 5 shows plots for learning tree-directed (and star-ordered) RCP-nets with parent size $p = 1, 2$ and 3. Such results indicate that uncontrollable attributes are crucial for prediction. Notably, by observing the cumulative loss of each component at the end of the 1000 rounds, the improvement from $p = 2$ to $p = 3$ is essentially due to the presence of rules involving both the user's gender and the user's age. The right part of Fig. 5 compares the algorithm's performance with star-ordered RCP-nets and bipartite-ordered RCP-nets. Here, the convergence rate is slightly faster with stars, indicating that a single maximally preferred value per entry in the tables of the network is sufficient for making accurate predictions.

Learning to rank Concerning ranking tasks, each instance x_t was generated by first selecting a user, and then by forming an outcome set of $m = 20$ movies, each selected at random from the user's watchlist. Here, a prediction mistake is made if the ranking $\hat{y}_t = N_t(x_t)$ is inconsistent with the user's ratings; if so, the response y_t is the closest ranking of $\hat{y}_t$ that is consistent with the user's ratings. Because our rank loss defined above is an upper bound on Kendall's tau distance, the accuracy was measured here using the corresponding *Kendall's tau coefficient*, given by $1 - 4D(\hat{y}_t, y_t)/m(m-1)$, where $D(\hat{y}_t, y_t)$ is the Kendall's tau distance between $\hat{y}_t$ and y_t.

The left part of Fig. 6 shows plots for learning bipartite ordered RCP-nets with parent size $p = 1, 2$ and 3. Again, it is apparent that the presence of uncontrollable attributes plays an important role on the algorithm's performance. However, a sharp contrast between optimization tasks and ranking tasks is observed when comparing star-ordered RCP-nets and bipartite-ordered RCP-nets. As plotted in the right part of Fig. 6, the degraded performance of the algorithm using star-shaped orderings indicates that such preference relations are too restrictive; the full expressiveness of bipartite orderings is required for ranking tasks.

Running times Both series of experiments were conducted on a 3.00 GHz Intel Xeon 5570 bi-processor with 8 GB RAM running Windows 7. All procedures were written in C++. For generating spanning trees we used the *cycle popping* algorithm due to Propp and Wilson (1998). Although this Markov chain-based procedure does not offer the theoretical guarantees of discriminant-based procedures, it is much simpler to implement. Based on this

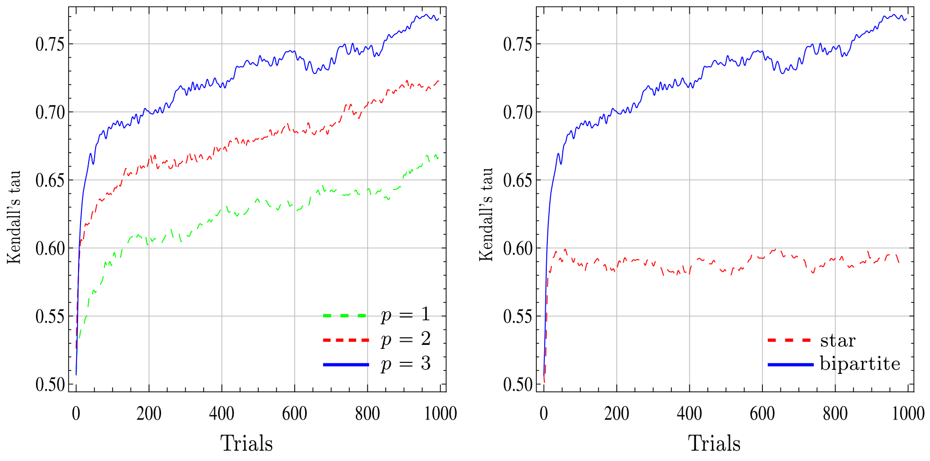


Fig. 6 Accuracy results on ranking tasks, comparing the parent sizes (*left part* with bipartite orderings) and the type of preference orderings (*right part* with $p = 3$)

algorithm and a B-tree data structure for storing components (about 10^6 for $p = 3$), the running time needed for generating tree-structured RCP-nets is less than 10 ms. Optimizing outcomes and ranking outcome sets with these RCP-nets takes less than 1 ms.

7 Related work

In recent years, the topic of preferences has received a great deal of attention in AI (see e.g. Brafman and Domshlak 2009; Fürnkranz and Hüllermeier 2011). A key issue in this topic is to devise preference models endowed with efficient forms of reasoning and learning. This section focuses on *preference networks* which adopt a graph-based representation for identifying preferential dependencies among variables. By analogy with probabilistic networks (Koller and Friedman 2009), preference networks can be divided into two main categories depending on whether the underlying structure is an undirected graph (more precisely a hypergraph[1] whose primal graph is undirected) or a digraph.

7.1 Undirected preference networks

Utility functions have long been recognized as a natural paradigm for modeling preferences. For an outcome space $\mathcal{I}$, a utility function is a map $\phi : \mathcal{I} \to \mathbb{R}$ that associates a "degree of desire" to each outcome I in $\mathcal{I}$. The function induces a total preorder $\succeq_\phi$ such that $I \succeq J$ if and only if $\phi(I) \leq \phi(J)$. By far, the most common approach for decomposing utility functions is the *additive independence* principle (Keeney and Raiffa 1976): a utility function ϕ is a weighted sum of sub-utility functions, or *features*, considered as pairwise independent. Given a multi-attribute representation of $\mathcal{I} = D_1 \times \cdots \times D_n$, the simplest class of utility functions satisfying this principle is the family of linear functions, each defined as a weighted sum over the attributes $\{X_1, \dots, X_n\}$.

[1] Recall that a hypergraph is a pair $\mathcal{H} = (\mathcal{X}, \mathcal{E})$ where $\mathcal{X}$ is a set of nodes and $\mathcal{E}$ is a set of subsets of $\mathcal{X}$, called hyperedges.

Since the strong independence assumption required by linear functions is too restrictive in many application domains, Fishburn (1967) introduced the *Generalized Additive Independence (GAI)* principle which allows for a similar additive decomposition of a utility function, but where overlapping subsets of attributes are the features involved rather than simple attributes. Bacchus and Grove (1995) and Gonzales and Perny (2004) proposed a graphical representation of this principle: a *GAI-net* is a pair $(\mathcal{H}, \phi)$ such that $\mathcal{H} = (\mathcal{X}, \mathcal{E})$ is a hypergraph with node set $\mathcal{X} = \{X_1, \ldots, X_n\}$, and ϕ is a map that associates to each hyperedge $E \in \mathcal{E}$ a feature $\phi_E : D_E \to \mathbb{R}$, where D_E is the product of domains of the variables in E. Also known as weighted CSPs (Dechter 2003), GAI-nets have been the subject of extensive research in the AI literature (Braziunas and Boutilier 2005; Boutilier et al. 2006; Gonzales et al. 2011). Since a GAI-net is merely a linear combination of features, various algorithms can be applied for learning GAI-nets from ranking instances in multi-attribute domains (Herbrich et al. 2000; Crammer and Singer 2001; Freund et al. 2003; Harrington 2003; Domshlak and Joachims 2005).

In this line of work, the closet framework to ours is due to Brafman (2008) who recently proposed a relational extension of GAI-nets. Intuitively, a relational GAI-net is a template over a database schema $\mathcal{S}$ that specifies a ground GAI-nets for each database of objects. More formally, a *template feature* is a pair (F, ϕ) where $F = \{\gamma_{A_1}(X.R_1.A_1), \ldots, \gamma_{A_p}(X.R_p, A_p)\}$ is a set of terms in $\mathcal{T}(\mathcal{S})$ and ϕ is a mapping from $D_{A_1} \times \cdots \times D_{A_p}$ into $\mathbb{R}$. A *relational GAI-net* over $\mathcal{S}$ is a set of template features. For a skeleton κ and an interpretation $I \in \mathcal{I}_\kappa$ the sub-utility assigned to I by a template feature (F, ϕ) is given by:

$$\phi(I) = \sum_{o \in [\![X_1]\!]_\kappa \times \cdots \times [\![X_p]\!]_\kappa} \phi\big([\![\gamma_{A_1}(o_1.R_1.A_1)]\!]_I, \ldots, [\![\gamma_{A_p}(o_p.R_1.A_p)]\!]_I\big)$$

Based on the generalized additive independence condition, the utility assigned by a relational GAI-net $N = \{(F_1, \phi_1), \ldots, (F_m, \phi_m)\}$ to any interpretation $I \in \mathcal{I}_\kappa$ is given by $\phi_N(I) = \sum_{i=1}^m \phi_i(I)$. Thus, relational GAI-nets can be used to compare relational outcomes and to answer relational preference queries. However, the crucial difference between RCP-nets and relational GAI-nets lies in the fact that preference optimization can be solved in polynomial time for acyclic RCP-nets, while it is NP-hard for relational GAI-nets (Brafman 2008, Theorem 1). To this point, finding subclasses of relational GAI-nets which are tractable for both reasoning and learning is a challenging problem.

7.2 Directed preference networks

From a cognitive viewpoint, the interest of directed graphical models lies their ability to represent conditional preferences in an intuitive manner (Boutilier et al. 2004b; Engel and Wellman 2008). Notably, CP-nets are directed graphical models in which the semantics of conditional preferences is defined according to the *ceteris paribus* principle. Various extensions of CP-nets have been proposed in multi-attribute domains (Boutilier et al. 2001; Brafman et al. 2006; Goldsmith et al. 2008; Binshtok et al. 2009). The learnability issue of CP-nets has been considered in different learning protocols including, notably, the distribution specific PAC learning model (Dimopoulos et al. 2009) and the exact learning model with equivalence and membership queries (Koriche and Zanuttini 2010). In a related setting, Yaman et al. (2011) developed efficient online algorithms for learning ensembles of lexicographic CP-nets whose dependency graph is a chain.

In a nutshell, our framework extends the CP-net formalism to handle reasoning and learning problems in relational domains. Because the number of objects in a database is typically large, an important aspect of our framework is to operate at the template level whenever as possible. Notably, optimization and ranking techniques presented in Theorems 2 and 3 do not require an explicit construction of the ground dependency graph $\mathcal{G}_\kappa(N)$ of the RCP-net N.[2] For this reason, the complexity of outcome optimization and outcome ranking is low-polynomial in the number n of objects.

Directed preference networks have similarities with *preference functions* described in Cohen et al. (1999). Conceptually, a preference function is a map $F : \mathcal{I} \times \mathcal{I} \to [0, 1]$ such that $F(I, J) = 1 - F(J, I)$. Following the additive independence principle, preference functions are defined as weighted sums of atomic functions. This line of work has been recently extended to the relational setting by Ceci et al. (2010) who use first-order logical rules for representing atomic functions. Yet, one of key issues with such approaches is that the weighted digraph over $\mathcal{I}$ induced by a preference function is not necessarily acyclic. For this reason, the problem of finding a ranking that is maximally consistent with a preference function is, in general, NP-hard.

8 Conclusions

In this study, we have presented a unifying framework for learning and reasoning with conditional preferences in relational domains. Our main theoretical results state that acyclic RCP-nets support tractable inference for both preference optimization and preference ranking, and tree-structured RCP-nets can be robustly learned from optimization and ranking tasks using linear loss functions. Our theoretical findings have been complemented by experiments on a large-scale recommendation domain.

Clearly, there are many directions in which one might attempt extensions of this study. Two of them are particularly important for addressing practical situations in relational domains.

Constraints In our framework, the decision maker's background knowledge is a relational schema with class names, attribute names, and functional dependencies among them. However, in many application domains, such as control systems (Brafman 2008) and configuration problems (Junker 2006), the background knowledge also involves other sorts of constraints. For example, the choice of a particular motherboard in a computer can restrict the choice of graphic cards. The sum of prices of the computer components cannot exceed the user's budget. In presence of such constraints, our result for outcome optimization (Theorem 2) no longer holds. To this point, the anytime algorithm suggested by Boutilier et al. (2004a) for CP-nets can provide a first step for investigating constrained optimization problems with RCP-nets.

Cyclic preferences All theoretical results presented in this paper are limited to classes of *acyclic* RCP-nets. As argued by Goldsmith et al. (2008), the acyclic assumption for conditional preferences can be too strong in some situations. Notably, a natural form of cyclic preference in relational domains arises from social networks: evidence suggests that people tend to rely more on recommendations from their friends than on recommendations from anonymous individuals (Sinha and Swearingen 2001; Golbeck 2006). However, from a semantical viewpoint cyclic RCP-nets are not always guaranteed to be coherent (Theorem 1

[2]For example, in the proof of Theorem 2, we only need the template graph $\mathcal{G}(N)$ in order to construct a topological ordering of $\mathcal{G}_\kappa(N)$.

does not hold for general RCP-nets). A challenging problem is thus to identify and learn coherent forms of cyclic RCP-nets capable of representing mutual influences between entities in relational domains.

Acknowledgements The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n^{o} 284715 for FP7-ICON project, and from the French National Research Agency under grant ANR-10-BLAN-0215 for LARDONS project.

References

Bacchus, F., & Grove, A. (1995). Graphical models for preference and utility. In *Proceedings of the 11th conference on uncertainty in artificial intelligence (UAI'95)*, Edinburgh, Scotland (pp. 3–10). Berkeley: AUAI Press.

Binshtok, M., Brafman, R. I., Domshlak, C., & Shimony, S. E. (2009). Generic preferences over subsets of structured objects. *The Journal of Artificial Intelligence Research*, *34*, 133–164.

Boutilier, C., Bacchus, F., & Brafman, R. (2001). UCP-networks: a directed graphical representation of conditional utilities. In *Proceedings of the 17th conference in uncertainty in artificial intelligence (UAI'01)*, Seattle, Washington, USA (pp. 56–64).

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Poole, D. (2004a). Preference-based constrained optimization with cp-nets. *Computational Intelligence*, *20*(2), 137–157.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Poole, D. (2004b). CP-nets: a tool for representing and reasoning with conditional Ceteris Paribus preference statements. *The Journal of Artificial Intelligence Research*, *21*, 135–191.

Boutilier, C., Patrascu, R., Poupart, P., & Schuurmans, D. (2006). Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artificial Intelligence*, *170*(8–9), 686–713.

Brafman, R., & Domshlak, C. (2008). Graphically structured value-function compilation. *Artificial Intelligence*, *172*(2–3), 325–349.

Brafman, R. I. (2008). Relational preference rules for control. In *Proceedings of the 11th conference on knowledge representation and reasoning (KR'08)*, Sydney, Australia (pp. 552–559). Berkeley: AAAI Press.

Brafman, R. I., & Domshlak, C. (2009). Preference handling—an introductory tutorial. *The AI Magazine*, *30*(1), 58–86.

Brafman, R. I., Domshlak, C., & Shimony, S. E. (2006). On graphical modeling of preference and importance. *The Journal of Artificial Intelligence Research*, *25*, 389–424.

Braziunas, D., & Boutilier, C. (2005). Local utility elicitation in Gai models. In *Proceedings of the 21st conference in uncertainty in artificial intelligence (UAI'05)*, Edinburgh, Scotland (pp. 42–49). Berkley: AUAI Press.

Cantador, I., Brusilovsky, P., & Kuflik, T. (2011). 2nd workshop on information heterogeneity and fusion in recommender systems (HetRec'11). In *Proceedings of the 5th ACM conference on recommender systems, RecSys'11*, Chicago, IL, USA. New York: ACM Press.

Ceci, M., Appice, A., Loglisci, C., & Malerba, D. (2010). Preference learning for document image analysis. In *3rd ECML/PKDD workshop on preference learning (PL-10)*, Barcelona, Spain.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge: Cambridge University Press.

Cohen, W., Schapire, R., & Singer, Y. (1999). Learning to order things. *The Journal of Artificial Intelligence Research*, *10*, 243–270.

Colbourn, C. J., Myrvold, W. J., & Neufeld, E. (1996). Two algorithms for unranking arborescences. *Journal of Algorithms*, *20*(2), 268–281.

Crammer, K., & Singer, Y. (2001). Pranking with ranking. In *Proceedings of the 15th international conference on neural information processing systems (NIPS'01)*, Vancouver, British Columbia, Canada (pp. 641–647). Cambridge: MIT Press.

Dechter, R. (2003). *Constraint processing*. San Mateo: Morgan Kaufmann.

Dimopoulos, Y., Michael, L., & Athienitou, F. (2009). Ceteris paribus preference elicitation with predictive guarantees. In C. Boutilier (Ed.), *Proceedings 21st international joint conference on artificial intelligence (IJCAI'09)*, Pasadena, California, USA (pp. 1890–1895).

Domshlak, C., & Joachims, T. (2005). Unstructuring user preferences: efficient non-parametric utility revelation. In *Proceedings of the 21st conference in uncertainty in artificial intelligence (UAI'05)*, Edinburgh, Scotland (pp. 169–177). Berkeley: AUAI Press.

Engel, Y., & Wellman, M. P. (2008). CUI networks: a graphical representation for conditional utility independence. *The Journal of Artificial Intelligence Research*, *31*, 83–112.

Fishburn, P. (1967). Independence and additivity in multivariate unidimensional expected utility theory. *International Economic Review*, *8*, 335–342.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, *4*, 933–969.

Fürnkranz, J., & Hüllermeier, E. (2011). *Preference learning*. Berlin: Springer.

Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2002). Learning probabilistic models of link structure. *Journal of Machine Learning Research*, *3*, 679–707.

Golbeck, J. (2006). Generating predictive movie recommendations from trust in social networks. *Lecture notes in computer science: Vol. 3986*. In *Proceedings of the 4th international conference on trust management (iTrust'06)*, Pisa, Italy (pp. 93–104). Berlin: Springer.

Goldsmith, J., Lang, J., Truszczynski, M., & Wilson, N. (2008). The computational complexity of dominance and consistency in CP-nets. *The Journal of Artificial Intelligence Research*, *33*, 403–432.

Gonzales, C., & Perny, P. (2004). GAI networks for utility elicitation. In *Principles of knowledge representation and reasoning: proceedings of the ninth international conference (KR'04)*, Whistler, Canada (pp. 224–234). Menlo Park: AAAI Press.

Gonzales, C., Perny, P., & Dubus, J. Ph. (2011). Decision making with multiple objectives using GAI networks. *Artificial Intelligence*, *175*(7), 1153–1179.

Harrington, E. F. (2003). Online ranking/collaborative filtering using the Perceptron algorithm. In *Proceedings of the 20th international conference on machine learning (ICML'03)*, Washington, DC, USA (pp. 250–257). Menlo Park: AAAI Press.

Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In *Advances in large margin classifiers* (pp. 115–132). Cambridge: MIT Press.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: an introduction*. Cambridge: Cambridge University Press.

Junker, U. (2006). Configuration. In *Handbook of constraint programming* (pp. 837–873). Amsterdam: Elsevier, Chap. 24.

Keeney, R. L., & Raiffa, H. (1976). *Decisions with multiple objectives: preferences and value tradeoffs*. New York: Wiley.

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, *30*, 81–93.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models*. Cambridge: MIT Press.

Koolen, W. M., Warmuth, M. K., & Kivinen, J. (2010). Hedging structured concepts. In *Proceedings of the 23th conference on learning theory (COLT'10)*, Haifa, Israel (pp. 93–105).

Koriche, F., & Zanuttini, B. (2010). Learning conditional preference networks. *Artificial Intelligence*, *174*(11), 685–703.

Kulkarni, V. G. (1990). Generating random combinatorial objects. *Journal of Algorithms*, *11*(2), 185–207.

Melville, P., Mooney, R. J., & Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the eighteenth national conference on artificial intelligence (AAAI'02)*, Edmonton, Alberta, Canada (pp. 187–192). Menlo Park: AAAI Press.

Newton, J., & Greiner, R. (2004). Hierarchical probabilistic relational models for collaborative filtering. In *3rd workshop on statistical relational learning (SRL'04)*, Banff, Alberta, Canada.

Propp, J. G., & Wilson, D. B. (1998). How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, *27*(2), 170–217.

Sinha, R. R., & Swearingen, K. (2001). Comparing recommendations made by online systems and friends. In *DELOS workshop on personalisation and recommender systems in digital libraries*, Dublin, Ireland.

Tutte, W. T. (1984). Graph theory. Cambridge.

Yaman, F., Walsh, T. J., Littman, M. L., & des Jardins, M. (2011). Democratic approximation of lexicographic preference models. *Artificial Intelligence*, *175*, 1290–1307.