UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LINGUAGENS DE CONSULTA PARA BANCO DE DADOS COM SUPORTE A PREFERÊNCIAS CONDICIONAIS

MARCOS ROBERTO RIBEIRO

Uberlândia - Minas Gerais
2008

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCOS ROBERTO RIBEIRO

LINGUAGENS DE CONSULTA PARA BANCO DE DADOS COM SUPORTE A PREFERÊNCIAS CONDICIONAIS

Dissertação de Mestrado apresentada à Faculdade de Ciência da Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados.

Orientadora:
Prof[a]. Dr[a]. Sandra Aparecida de Amo

Uberlândia, Minas Gerais
2008

Dados Internacionais de Catalogação na Publicação (CIP)

R484L Ribeiro, Marcos Roberto, 1982-
Linguagens de consulta para banco de dados com suporte a preferências condicionais / Marcos Roberto Ribeiro. - 2008.
161 f. : il.

Orientadora: Sandra Aparecida de Amo.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Banco de dados - Teses. I. Amo, Sandra Aparecida de. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3.07

Elaborado pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Ciência da Computação a aceitação da dissertação intitulada "**Linguagens de Consulta para Banco de Dados com Suporte a Preferências Condicionais**" por **Marcos Roberto Ribeiro** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 3 de Dezembro de 2008

Orientadora: _____

Profª. Drª. Sandra Aparecida de Amo
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. João Nunes de Souza
Universidade Federal de Uberlândia

_____

Prof. Dr. Caetano Traina Junior
Universidade de São Paulo

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Dezembro de 2008

Autor:	**Marcos Roberto Ribeiro**
Título:	**Linguagens de Consulta para Banco de Dados com Suporte a Preferências Condicionais**
Faculdade:	**Faculdade de Ciência da Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

Aos meus pais Gezu e Santinha e a minha irmã Mírian Cecília.
A minha namorada Mírian Helena.

Agradecimentos

Agradeço...

A Deus, por minha vida.

A meus pais Gezu Messias Ribeiro e Sebastiana de Fátima da Silva Ribeiro pela dedicação, confiança, apoio, carinho e amor incondicional em todos os momentos.

A minha irmã Mírian Cecília Ribeiro pelos conselhos, incentivos, carinho e amor durante toda esta etapa da minha vida.

À minha namorada Mírian Helena de Souza por ter me apoiado tanto durante este projeto. Você me deu muita força, tranqüilidade, paz, alegria e amor.

A meus amigos da UFU que se mostraram companheiros e, diretamente ou indiretamente, contribuíram para a realização deste trabalho, Adriano Fiad Farias, Crícia Zilda Felício, Daniel Antônio Furtado, Eduardo Ferreira Ribeiro, Éverton Hipólito de Freitas, Fabiola Souza Fernandes Pereira, Felipe Cézar de Castro Antunes, Gutierrez Soares Caixeta, Italo Tiago da Cunha, Janine Fabiana Prates Teixeira Oliveira, Jean Carlo de Souza Santos, Johnatas Teixeira de Freitas, Juliano Dalóia Carvalho, Klérisson Vinícius Ribeiro da Paixão, Liliane do Nascimento do Vale, Lucio Agostinho Rocha, Mariangela Soares Simedo, Mirella Silva Junqueira, Núbia Rosa da Silva, Nyara de Araújo Silva, Pedro Junior Ashidani, Rafael Dias Araújo, Ricardo Soares Boaventura, Ricardo Bortolatto Nunes, Ricardo José da Silva, Rodrigo Reis Pereira, Robson Silva Lopes, Stéfano Schwenck Borges Vale Vita, Tarcísio Gotto Clemente, Tauller Augusto de Araújo Matos, Valquíria Aparecida Rosa Duarte, Vinícius Borges Pires, Waldecir Pereira Junior, Walter Borges Dias pelos conselhos e apoio.

A meus amigos Rodrigo, Neilson, Leonardo, Cláudia, Walter, Irene, Cleomar pelo incentivo e paciência.

A CAPES, CNPq pelo apoio financeiro.

Principalmente à professora Sandra de Amo pelo profissionalismo, apoio, paciência, amizade e orientação em todos os momentos da realização deste trabalho.

"As invenções são, sobretudo, o resultado de um trabalho teimoso."
(Santos Dumont)

Resumo

Atualmente, o tratamento de preferências vem se tornando uma tarefa cada vez mais importante para diversos tipos de aplicações como comércio eletrônico e sistemas de buscas personalizados. Este tema tem originado muitas pesquisas tanto na área de Inteligência Artificial quando na área de Banco de Dados.

Na área de Inteligência Artificial, foram criados vários formalismos para especificação e raciocínio com preferências. Na área de Banco de Dados, os principais trabalhos desenvolvidos concentram-se em estender a linguagem SQL padrão para suportar preferências. Nesta dissertação utilizamos os formalismos para tratamento de preferências da área de Inteligência Artificial para especificar as linguagens de consulta *CPref-SQL* e *TPref-SQL*.

A linguagem de consulta *CPref-SQL* consiste de uma extensão da linguagem SQL padrão por meio de operadores dos seleção de tuplas ótimas **Best-E**, **Best-N** e **Best-D** que selecionam as tuplas de uma relação considerando um conjunto de *preferências condicionais* especificadas por um usuário. Propomos algoritmos para cada um dos operadores de seleção de tuplas ótimas e implementamos um protótipo para um fragmento da linguagem *CPref-SQL*.

A linguagem de consulta *TPref-SQL* é baseada em um modelo relacional onde os dados são representados através de *relações de seqüências*. No contexto deste *modelo relacional de seqüências*, introduzimos uma Álgebra Relacional dotada de operadores especiais para trabalhar com relações de seqüências. Os principais operadores propostos são os três operadores para seleção de seqüências ótimas **BestSeq-E**, **BestSeq-N** e **BestSeq-D** (correspondentes aos operadores **Best-E**, **Best-N** e **Best-D** propostos para a linguagem *CPref-SQL*) que consideram um conjunto de *preferências condicionais temporais*. Propomos algorimos para implementação destes operadores e implementamos um protótipo para um fragmento da linguagem *TPref-SQL*.

Palavras chave: linguagens de consulta, preferências, lógica temporal.

Abstract

Nowadays, the treatment of preferences has become an important task in several kinds of applications like e-commerce and personalized search engines. This arised a lot of research in the Artificial Intelligence field as well as in the Database field.

In the Artificial Intelligence field, several formalisms for preference specification and reasoning have been created. In the Database field, the research has been focused on extending SQL to support preference specification. In this dissertation, we use preference formalisms originated in the AI field in order to specify the query languages *CPref-SQL* and *TPref-SQL*.

The *CPref-SQL* query language consists of an extension of the standard SQL language with the operators **Best-E**, **Best-N** and **Best-D** which select the tuples of a relation taking into account a set of *conditional preferences* specified by the user. We propose algorithms for implementing these operators and implement a prototype for a fragment of the *CPref-SQL* language.

The *TPref-SQL* query language is based on a relational model where data are stored in *sequence relations*. In the context of this *sequence relational model*, we introduce a Relational Algebra with special constructors allowing to operate on sequence relations. Among these constructors we have the three operators **BestSeq-E**, **BestSeq-N** and **BestSeq-D** (the counterparts of the *CPref-SQL* operators **Best-E**, **Best-N** and **Best-D**) allowing to select the best sequences in a sequence relation, according to a set of *temporal conditional preferences*. We propose algorithms for implementing these operators and implement a prototype for a fragment of the *TPref-SQL* language.

Keywords: query languages, preferences, temporal logic.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Com o enorme aumento de informação tanto na Internet quanto em bancos de dados privados, ao realizar uma pesquisa qualquer, um usuário pode ser confrontado com uma quantidade muito grande de resultados, sendo que, muitas vezes, tais resultados encontram-se sem qualquer tipo de organização. Para solucionar este problema, pesquisas recentes têm se preocupado em atender as preferências do usuário da melhor forma possível.

Preferências estão presentes em quase tudo que é feito no dia-a-dia e também em muitas situações comerciais. Sob o ponto de vista filosófico muitos trabalhos abordando preferências já foram realizados como [Halldén 1957,Castañeda 1958,von Wright 1963,von Wright 1972,Kron e Milovanović 1975,Trapp 1985,Hansson 1996]. Entretanto nos últimos anos diversos pesquisadores têm voltado seu interesse para a especificação, representação e raciocínio com preferências na linha de pesquisa de Inteligência Artificial e para linguagens de consulta com suporte a preferências na linha de pesquisa de Banco de Dados. Tal interesse se deve principalmente à utilização de preferências cada vez mais freqüente em várias aplicações das referidas linhas de pesquisa.

Em Inteligência Artificial o uso de preferências se dá através de diversas aplicações como Sistemas de Recomendação [Nguyen e Haddaway 1999, Resnick e Varian 1997] (capazes de fazer sugestões ao usuário), Personalização de Conteúdo [Domshlak et al. 2001,Brafman et al. 2004] (onde a apresentação de conteúdos pode ser personalizada automaticamente) e aplicações médicas [Chajewska et al. 1998, Brafman e Friedman 2006] (que usam preferências para adaptar a visualização de exames clínicos para o usuário).

Um usuário pode especificar preferências de forma quantitativa ou de forma qualitativa. Na especificação de preferências de forma quantitativa o usuário deve associar a cada objeto o (ou tupla no caso de banco de dados) um número $f(o)$ chamado de *função de utilidade* ou *função de ranking* que representa seu grau de preferência em relação a este objeto [Agrawal e Wimmers 2000, Hristidis et al. 2001] (quanto menor o $f(o)$ mais preferido é o objeto o). Já na especificação de preferências de forma qualitativa o usuário informa uma ordem sobre os valores assumidos pelos atributos dos objetos. No Exemplo

1.1 são demonstradas estas duas formas de especificação de preferências.

Exemplo 1.1. A especificação de preferências de forma quantitativa e de forma qualitativa será feita com as seguintes músicas na forma de objetos:

- o_1 =("Fácil", "rock", "contemporânea");
- o_2 =("Apaixonado por você", "sertanejo", "contemporânea");
- o_3 =("Menino da porteira", "sertanejo", "antiga");
- o_4 =("Dias melhores", "rock", "antiga").

Onde o primeiro atributo é o Nome, o segundo atributo é o Gênero e o terceiro atributo é a Época. Um usuário X pode expressar suas preferências de forma quantitativa como se segue:

- $f(o_1) = 6$;
- $f(o_2) = 2$;
- $f(o_3) = 1$;
- $f(o_4) = 10$.

Um usuário Y pode expressar suas preferências de forma qualitativa através das seguintes "regras":

1. "Músicas sertanejas são melhores do que músicas de rock";
2. "Para músicas sertanejas prefiro épocas antigas a épocas contemporâneas, para músicas de rock prefiro épocas contemporâneas a épocas antigas".

É possível constatar que tanto o usuário X quanto o usuário Y expressam as mesmas preferências de formas diferentes, classificando os objetos na mesma ordem:

- o_3 é preferido a o_2 (tanto para X quanto para Y);
- o_2 é preferido a o_1 (tanto para X quanto para Y);
- o_1 é preferido a o_4 (tanto para X quanto para Y).

A classificação feita pelas preferências do usuário X se dá de maneira direta usando os valores retornados pela função $f(o_i)$ organizado-os em ordem crescente. No caso do usuário Y, a primeira regra permite inferir que os objetos o_2 e o_3 são melhores que os objetos o_1 e o_4 e a segunda regra permite inferir que o_3 é melhor do que o_2 e que o_1 é melhor do que o_4.

A especificação de preferências de forma qualitativa é considerada mais eficiente pois se a quantidade de objetos for muito grande, o que é comum em aplicações de banco de dados, será exigido um enorme esforço do usuário e um razoável consumo de tempo na

especificação de forma quantitativa por ser necessário atribuir uma nota a cada objeto. Desta forma a especificação de preferências de forma quantitativa é muitas vezes inconveniente e em certos casos impraticável, porque em muitas aplicações os usuários não estão aptos ou não possuem tempo suficiente para fornecer mais do que informações sobre preferências qualitativas [Burke 2000]. Por este motivo este trabalho está focado somente no tratamento de preferências de forma qualitativa.

Outra grande vantagem da especificação de preferências através de instruções ou regras de preferências qualitativas está na intuição natural que este tipo de especificação oferece ao usuário.

A especificação de preferências de forma qualitativa possui inerentemente a semântica *ceteris paribus*. Tal semântica foi utilizada em diversos trabalhos como [Doyle et al. 1991, Wellman e Doyle 1991, Doyle e Wellman 1994] e permite ao usuário expressar preferências sobre objetos que diferem em um atributo e possuem todos os demais atributos com valores iguais. Por exemplo, se um usuário diz que "prefere músicas sertanejas a músicas de rock", sempre que houver duas músicas com gêneros diferentes (sertanejo ou rock) e com mesma época, ritmo etc. a música sertaneja será a melhor para o usuário. É relevante notar que a semântica *ceteris paribus* não permite a comparação de dois objetos que diferem em mais de um atributo. Para tanto devem ser consideradas todas as instruções de preferências impostas pelo usuário e utilizar a propriedade de transitividade como é mostrado no Exemplo 1.2.

Exemplo 1.2. Supondo as seguintes instruções de preferências especificadas por um usuário:

1. Músicas sertanejas são preferidas a músicas de rock;
2. Músicas antigas são preferidas a músicas de contemporâneas.

E ainda três músicas idênticas exceto pelos seguintes atributos de gênero e época:

1. sertanejo, antiga;
2. sertanejo, contemporânea;
3. rock, contemporânea;

A música 1 não pode ser comparada diretamente com a música 3. Mas pela primeira instrução de preferências a música 2 é melhor do que a música 3 e pela segunda instrução de preferência a música 1 é melhor do que a música 2. Logo, a música 1 é preferida à música 3.

É interessante notar no Exemplo 1.2 que as instruções de preferências impõem, implicitamente, uma *ordem parcial estrita* sobre os objetos. Mais detalhes sobre relações de ordem serão abordados no Capítulo 2.

Outro fator relevante levado em consideração na representação de preferências é a utilização de *independência preferencial* que possibilita ao usuário exprimir a influência que um determinado atributo exerce sobre outro atributo. Um atributo X é *preferencialmente independente* de um atributo Y se e somente se para qualquer valor assumido por Y as preferências sobre o atributo X permanecem as mesmas. No exemplo das músicas o usuário pode dizer que sua preferência sobre o gênero independe da época ou ritmo da música. Neste caso, para músicas de qualquer época ou ritmo as preferências do usuário sobre o gênero não vão mudar. Analogamente, um atributo X é *condicionalmente preferencialmente dependente* de um atributo Y se para cada valor assumido por Y as preferências sobre X são alteradas. No exemplo das músicas o usuário pode dizer que prefere época antiga a época contemporânea se o gênero da música for sertanejo e prefere época contemporânea a época antiga se o gênero da música for rock. Assim o atributo época depende do atributo gênero, ou seja, o atributo época é condicionalmente dependente do atributo gênero.

Em Banco de Dados, os principais trabalhos concentram-se na incorporaração de preferências em linguagens de consultas e desenvolvimento de algoritmos para resolver consultas considerando preferências [Kießling e Köstler 2002, Kießling et al. 2004, Chan et al. 2005, Koutrika et al. 2006, Preisinger et al. 2006, Preisinger e Kießling 2007, Endres e Kießling 2008, Stefanidis e Pitoura 2008]. A criação de novas linguagens de consulta com suporte a preferências permitem o aprimoramento de aplicações como máquinas de busca e comércio eletrônico e diversos tipos de aplicações de banco de dados que envolvem personalização [Ioannidis e Koutrika 2005]. O Exemplo 1.3 demonstra como as preferências podem ser declaradas através de uma linguagem de consulta.

Exemplo 1.3. Supondo uma relação MUSICAS sobre um esquema com os atributos N (para Nome), G (para Gênero), D (para Duração) e E (para Época). E que um usuário expresse as seguintes preferências:

1. Músicas sertanejas (s) são melhores do que músicas de rock (r);
2. Para músicas sertanejas prefiro durações longas (l) a durações breves (b), mas para músicas de rock prefiro o contrário.

O usuário quer saber o nome das músicas presentes no banco de dados que melhor atendam suas preferências, mas "impõe" que somente as músicas antigas (a) devem ser retornadas. Desta forma a seguinte consulta pode ser realizada:

```
SELECT N FROM MUSICAS
WHERE E = a
ACCORDING TO PREFERENCES
(G = s) > (G = r) AND
IF G = s THEN (D = l) > (D = b) AND
IF G = r THEN (D = b) > (D = l)
```

Na consulta do Exemplo 1.3 as preferências especificadas pelo usuário são declaradas através da cláusula ACCORDING TO PREFERENCES e a imposição de que as músicas devem ser antigas é feita através da cláusula WHERE.

Assim como na linguagem SQL (Structured Query Language) padrão as restrições da cláusula WHERE devem ser obrigatoriamente atendidas e por este motivo são chamadas de *restrições hard*. As preferências por sua vez são consideradas *restrições soft* porque expressam um desejo que pode ser atendido (parcialmente) ou não.

O principal objeto deste trabalho é estender a linguagem de consulta SQL para que regras de preferências possam ser declaradas e resolvidas através de consultas. Desta forma, será construída uma *ponte* para interligar a área de linguagens de consultas em Banco de Dados com a área de raciocínio com preferências em Inteligência Artificial.

1.1 Contribuições

As principais contribuições deste trabalho são divididas em duas partes:

- Parte I

    1. Extensão da álgebra relacional convencional através dos operadores **Best-E**, **Best-N** e **Best-D** capazes de selecionar tuplas considerando um conjunto de preferências condicionais (cada um dos operadores possui uma semântica diferente);
    2. No caso do operador **Best-D** é proposto o conceito de listas de escopos para realizar teste de dominância (comparação) entre tuplas;
    3. Além das listas de escopos, propomos uma otimização para a construção da estrutura de grafo utilizada para obtê-las;
    4. Proposta dos algoritmos **GetBest-E**, **GetBest-N** e **GetBest-D** para resolver os operadores de seleção de tuplas ótimas **Best-E**, **Best-N** e **Best-D**, respectivamente;
    5. Especificação da linguagem de consulta *CPref-SQL* capaz de resolver consultas contendo preferências condicionais;
    6. Implementação de um protótipo contendo os algoritmos **GetBest-E** e **GetBest-N**.

- Parte II

    1. Especificação da álgebra relacional *TPrefAR* com diversos operadores próprios para trabalhar com banco de dados de seqüências e operadores capazes de selecionar e construir seqüências ótimas considerando um conjunto de preferências condicionais temporais;

2. Especificação da linguagem de consulta *TPref-SQL* baseada nos operadores da álgebra *TPrefAR*;
3. Proposta dos algoritmos **GetBestSeqs-E**, **GetBestSeqs-N** e **GetBestSeqs-D** para resolver os operadores de seleção de seqüências ótimas **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**, respectivamente;
4. Implementação de um protótipo contendo os algoritmos **GetBestSeqs-E**, **GetBestSeqs-N**, **BuildSeqs-E** e **BuildSeqs-N**.

1.2 Organização da dissertação

Esta dissertação está dividida em duas partes. A primeira parte está organizada como se segue.

No Capítulo 2 serão apresentados alguns conceitos básicos sobre preferências, juntamente com os principais trabalhos relacionados a representação e raciocínio sobre preferências e incorporaração de preferências em linguagem de consultas para banco de dados.

O Capítulo 3 propõe os novos operadores **Best-E**, **Best-N** e **Best-D** para a álgebra relacional capazes de selecionar tuplas ótimas considerando um conjunto de preferências condicionais.

A linguagem de consulta *CPref-SQL* capaz de resolver consultas contendo preferências condicionais é especificada no Capítulo 4.

No Capítulo 5 é descrito como foi implementado um protótipo para linguagem *CPref-SQL*.

A segunda parte está organizada da seguinte maneira.

O Capítulo 6 apresenta alguns conceitos básicos relacionados a preferências sobre estrutura mais complexas como conjunto de objetos e seqüências de objetos. Sendo que as preferências sobre seqüências de objetos são abordadas mais detalhadamente através do formalismo lógico *TPref*.

No Capítulo 7 é proposta a **Álgebra TPrefAR** com diversos operadores próprios para trabalhar com banco de dados de seqüências e operadores capazes de selecionar e construir seqüências ótimas considerando um conjunto de preferências condicionais temporais.

A linguagem de consulta *TPref-SQL* capaz de resolver consultas contendo preferências condicionais temporais é especificada no Capítulo 8.

O Capítulo 9 descreve como foi feita a implementação de um protótipo com alguns fragmentos da linguagem *TPref-SQL*.

Finalmente, no Capítulo 10 serão apresentadas conclusões e perspectivas para trabalhos futuros.

Parte I

Preferências sobre Objetos

Capítulo 2

Fundamentos Teóricos e Trabalhos Correlatos

Neste capítulo são destacados alguns fundamentos teóricos importantes para a representação e raciocínio sobre preferências. Além disto serão destacados diversos trabalhos já realizados nas áreas de obtenção, representação, raciocínio e aplicações envolvendo preferências, bem como a utilização de preferências no contexto de banco de dados.

Na Seção 2.1 introduzimos uma de noção preferências como relações de ordem sobre um conjunto de objetos (ou tuplas). Em seguida na Seção 2.2 mostramos como realizar operações sobre preferências. Na Seção 2.3 apresentamos trabalhos envolvendo linguagens de consulta para banco de dados que tratam preferências. Já na Seção 2.4 apresentamos trabalhos sobre o tratamento de preferências na área de Inteligência Artificial. Na Seção 2.5 discutimos sobre como adaptar o tratamento de preferências de Inteligência Artificial no contexto de Banco de Dados. E, por fim, na Seção 2.6 expomos as considerações finais sobre este capítulo.

2.1 Preferências como Relações de Ordem

Quando um usuário expressa suas preferências sobre um conjunto de elementos (sejam estes elementos objetos, conjunto de objetos ou seqüências de objetos), mesmo que implicitamente, existe uma relação de ordem impostas pelas preferências entre os elementos do conjunto. Sendo assim, é fundamental entender as possíveis relações de ordem que podem estar ligadas às preferências.

Uma relação de ordem entre os elementos de um conjunto pode ser induzida através de certas medidas (como distância, freqüência, etc.), mas neste trabalho as relações de ordem estão diretamente ligadas às preferências entre objetos (a é preferido a b implica em $a > b$).

O estabelecimento de uma relação de ordem entre os elementos de um conjunto é extre-

mamente importante para a resolução de problemas ligados a dominância dos elementos, ou seja, dados dois elementos, descobrir qual destes elemento é o mais preferido [Öztürk et al. 2006]. Antes que sejam definidas as possíveis relações de ordem é necessário conhecer o que é uma relação binária, veja a Definição 2.1.

Definição 2.1. (Relação binária) Seja A um conjunto finito de elementos $(a, b, c, ..., n)$, uma *relação binária* R sobre o conjunto A é um subconjunto do produto cartesiano $A \times A$, isto é, um conjunto de pares ordenados (a, b) tal que a e b pertencem a A: $R \subseteq A \times A$. Para denotar que um par ordenado (a, b) pertence a uma relação R usa-se a notação aRb.

Uma Relação Binária pode ser classificada em:

- reflexiva, se $\forall a \in A$, aRa ;
- irreflexiva, se $\forall a \in A$, $\neg(aRa)$;
- simétrica, se $\forall a, b \in A$, $aRb \longrightarrow bRa$;
- anti-simétrica, se $\forall a, b \in A$, $(aRb, bRa) \longrightarrow a = b$;
- completa, se $\forall a, b \in A$, com $a \neq b$, (aRb ou bRa);
- fortemente completa, se $\forall a, b \in A$, aRb ou bRa;
- transitiva, se $\forall a, b, c \in A$, $(aRb, bRc) \longrightarrow aRc$;
- negativamente transitiva, se $\forall a, b, c \in A$, $\neg(aRb) \wedge \neg(bRc) \longrightarrow \neg(aRc)$;
- semi-transitiva, se $\forall a, b, c, d \in A$, $(aRb, bRc) \longrightarrow$ (aRd ou dRc);

As relações de ordem ou simplesmente *ordens* são tipos específicos de relações binárias que podem caracterizar certas preferências, como por exemplo a ordem total da Definição 2.2 que permite a comparação entre todos os elementos de um conjunto.

Definição 2.2. (Ordem total) Seja R uma relação binária sobre o conjunto A. R é uma *ordem total* se e somente se R é reflexiva, anti-simétrica, completa e transitiva.

Como exemplo de ordem total vamos considerar o conjunto de objetos $Z = \{o_1, o_2, o_3, o_4\}$ e as seguintes relações entre os objetos:

- $o_1 > o_2$;
- $o_2 > o_3$;
- $o_3 > o_4$.

Como se trata de uma ordem total é possível fazer qualquer comparação entre os objetos de Z dado que uma ordem total é completa. Para tratar indiferença, ou seja, quando dois objetos possuem o mesmo grau de preferência, podemos utilizar a pré-ordem dada pela Definição 2.3.

Definição 2.3. (Pré-ordem) Seja R uma relação binária sobre o conjunto A. R é uma *pré-ordem* se e somente se R é reflexiva, fortemente completa e transitiva.

Como exemplo de pré-ordem pode ser considerado o conjunto de objetos Z anterior e as seguintes relações entre os objetos:

- $o_1 > o_2$;
- $o_2 > o_1$;
- $o_2 > o_3$;
- $o_3 > o_4$.

Neste caso os objetos o_1 e o_2 são indiferentes, isto pode ser denotado por $o_1 \sim o_2$.

As relações de ordem definidas anteriormente não tratam a possibilidade de dois elementos serem incomparáveis, ou seja, o é incomparável a o' quando $o \not> o'$ e $o' \not> o$. Entretanto, quando se trabalha com preferências em diversas situações a incomparabilidade deve ser considerada, para isto podemos usar uma ordem parcial, veja a Definição 2.4.

Definição 2.4. (Ordem parcial estrita) Seja R uma relação binária sobre o conjunto A. R é uma *ordem parcial estrita* se e somente se R é irreflexiva, anti-simétrica e transitiva.

Como exemplo de ordem parcial estrita vamos considerar novamente o conjunto de objetos Z anterior e as seguintes relações entre os objetos:

- $o_1 > o_2$;
- $o_2 > o_4$;
- $o_3 > o_4$.

Neste caso os objetos o_1 e o_3 são incomparáveis entre si (o mesmo ocorre com o_2 e o_3).

Existem casos particulares de ordens parciais que são equivalentes a preferências quantitativa representadas através de funções de *ranking*, estas ordens são denominadas ordens parciais estritas fracas [Fishburn 1999], veja a Definição 2.5.

Definição 2.5. (Ordem parcial estrita fraca) Seja R uma relação binária sobre o conjunto A. R é uma *ordem parcial estrita fraca* se e somente se R é irreflexiva, anti-simétrica, transitiva e negativamente transitiva.

Como exemplo de ordem fraca pode ser considerada a relação $o > o' \Leftrightarrow f(o) < f(o')$, o conjunto de objetos Z anterior e as seguintes funções de *ranking*:

- $f(o_1) = 4$;
- $f(o_2) = 3$;
- $f(o_3) = 2$;
- $f(o_4) = 1$.

Neste caso os objetos podem ser comparados diretamente através da função f. Entretanto preferências especificadas de forma qualitativa são mais gerais podendo produzir ordens parciais não fracas. Isto ocorre no exemplo de ordem parcial estrita, podemos notar que as preferências não atendem a propriedade de transitividade negativa ($\nexists o_1 > o_3$ e $\nexists o_3 > o_2$ é verdade, mas $\nexists o_1 > o_2$ não é verdade).

A demonstração de que ordens parciais estritas fracas são equivalentes a preferências quantitativas é dada pelos Teoremas 2.1 e 2.2. Antes de enunciar estes teoremas são demonstrados alguns resultados referentes a ordens parciais estritas fracas.

Definição 2.6. (relação de equivalência ($\equiv$)) Seja A um conjunto e $<$ uma relação de ordem parcial estrita fraca sobre A. A *relação de equivalência* $a_1 \equiv a_2$ acontece se $\neg(a_1 < a_2)$ e $\neg(a_2 < a_1)$.

Lema 2.1. *Seja A um conjunto e $<$ uma relação de ordem parcial estrita fraca sobre A e $\equiv$ a relação de equivalência da Definição 2.6. Se $a < c$, $a \equiv b$ e $c \equiv d$ então $b < d$.*

Prova. Supondo por absurdo que $\neg(b < d)$. Então temos $\neg(a < b)$, $\neg(b < d)$ e $\neg(d < c)$. Pelo fato de $<$ ser ordem fraca, pode-se concluir que $\neg(a < c)$, o que é um absurdo, pois por hipótese $a < c$.

Definição 2.7. (Função l) Seja A um conjunto e $<$ uma relação de ordem parcial estrita fraca sobre A. A *função* l é definida por $l : A \to \mathbb{N}$ da seguinte maneira indutiva:

- $i = 0$: Define-se que $B_0 = A$ e $A_0 = \{a \in B_0 \mid \nexists b \in B_0$ tal que $a < b\}$. Para todo $a \in A_0$ define-se $l(a) = 0$.
- $i \geq 1$: Define-se que $B_i = B_{i-1} - A_{i-1}$ e $A_i = \{a \in B_i \mid \nexists b \in B_i$ tal que $a < b\}$. Para todo $a \in A_i$ define-se que $l(a) = i$.

Podemos notar que:

- $B_i \subseteq B_{i-1}$ para todo $i \geq 0$.
- $A_i \subseteq B_i$ para todo $i \geq 0$.
- $A = \bigcup_{i=1}^{\infty} A_i$
- $A_i \cap A_j = \emptyset$ para todo i, j tal que $i \neq j$.
- $A_i \cup B_{i+1} = B_i$ para todo $i \geq 0$.

Lema 2.2. *Seja A um conjunto e $<$ uma relação de ordem parcial estrita fraca sobre A e l definida de acordo com a definição 2.7. Então $l(a) = l(b)$ se e somente se $a \equiv b$.*

Prova. Supondo que $l(a) = l(b) = i$. Então $a, b \in A_i$. Logo, não existe $c \in B_i$ tal que $c > a$. Portanto $\neg(b > a)$ (já que $A_i \subseteq B_i$ e $b \in A_i$). De forma análoga provamos que $\neg(a > b)$. Portanto, $a \equiv b$. Reciprocamente, supondo que $a \equiv b$. Supondo que $l(a) > l(b)$, $b \in A_j$, $a \in A_i$, $i > j$ é possível mostrar que existe $b' \in A_j$ tal que $a > b'$.

Como $l(b') = l(b)$, podemos concluir (pela recíproca já demonstrada) que $b' \equiv b$. Como por hipótese $a \equiv b$, então concluímos que $a \equiv b'$. Logo, $\neg(a > b')$. Absurdo.

Finalmente, serão passados os Teoremas 2.1 e 2.2 que mostram que ordens parciais estritas fracas são equivalentes a uma ordenação gerada por uma função de *ranking* e vice-versa, isto é, ordens dadas por uma função de *ranking* são necessariamente ordens parciais estritas fracas.

Teorema 2.1. *Seja A um conjunto e $<$ uma relação de ordem parcial estrita fraca sobre A e l definida de acordo com a definição 2.7. Então $a < b$ se e somente se $l(a) > l(b)$.*

Prova. Suponha que $b > a$. Então, pelo lema 2.2, temos que $l(a) \neq l(b)$. Suponha que $l(a) < l(b)$. Então $a \in A_i$, $b \in A_j$, $i < j$. Pode-se construir um $a' \equiv a$ tal que $a' > b$. Pelo lema 2.1, tem-se que $a > b$. Absurdo. Portanto, $l(a) > l(b)$. Reciprocamente, suponha que $l(a) > l(b)$. Então $a \in A_i$, $b \in A_j$, $j < i$. Pode-se construir um $b' \equiv b$ tal que $b' > a$. Pelo lema 2.1, tem-se que $b > a$.

Teorema 2.2. *Seja A um conjunto e f uma função $f : A \to \mathbb{N}$. Define-se a relação $<$ sobre A da seguinte maneira: $a < b$ se e somente se $f(a) > f(b)$. A relação $<$ é uma ordem parcial estrita fraca.*

Prova. De fato:

- Irreflexividade: Seja $a \in A$. Como $f(a) = f(a)$ então $\neg(a < a)$.
- Anti-simetria: Sejam $a, b \in A$ onde $a < b$. Então $f(a) > f(b)$. Logo $f(b) \not> f(a)$ e portanto $\neg(b < a)$.
- Transitividade: Sejam $a, b, c \in A$ tais que $a < b$ e $b < c$. Então $f(a) > f(b)$ e $f(b) > f(c)$. Logo $f(a) > f(c)$ e portanto, por definição: $a < c$.
- Transitividade negativa: Sejam $a, b, c \in A$ tais que $\neg(a < b)$ e $\neg(b < c)$. É necessário mostrar que $\neg(a < c)$. Como $\neg(a < b)$ e $\neg(b < c)$ então $f(a) \leq f(b)$ e $f(b) \leq f(c)$. Logo, $f(a) \leq f(c)$. Portanto não é possível que $(a < c)$ (pois isto só ocorreria se $f(a) > f(c)$. Portanto tem-se $\neg(a < c)$.

2.2 Operações sobre Preferências

As preferências representadas através das relações de ordem explicadas na seção anterior podem ser combinadas através de certas operações para a construir preferências compostas [Andréka et al. 2002]. Nesta seção serão estudadas as principais operações sobre relações de ordem e a influência de tais operações sobre as propriedades das relações de ordem.

As principais operações sobre para composição de relações de ordem são união, interseção, priorização e composição *Pareto*. As operações de união de relações de ordem dada pela Definição 2.8 e interseção de relações de ordem dada pela Definição 2.9 são análogas às operações de união e interseção de conjuntos, respectivamente.

Definição 2.8. (União de preferências) Dadas duas relações de ordem $>_{P_1}$ e $>_{P_2}$. A operação de *união de preferências* $>_{P_{1,2}} = >_{P_1} \cup >_{P_2}$ é definida da seguinte maneira: $o_1 >_{P_{1,2}} o_2$ se e somente se $o_1 >_{P_1} o_2 \vee o_1 >_{P_2} o_2$

Definição 2.9. (Interseção de preferências) Dadas duas relações de ordem $>_{P_1}$ e $>_{P_2}$. A operação de *interseção de preferências* $>_{P_{1,2}} = >_{P_1} \cap >_{P_2}$ é definida da seguinte maneira: $o_1 >_{P_{1,2}} o_2$ se e somente se $o_1 >_{P_1} o_2 \wedge o_1 >_{P_2} o_2$

É importante destacar a questão da preservação das propriedades das relações de ordem após as operações de união e interseção, a Proposição 2.1 diz respeito a ordens parciais estritas e a Proposição 2.2 diz respeito a ordens fracas.

Proposição 2.1. ([Chomicki 2003]) *Ordem parciais estritas são preservadas por interseção mas não por união.*

Proposição 2.2. ([Chomicki 2003]) *Ordem fracas não são preservadas nem por interseção e nem por união.*

A composição de relações de ordem pode também dar mais prioridade a determinada relação de ordem, isto é feito através da operação de priorização dada pela Definição

Definição 2.10. (Priorização de preferências) Dadas duas relações de ordem $>_{P_1}$ e $>_{P_2}$. A operação de *priorização de preferências* $>_{P_{1,2}} = >_{P_1} \rhd >_{P_2}$ é definida da seguinte maneira: $o_1 >_{P_{1,2}} o_2$ se e somente se $o_1 >_{P_1} o_2 \ \vee \ (o_1 \sim_{P_1} o_2 \ \wedge \ o_1 >_{P_2} o_2)$[1]

As operações de união, interseção e priorização são operações de composição unidimensionais e são utilizadas para construir relações de ordem compostas a partir de relações de ordem sobre um mesmo conjunto de atributos. Quando existem relações de ordem sobre conjuntos de atributos distintos deve ser utilizada uma operação composição multidimensional. A principal operação multidimensional é a *composição Pareto*, dada pela Definição 2.11.

Definição 2.11. (Composição Pareto) Dados dois conjuntos de atributos R_1 e R_2 e as relações de ordem $>_{P_1}$ sobre R_1 e $>_{P_2}$ sobre R_2. A operação *composição Pareto* $>_{P_{1,2}} = >_{P_1} \otimes >_{P_2}$ sobre o conjunto $R_1 \times R_2$ é definida da seguinte maneira: $(a, b) >_{P_{1,2}} (a', b')$ se e somente se $a \geq_{P_1} a' \wedge b \geq_{P_2} b' \wedge (a >_{P_1} a' \vee b >_{P_2} b')$

Vale ressaltar que a composição *Pareto* de ordens fracas sempre gera uma ordem parcial estrita como determina a Proposição 2.3.

Proposição 2.3. ([Chomicki 2003]) *A composição Pareto de ordens fracas resulta em uma ordem parcial estrita.*

[1] O sinal $\sim$ indica indiferença, ou seja, $o \sim o'$ significa que nem o é preferido a o' e nem o' é preferido a o.

Existem certas relações de ordem que são transitivas por definição como é o caso das ordens fracas, em muitas situações é necessário encontrar o *fecho transitivo* de uma relação de ordem para que esta possa ser usada efetivamente em comparações de objetos. Por exemplo, a relação de ordem sobre o conjunto de objetos $Z = \{o_1, o_2, o_3\}$ definida da seguinte maneira:

- $o_1 > o_2$;
- $o_2 > o_3$.

Intuitivamente, sabe-se que $o_1 > o_3$ mas a relação de ordem não possui esta informação explicitamente. Para isto é dada a Definição 2.12.

Definição 2.12. (Fecho transitivo) O *fecho transitivo* de uma relação de ordem $>_P$ é uma relação de ordem $>_{P^*}$ definido da seguinte maneira: $o_1 >_{P^*} o_2$ se e somente se $o_1 >_P^n o_2$ para algum $n > 0$, onde $o_1 >_P^1 o_2 \equiv o_1 >_P o_2$ e $o_1 >_P^{n+1} o_2 \equiv o_1 >_P o_3 \wedge o_3 >_P^n o_2$.

2.3 Preferências em Bancos de Dados

Na área de banco de dados os principais trabalhos no sentido de resolver consultas com a utilização de preferências são [Kießling e Köstler 2002, Kießling 2002] onde foi desenvolvida a linguagem de consulta *Preference SQL* e [Chomicki 2003] que propôs o operador relacional *winnow*.

2.3.1 Preference SQL

A linguagem de consulta *Preference SQL* é uma extensão da linguagem SQL padrão que permite a realização de consultas contendo preferências. As preferências contidas nas consultas são especificadas por meio de um conjunto de operadores definidos através de uma *álgebra de preferências*, basicamente cada operador impõe uma ordem fraca sobre o domínio de um atributo e a combinação dos operadores é feita com a operação de composição *Pareto*.

As consultas da linguagem *Preference SQL* são resolvidas utilizando o modelo *Best Matches Only (BMO)* que consiste dos seguintes passos:

- Busca-se as tuplas da relação que atendem exatamente às preferências;
- Se nenhuma tupla que atende exatamente as preferências for encontrada, buscam-se as melhores tuplas possíveis, ou seja, aquelas tuplas que mais se aproximam das preferências.

É importante frisar que não existe a necessidade de calcular o fecho transitivo da relação de ordem imposta pelas preferências de uma consulta *Preference SQL*, isto porque

as preferências são expressas através de ordens fracas. Como vimos na seção anterior, as ordens fracas estão ligadas a funções de *ranking*, desta forma as consultas são resolvidas através do cálculo de funções de *ranking* e da utilização do conceito de composição *Pareto*. Por meio do Exemplo 2.1 é ilustrado como é resolvida uma consulta *Preference SQL*.

Exemplo 2.1. Neste exemplo vamos considerar a relação m1 sobre o esquema M1(NOME, GENERO, DURACAO) apresentada na Figura 2.1(a) e seguinte consulta *Preference SQL*:

```
SELECT *
FROM m1
PREFERRING (GENERO = 'sertanejo' ELSE GENERO = 'rock')
AND DURACAO = 'longa'
```

As preferências são especificadas utilizando a cláusula *PREFERRING*. Nesta consulta há duas preferências combinadas através de uma composição *Pareto* (feita com o conectivo *AND*). A primeira preferência diz que o melhor valor para GENERO é 'sertanejo' e que o segundo melhor valor é 'rock'. A segunda preferência diz que o melhor valor para DURACAO é 'longa'. Com estas preferências são calculadas as funções de *ranking* $f(G)$ para GENERO e $f(D)$ para DURACAO, quanto mais preferido é o valor, menor é o número dado pela função de *ranking*, conforme mostra a Figura 2.1(b). As tuplas ótimas de acordo com as preferências devem possuir $f(G) = 1$ e $f(D) = 1$, como estas tuplas não existem em M1, são retornadas as tuplas t_2 e t_6 que são as tuplas que mais se aproximam das preferências e para as quais não existem nenhuma outra melhor em M1.

	NOME	GENERO	DURACAO
t_1	m_1	clássico	longa
t_2	m_2	sertanejo	breve
t_3	m_3	rock	breve
t_4	m_4	pop	longa
t_5	m_5	pop	breve
t_6	m_6	rock	longa

(a) Relação m1

	NOME	GENERO	DURACAO	$f(G)$	$f(D)$
t_1	m_1	clássico	longa	3	1
t_2	m_2	sertanejo	breve	1	2
t_3	m_3	rock	breve	2	2
t_4	m_4	pop	longa	3	1
t_5	m_5	pop	breve	3	2
t_6	m_6	rock	longa	2	1

(b) Relação m1 com funções de *ranking*

Figura 2.1: Relação m1 e funções de *ranking* associadas aos atributos

2.3.2 O Operador Winnow

No trabalho de [Chomicki 2003], as preferências são expressas através de fórmulas da lógica de primeira ordem sobre os atributos de uma relação, tais fórmulas são denominadas fórmulas de preferência. Considerando uma relação m sobre o esquema M(N, G, R), onde N, G e R representam os atributos nome, gênero e ritmo, respectivamente, pode ser especificada uma fórmula de preferência P_1 da seguinte maneira:

$$(N, G, R) > (N', G', R') \equiv G = s \wedge G' = s \wedge R = m \wedge R' = t,$$

onde s, m e t equivalem aos valores sertanejo, movimentado e tranqüilo. Esta fórmula de preferências indica que as músicas sertanejas e movimentadas são preferidas as músicas sertanejas e tranqüilas.

Em certas fórmulas de preferência também é necessário calcular o fecho transitivo de maneira semelhante à Definição 2.12, sendo que o fecho transitivo de uma fórmula de preferência também é uma fórmula de preferência.

O operador *winnow* (w) recebe uma fórmula de preferência P, uma relação r e retorna as tuplas de r que não são dominadas por nenhuma outra tupla de r segundo as preferências P, ou seja,

$$w_P(r) = \{t \in r \mid \nexists t' \in r, t' >_P t\}.$$

Quando uma tupla t_1 domina uma tupla t_2 é porque t_1 é preferida a t_2. Para exemplificar o operador *winnow*, pode ser considerada a fórmula de preferência P_1 e a relação m2 sobre o esquema M2(N, G, R) apresentada na Figura 2.2. Neste caso são retornadas as tuplas t_3 e t_4 pois nenhuma outra tupla da relação é preferida a t_3 ou t_4 de acordo com P_1.

	N	G	R
t_1	As Águas do São Francisco	s	t
t_2	Menino da Porteira	s	t
t_3	Panela Velha	s	m
t_4	Dias Melhores	r	t

Figura 2.2: Relação m2

O operador *winnow* não acrescenta nenhum poder de expressão adicional a álgebra relacional conforme determina o Teorema 2.3.

Teorema 2.3. ([Chomicki 2003]) *O poder de expressão da álgebra relacional padrão não muda se o operador de diferença for substituído pelo operador winnow*

2.4 Preferências em Inteligência Artificial

Nesta seção serão apresentados os principais formalismos para o tratamento de preferências sobre objetos em Inteligência Artificial. São eles CP-nets, TCP-nets e a linguagem de preferências condicionais.

2.4.1 CP-nets e TCP-nets

CP-nets é um formalismo para representação de preferências qualitativas sobre objetos que tem sido alvo de diversas pesquisa na área de Inteligência Artificial [Boutilier et al. 1999, Boutilier et al. 2004a, Boutilier et al. 2004b]. A principal vantagem da representação de instruções de preferências qualitativas por meio de CP-nets está na forma

compacta com que as preferências são reproduzidas sem perder a semântica *ceteris paribus* e permitindo o uso de dependência preferencial condicional.

Uma CP-net é um grafo dirigido sobre um conjunto de atributos onde cada nó representa um atributo e as arestas representam as dependências preferenciais condicionais. Quando há uma aresta do atributo X para o atributo Y significa que as preferências sobre o atributo Y são condicionadas pelo valor assumido pelo atributo X, além disto X é chamado de *atributo pai* em relação ao atributo Y que por sua vez é chamado de *atributo filho* ou *atributo dependente*. Cada atributo X possui vinculada ao seu respectivo nó da CP-net uma *tabela de preferência condicional* (TPC) que descreve as preferências sobre o atributo X levando em conta a combinação dos possíveis valores assumidos pelos atributos pais de X. No Exemplo 2.2 são expostas algumas preferências e criada uma CP-net para representá-las.

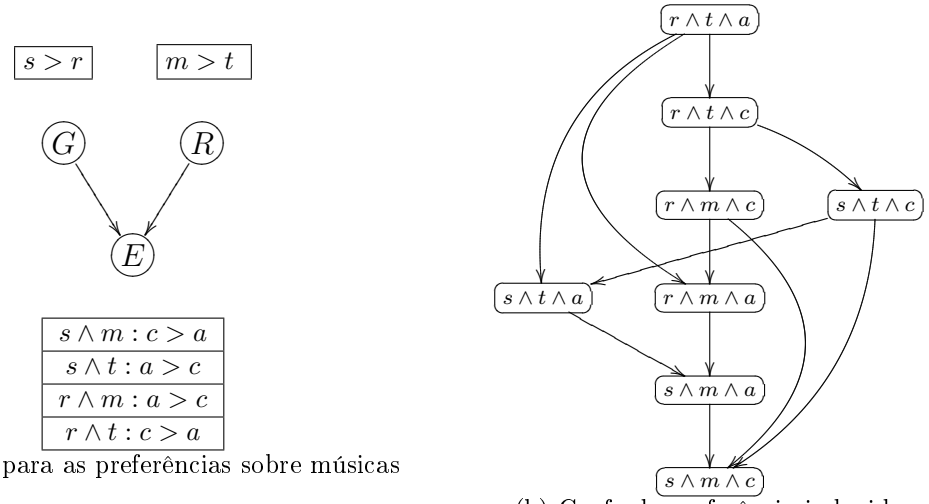
Exemplo 2.2. Supondo as seguintes preferências especificadas por um usuário sobre músicas:

1. Músicas cujo gênero (G) é sertanejo (s) são melhores do que músicas cujo gênero é rock (r);
2. Músicas com ritmo (R) movimentado (m) são preferidas a músicas tranqüilas (t);
3. Para músicas sertanejas tranqüilas ou músicas de rock movimentadas épocas (E) antigas (a) são preferidas a épocas contemporâneas (c), caso contrário épocas contemporâneas são preferidas a épocas antigas.

A Figura 2.3(a) exibe uma CP-net para as preferências especificadas enquanto a Figura 2.3(b) exibe um *grafo de preferência* induzido pela CP-net. O grafo de preferência possui uma aresta de o_i para o_j quando o objeto o_j é preferido ao objeto o_i. Todas as arestas do grafo de preferência são conseqüências da CP-net, por exemplo a aresta de $s \wedge a \wedge n$ para $s \wedge a \wedge c$ é uma conseqüência direta da TPC sobre o atributo época.

Em diversos trabalhos como [Boutilier et al. 1999,Boutilier et al. 2004a,Boutilier et al. 2004b] foram propostos algoritmos que exploram a estrutura das CP-nets para tarefas de otimização e testes de dominância. O trabalho de [Boutilier et al. 1999] propôs um algoritmo de otimização que obtém o objeto ótimo ou o objeto mais preferido, caminhando através da estrutura da CP-net e tomando os melhores valores para os atributos de acordo com suas TPC's. Os primeiros atributos considerados são os atributos *independentes*, ou seja, aqueles que não possuem pais e o caminho percorrido pelo algoritmo é sempre partindo dos atributos pais para os atributos filhos. O Exemplo 2.3 mostra como o algoritmo de otimização é executado sobre CP-net da Figura 2.3(a).

Exemplo 2.3. A execução do algoritmo de otimização inicia pelos atributos independentes, na CP-net da Figura 2.3(a) estes atributos são G e R, considerando suas respectivas TPC's serão tomados os valores $G = s$ e $R = m$, este passo pode ser ilustrado pela Figura

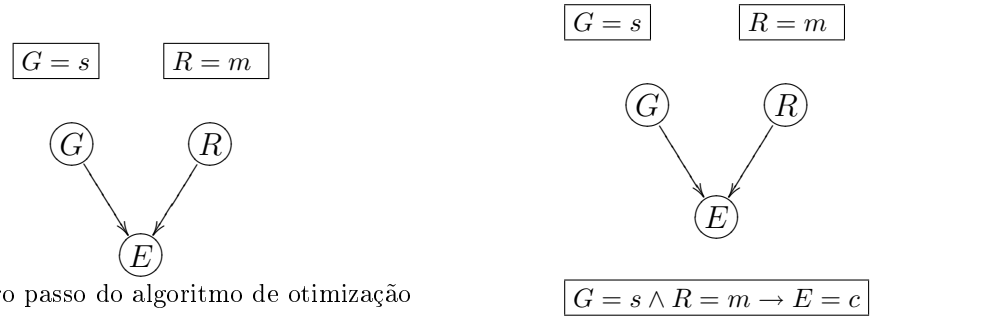


(a) CP-net para as preferências sobre músicas

(b) Grafo de preferência induzido

Figura 2.3: CP-net e seu grafo de preferência induzido

2.4(a). Caminhando pela CP-net dos atributos pais para os atributo filhos chegamos ao atributo E, de acordo com a TPC deste atributo e os valores já tomados para seus pais $E = c$, conforme é exibido na Figura 2.4(b).



(a) Primeiro passo do algoritmo de otimização

(b) Segundo passo do algoritmo de otimização

Figura 2.4: Execução do algoritmo de otimização sobre CP-nets

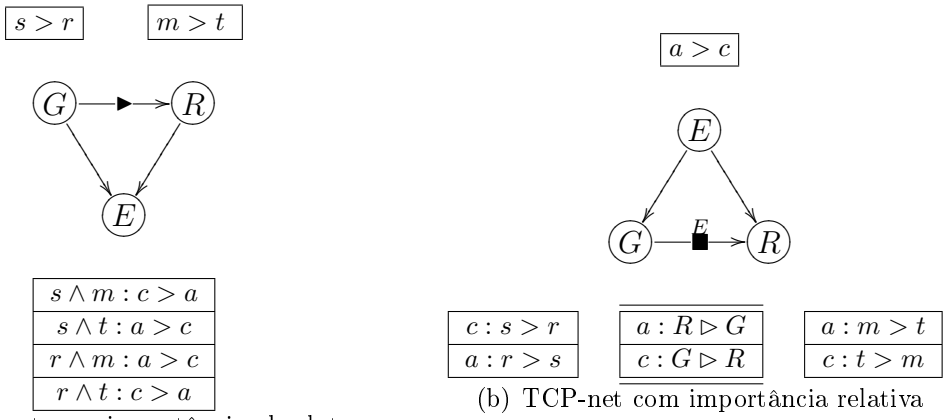
O algoritmo executado no Exemplo 2.3 é o algoritmo de otimização básico, no trabalho de [Boutilier et al. 2004b] tal algoritmo foi estendido para tratar restrições.

Os algoritmos de teste de dominância são utilizados para verificar se um objeto o_i domina um objeto o_j considerando uma certa CP-net, ou seja, são utilizados para verificar se o_i é preferido a o_j. Algumas formas de implementação de tais algoritmos são equivalentes a *algoritmos de planejamento* em Inteligência Artificial que vêm sendo alvo de várias pesquisas como [Bylander 1994, Bäckström e Nebel 1995, Domshlak e Brafman 2002b, Shimony e Domshlak 2003].

O formalismo *TCP-nets* é uma extensão do formalismo *CP-nets* que permite a representação de *importância absoluta ou relativa*, e com isto, em certos casos é possível fazer comparações diretas entre objetos que diferem em mais de um atributo [Brafman e Domshlak 2002, Brafman et al. 2006a].

Importância absoluta é quando, dados dois atributos X e Y preferencialmente independentes, X é mais importante do que Y, denotado por $X \rhd Y$. Isto faz com que as preferências sobre os valores de X sobreponham as preferências sobre os valores de Y. Já na importância relativa, um atributo é mais importante do que outro de acordo com os valores de um terceiro atributo ou um conjunto de atributos. Isto é, dados três atributos X, Y e Z, X é mais importante do que Y dependendo do valor assumido por Z.

Uma *TCP-net* é um grafo que possui as mesmas características de uma *CP-net* e ainda possui mais dois tipos de arestas, uma para representar importância absoluta e outra para representar importância relativa. Uma aresta de X para Y contendo o símbolo ▶ indica que X é mais importante do que Y de forma absoluta. Uma aresta de importância relativa é uma aresta não direcionada rotulada pelos atributos que condicionam a importância relativa, esta aresta contém o símbolo ■ e possui também uma *tabela de importância condicional* (TIC) para definir a importância relativa. A TIC descreve a importância relativa de acordo com os atributos que condicionam a importância. O Exemplo 2.4 apresenta duas *TCP-nets* que ilustram os dois tipos de importância.



(a) TCP-net com importância absoluta

(b) TCP-net com importância relativa

Figura 2.5: TCP-nets com importância absoluta e relativa

Exemplo 2.4. Para demonstrar a importância absoluta serão consideradas as mesmas preferências do Exemplo 2.2 acrescidas da preferência "O gênero da música é mais importante do que o ritmo". A TCP-net da Figura 2.5(a) corresponde a estas preferências, sendo que a importância absoluta é representada pela aresta de G para R contendo o símbolo ▶. Já a TCP-net da Figura 2.5(b) corresponde as seguintes preferências:

1. Época (E) antiga (a) é melhor do que época contemporânea (c);
2. Se a época for antiga prefere-se ritmo (R) movimentado (m) a ritmo tranqüilo (t), se a época for contemporânea prefere-se o contrário;
3. Se a época for antiga prefere-se rock (r) a sertanejo (s) para gênero (G), se a época for contemporânea prefere-se o contrário;
4. Se a época for antiga o gênero é mais importante do que o ritmo, mas e a época for contemporânea o gênero é mais importante do que o ritmo.

Ainda no trabalho de [Brafman et al. 2006a] é proposto um algoritmo de otimização semelhante ao algoritmo utilizado pelas *CP-nets* que considera importância absoluta e relativa no processo de otimização.

2.4.2 Linguagem de Preferências Condicionais

Em [Wilson 2004b] foi especificada uma linguagem de preferências condicionais $\mathcal{L}$ que permite expressar preferências mais genéricas do que aquelas tratadas pelas CP-nets e TCP-nets.

A representação de preferências nesta linguagem considera um conjunto de atributos $A = \{X_1, ..., X_n\}$, onde $\mathbf{Dom}(X_i)$ denota o domínio de X_i, ou seja, os possíveis valores que podem ser assumidos por X_i. O conjunto $\mathbf{Dom}(X_1) \times ... \times \mathbf{Dom}(X_n)$ é denotado por $\mathcal{O}$ e representa o conjunto de objetos formados por todas as combinações possíveis de valores para os atributos de A. Se $o = (x_1, ..., x_n)$ é um objeto então $o[X_i] = x$ denota valor x para o atributo X_i no objeto o.

A representação de preferências na linguagem $\mathcal{L}$ é realizada através de *regras de preferências condicionais*, veja a Definição 2.13.

Definição 2.13. (Regra de preferência condicional) Seja $A = \{X_1, ..., X_n\}$ um conjunto de atributos. Uma *regra de preferência condicional* ou simplesmente *regra-pc* sobre A é uma instrução no formato $\varphi : u \rightarrow (X = x) > (X = x')[W]$. A condição u é uma fórmula no formato $(X_{i_1} = x_1) \wedge ... \wedge (X_{i_k} = x_k)$ onde $X_{i_j} \in A$ e $x_j \in \mathbf{Dom}(X_{i_j})$ para todo $j \in \{1, ..., k\}$. Sendo que $W \subseteq (A - \{X, X_{i_1}, ..., X_{i_k}\})$ é o conjunto de atributos desconsiderados pela regra de preferência.

Dada uma regra-pc $\varphi : u \rightarrow (X = x) > (X = x')[W]$, para referenciar diretamente a condição da regra-pc usa-se a notação u_φ, o mesmo acontece para o atributo X, os valores x e x' e o conjunto W, que podem ser referenciados diretamente através das notações X_φ, x_φ, x'_φ e W_φ respectivamente. O conjunto de atributos que aparecem em uma condição u_φ é denotado por U_φ.

A representação de um conjunto maior de preferências é feita através de uma *teoria de preferência condicional* dada pela Definição 2.14.

Definição 2.14. (Teoria de preferência condicional) Dado um conjunto de atributos A. Uma *teoria de preferência condicional* ou simplesmente *teoria-pc* é um conjunto finito de regras-pc sobre A.

Uma regra-pc $\varphi : u \rightarrow (X = x) > (X = x')[W]$ induz uma ordem de preferência sobre os objetos do conjunto $\mathcal{O}$. Sejam $o = (t, y, x, w)$ e $o' = (t, y, x', w')$ dois objetos de $\mathcal{O}$, onde y é um objeto sobre U_φ, onde w, w' são objetos sobre W_φ e t é um objeto sobre $A - (\{X\} \cup U_\varphi \cup W_\varphi)$, então $o >_\varphi o'$, ou seja, o é preferido a o' de acordo com φ.

O conjunto de pares de objetos (o, o') onde o é preferido a o' de acordo com φ é denotado por φ^*. Dada uma teoria-pc Γ, $>_\Gamma$ denota o fecho transitivo da relação binária

$\Gamma^* = \bigcup_{\varphi \in \Gamma} \varphi^*$ e conseqüentemente a ordem induzida pela teoria-pc Γ sobre os objetos de $\mathcal{O}$. Por ser uma linguagem lógica mais geral, qualquer CP-net ou TCP-net pode ser representada pela linguagem $\mathcal{L}$.

O Exemplo 2.5 mostra uma teoria-pc e sua ordem induzida sobre um conjunto de objetos.

Exemplo 2.5. Neste exemplo são usadas as mesmas preferências do Exemplo 2.2, sendo que "A preferência sobre o gênero não considera o ritmo". O conjunto de atributos em questão é $A = \{G, R, E\}$ (G para Gênero, R para Ritmo, E para Época). Quanto aos domínios é assumido que $\mathbf{Dom}(\mathrm{G}) = \{s, r\}$, $\mathbf{Dom}(\mathrm{R}) = \{m, t\}$ e $\mathbf{Dom}(\mathrm{E}) = \{a, c\}$, onde s, r, m, t, a e c correspondem a sertanejo, rock, movimentado, tranqüilo, antiga e contemporânea, respectivamente. Tais preferências podem ser representadas pela teoria-pc $\Gamma = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$, onde:

- $\varphi_1 : (G = s) > (G = r)\{R\}$;
- $\varphi_2 : (R = m) > (R = t)$;
- $\varphi_3 : (G = s \wedge R = t) \rightarrow (E = a) > (E = c)$;
- $\varphi_4 : (G = r \wedge R = m) \rightarrow (E = a) > (E = c)$;
- $\varphi_5 : (G = s \wedge R = m) \rightarrow (E = c) > (E = a)$;
- $\varphi_6 : (G = r \wedge R = t) \rightarrow (E = c) > (E = a)$.

Esta teoria-pc é equivalente a TCP-net da Figura 2.5(a). Como a preferência sobre G desconsidera R, podemos comparar os objetos (s, t, a) e (r, m, a) através de φ_1 (mesmo com R possuindo valores diferentes nos dois objetos). Isto é, podemos concluir que $(s, t, a) >_{\varphi_1} (r, m, a)$.

Para facilitar a apresentação, no restante deste trabalho será considerado $W = \emptyset$, contudo todas as propriedades da linguagem $\mathcal{L}$ continuam válidas para $W \neq \emptyset$.

Quando uma teoria-pc possui todas as regras-pc φ_i sobre um atributo X com condição vazia o atributo X é um atributo *independente*. Caso contrário, X_φ é um atributo *dependente*, pois X dependente de pelo menos um atributo Y presente na condição de uma regra-pc. No Exemplo 2.5 os atributos Gênero e Ritmo são independentes e o atributo Época é dependente.

Na linguagem $\mathcal{L}$ existem determinadas teorias-pc que podem ser inconsistentes, e por este motivo antes de proceder com raciocínios sobre uma teoria-pc é necessário garantir sua consistência.

Definição 2.15. (Teoria-pc consistente [Wilson 2004b]) Uma teoria-pc Γ é *consistente* se existe um *modelo* $>$ para Γ. Tal modelo é uma ordem parcial estrita $>$ sobre os objetos de $\mathcal{O}$ onde $>$ contém a ordenação induzida $>_\Gamma$.

Podemos notar que uma teoria-pc Γ é consistente se e somente se sua relação de ordem induzida é irreflexiva, uma vez que $>_\Gamma$ é transitiva por definição. A consistência de uma teoria-pc pode verificada através de seu *grafo de dependência preferencial* dado pela Definição 2.16 e de sua *consistência local* dada pela Definição 2.17.

Definição 2.16. (Grafo de dependência preferencial [Wilson 2004b]) Dado um conjunto de atributos A e uma teoria-pc Γ sobre A, o *grafo de dependência preferencial* sobre Γ denotado por $G(\Gamma)$ é um grafo dirigido definido como $G(\Gamma) = (V, E)$. Onde $V = A$ e $E = \bigcup_{\varphi_i \in \Gamma} E(\varphi_i)$, sendo que $E(\varphi_i) = \{(Y, X_{\varphi_i}) \mid Y \in U_{\varphi_i}\} \cup \{(X_{\varphi_i}, Z) \mid Z \in W_{\varphi_i}\}$.

Definição 2.17. (Consistência local [Wilson 2004b]) Seja Γ uma teoria-pc sobre um conjunto de atributos A. Seja $U_\Gamma = \{u_{\varphi_i} \mid \varphi_i \in \Gamma\}$. Para cada $u \in U_\Gamma$ e para cada $X \in A$, $R^*_{u,X} = \{(x_{\varphi_i}, x'_{\varphi_i}) \mid X_{\varphi_i} = X$ e u *satisfaz* $u_{\varphi_i}\}$. Seja $>^X_u$ o fecho transitivo sobre $R^*_{u,X}$. Uma teoria-pc Γ é *localmente consistente* se e somente se para todo $X \in A$ e para todo $u \in U_\Gamma$, a relação $>^X_u$ é irreflexiva.

A garantia de consistência de uma teoria-pc é dada pelo Teorema 2.4. O Exemplo 2.6 mostra como este teorema é aplicado.

Teorema 2.4. ([Wilson 2004b]) *Seja Γ uma teoria de preferência condicional tal que o grafo $G(\Gamma) = \bigcup_{\varphi \in \Gamma} (Y, X_\varphi) : Y \in U_\varphi$ é acíclico. Então Γ é consistente se e somente se Γ é localmente consistente.*

Exemplo 2.6. Considerando a teoria-pc

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : (G = s) > (G = r), \\ \varphi_2 : (G = r) \rightarrow (E = a) > (E = c) \end{array} \right\}.$$

Constatamos que Γ é consistente pois a relação $>_\Gamma = \{(o_1, o_3), (o_3, o_2), (o_4, o_2)\}$ é uma ordem parcial estrita, onde $o_1 = (s, a)$, $o_2 = (r, c)$, $o_3 = (r, a)$ e $o_4 = (s, c)$ (conseqüentemente o grafo $G(\Gamma)$ é acíclico e Γ é localmente consistente). Já a teoria-pc

$$\Gamma' = \left\{ \begin{array}{l} \varphi_1 : (G = s) > (G = r), \\ \varphi_2 : (R = m) \rightarrow (G = r) > (G = s) \end{array} \right\}$$

não é localmente consistente, pois a relação $>^G_{R=m} = \{(s, r), (r, s)\}$ é reflexiva. O par (r, s) é obtido da regra-pc φ_2, já que $R = m$ valida u_{φ_2}. O par (s, r) é obtido da regra-pc φ_1, pois u_{φ_1} é uma condição vazia e portanto é validada por qualquer atribuição.

Uma teoria-pc com sua consistência garantida possibilita a realização de raciocínios lógicos como o teste de dominância (comparação) entre objetos. O teste de dominância entre objetos segundo uma teoria-pc consistente é feito utilizando o conceito *cadeias de melhoramento de objetos*, veja a Definição 2.18. Através deste conceito o Teorema 2.5 apresenta a condição necessária e suficiente para que dois objetos quaisquer sejam comparados de acordo com uma teoria-pc.

Definição 2.18. (Cadeia de melhoramento de objetos [Wilson 2004b]) Sejam o e o' dois objetos. Existe uma *cadeia de melhoramento de objetos* de o para o' de acordo com uma teoria-pc Γ se existe uma conjunto de objetos $\{o_1, ..., o_{k+1}\}$ e um conjunto de regras-pc $\{\varphi_1, ..., \varphi_k\}$ em Γ tal que $o = o_1$, $o' = o_{k+1}$ e $o_j >_{\varphi_j} o_{j+1}$ para todo $j \in \{1, ..., k\}$.

Teorema 2.5. ([Wilson 2004b]) *Seja Γ uma teoria-pc consistente sobre A e sejam o, $o' \in \mathcal{O}$. Então, $o >_\Gamma o'$ se e somente se existe uma cadeia de melhoramento de objetos de o' para o com respeito a Γ.*

O teste de consistência de uma teoria-pc é uma tarefa muito específica da área de Inteligência Artificial que possui uma alta complexidade [Wilson 2004a, Wilson 2006]. Como o foco principal deste trabalho está na área de Banco de Dados, não serão abordados maiores detalhes sobre técnicas para realizar o teste de consistência e por este motivo será assumido que as teoria-pc são consistentes.

Outra importante contribuição do trabalho de [Wilson 2004b] foi a especificação de um método para obter os objetos ótimos a partir de um conjunto de preferências. Neste trabalho foi proposto e implementado um algoritmo baseado neste método, maiores detalhes do algoritmo são dados no Capítulo 3.

2.5 Preferências em Inteligência Artificial Adaptadas para o Contexto de Banco de Dados

Em [Endres e Kießling 2006] foi proposto um método para transformar TCP-nets em consultas na linguagem *Preference SQL*, porém este método não é muito simples e envolve uma série de passos que convertem progressivamente uma TCP-net em operadores da *álgebra de preferências* para chegar a uma consulta *Preference SQL*. Além disto, a consulta *Preference SQL* não possui um significado semântico intuitivo equivalente ao das TCP-nets e pode ser muito extensa, conforme é exibido no Exemplo 2.7.

Exemplo 2.7. Considerando a TCP-net da Figura 2.5(a) e uma relação MUSICAS sobre o esquema $M(G, R, E)$, a TCP-net seria convertida para seguinte consulta *Preference SQL*:

```
SELECT * FROM MUSICAS
PREFERRING ((G = 's' AND R = 'm' AND E = 'c')
CASCADE (G = 's' AND R = 'm' AND E = 'a')
CASCADE (G = 's' AND R = 't' AND E = 'a')
CASCADE (G = 's' AND R = 't' AND E = 'c'))
CASCADE
(CASCADE (G = 'r' AND R = 'm' AND E = 'a')
CASCADE (G = 'r' AND R = 'm' AND E = 'c')
CASCADE (G = 'r' AND R = 't' AND E = 'c')
CASCADE (G = 'r' AND R = 't' AND E = 'a'))
```

O operador CASCADE é usado para definir a ordem de preferência para os possíveis valores dos atributos, pois não é possível representar as condições semânticas como "se $A = a_1$ então $B = b_1 > B = b_2$" presentes implicitamente nas TCP-nets.

Apesar dos inconvenientes e de uma certa ineficiência na transformação, o trabalho de [Endres e Kießling 2006] propõe uma idéia interessante que é a utilização de uma ferramenta de Inteligência Artificial para resolver consultas de banco de dados contento preferências.

2.6 Considerações Finais

Neste capítulo foram apresentados diversos conceitos e propriedades relacionados à preferências sobre objetos. Também foram abordados os principais trabalhos sobre o tratamento de preferências e a incorporaração de preferências em linguagens de consultas para banco de dados. Tanto os conceitos apresentados quanto os trabalhos abordados são essencialmente importantes pois serão utilizados e referenciados em diversos momentos neste trabalho.

Capítulo 3

Operadores de Seleção de Tuplas Ótimas

Um dos principais objetivos deste trabalho é a especificação da linguagem de consulta para banco de dados *CPref-SQL* que permite a realização de consultas contendo regras de preferências condicionais [de Amo e Ribeiro 2009]. A linguagem *CPref-SQL* tem como base uma extensão da álgebra relacional convencional. Tal extensão é realizada através do operador **Best**, cuja sintaxe é a seguinte:

$$\mathbf{Best}_{\Gamma}(r).$$

O operador **Best** seleciona as tuplas da relação r considerando um conjunto de preferências representado através da teoria-pc Γ (ver Definição 2.14). Podemos considerar três semânticas distintas para o operador **Best**. Para ilustrar cada uma destas semânticas vamos elaborar exemplos considerando a relação r_1 sobre o esquema R(A, B, C) exibida na Figura 3.1.

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_1	c_1
a_3	b_1	c_2
a_2	b_2	c_2

Figura 3.1: Relação r_1

Inicialmente, podemos considerar a semântica na qual o operador **Best** retorna somente as tuplas ótimas no *sentido absoluto*, ou seja, as tuplas que atendem exatamente as preferências informadas independente da relação. O Exemplo 3.1 ilustra esta primeira semântica.

Exemplo 3.1. Como exemplo vamos supor que um usuário expresse suas preferências através da teoria-pc $\Gamma_1 = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde:

- $\varphi_1 : (A = a_1) > (A = a_2)$;

- $\varphi_2 : (A = a_3) > (A = a_2)$;
- $\varphi_3 : (A = a_1) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi_4 : (A = a_3) \rightarrow (B = b_2) > (B = b_1)$;
- $\varphi_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

As tuplas que atendem exatamente as preferências de Γ_1, (isto é as tuplas ótimas absolutas) devem possuir $A = a_1$ e $B = b_1$ ou $A = a_3$ e $B = b_2$. No caso da relação r_1 da Figura 3.1, a única tupla que atende estas condições é (a_1, b_1, c_1). Porém, se as preferências do usuário forem dadas pela teoria-pc $\Gamma_2 = \{\varphi_2, \varphi_4, \varphi_5\}$, onde:

- $\varphi_2 : (A = a_3) > (A = a_2)$;
- $\varphi_4 : (A = a_3) \rightarrow (B = b_2) > (B = b_1)$;
- $\varphi_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

A relação r_1 da Figura 3.1 não possui nenhuma tupla ótima absoluta segundo Γ_2 e o operador retorna um resultado vazio.

Com a utilização desta primeira semântica, em diversas situações poderá ocorrer um resultado vazio o que causará a insatisfação do usuário.

Uma alternativa para evitar os resultados vazios é verificar se a relação possui alguma das tuplas ótimas absoluta e caso não possua nenhuma, retornar todas as tuplas da relação.

Vamos considerar novamente as teorias-pc Γ_1 e Γ_2 do Exemplo 3.1 e a relação r_1 da Figura 3.1. Para a teoria-pc Γ_1, o operador com esta nova semântica também retorna a tupla (a_1, b_1, c_1). No caso da teoria-pc Γ_2, o operador retorna todas as tuplas da relação r_1. Esta alternativa resolve o problema dos resultados vazios, mas por outro lado, quando as preferências não são atendidas por nenhuma tupla da relação pode ser retornada uma quantidade muito grande de tuplas e o usuário será inundado com uma resposta excessivamente grande.

A solução para evitar resultados vazios e resultados exageradamente grandes é adotar uma terceira semântica, onde são retornadas as tuplas ótimas *com respeito* a r que melhor atendem as preferências de Γ, isto é, as tuplas para as quais não existe nenhuma outra tupla melhor em r segundo Γ. Ou seja, é retornado o conjunto de tuplas

$$\{t \in r \mid \nexists t' \in r, t' >_\Gamma t\}.$$

As tuplas deste conjunto são chamadas de tuplas *dominantes* em r de acordo com Γ. Como exemplo, vamos considerar mais uma vez a teoria-pc Γ_2 do Exemplo 3.1 e a relação r_1 da Figura 3.1 utilizando esta terceira semântica. Neste caso são retornadas as tuplas dominantes (a_1, b_1, c_1), (a_3, b_1, c_1) e (a_3, b_1, c_2).

Para distinguir entre as três semânticas abordadas será definido um operador para cada semântica, são eles:

- Operador **Best-E**$_\Gamma(r)$, retorna somente as tuplas ótimas absolutas de acordo com Γ que estão contidas na relação r (primeira semântica);
- Operador **Best-N**$_\Gamma(r)$, retorna as tuplas ótimas absolutas de acordo com Γ que estão contidas na relação r, se nenhuma destas tuplas existir então são retornadas todas as tuplas de r (segunda semântica);
- Operador **Best-D**$_\Gamma(r)$, retorna as tuplas dominantes em r de acordo com Γ (terceira semântica).

No que diz respeito ao poder de expressão dos operadores **Best-E**, **Best-N** e **Best-D**, podemos constatar que o operador **Best-D** possui a mesma semântica do operador *winnow* apresentado no Capítulo 2. O operador *winnow* pode ser obtido através dos operadores básicos da álgebra relacional [Chomicki 2003].

Os operadores **Best-E** e **Best-N** obtêm as tuplas ótimas absolutas com respeito a uma teoria-pc Γ contidas em uma relação r sobre o esquema $R(A_1, ..., A_n)$ que podem ser obtidas pela operação **Best-D**$_\Gamma(\mathcal{D}(r)) \cap r$. A operação $\mathcal{D}(r)$ constrói uma *relação universal* que possui as tuplas obtidas pela combinação dos domínios dos atributos de R, ou seja, $\mathbf{Dom}(A_1) \times ... \times \mathbf{Dom}(A_n)$. De acordo com [Abiteboul et al. 1995] está operação pode ser feita usando os operadores básicos da álgebra relacional. Logo, podemos concluir que os operadores **Best-E**, **Best-N** e **Best-D** não aumentam o poder de expressão da álgebra, mas por serem operadores com características muito particulares, sua especificação permite desenvolver algoritmos próprios com um desempenho otimizado.

Os operadores **Best-E** e **Best-N** obtêm as tuplas ótimas de forma direta verificando quais os valores de atributos estas tuplas devem possuir e não é necessário nenhum teste de dominância, ou seja, para obter as tuplas ótimas absolutas não é necessário comparações entre tuplas, como veremos no Capítulo 5 através dos algoritmos **GetBest-E** e **GetBest-N**. Entretanto, para o operador **Best-D** o teste de dominância entre tuplas é uma tarefa indispensável pois antes de retornar uma tupla t é preciso garantir que nenhuma outra tupla da relação seja preferida a t, a necessidade de tal tarefa poderá ser observada no algoritmo **GetBest-D** apresentado no Capítulo 5.

Em determinadas situações, a comparação entre tuplas pode ser feita de forma direta, como acontece nos trabalhos de [Kießling 2002, Kießling e Köstler 2002] onde as preferências são especificadas na forma de composições *Pareto* de ordens fracas sobre atributos. Conforme foi mostrado no Capítulo 2, é possível associar funções de *ranking* a ordens fracas e desta forma realizar comparações usando estas funções.

No nosso caso o teste de dominância entre tuplas não é uma tarefa trivial pois as preferências são especificadas através de regras condicionais o que resulta em ordens parciais não fracas. Inclusive existem trabalhos na área de Inteligência Artificial que tratam especificamente desta tarefa como [Boutilier et al. 1999, Domshlak e Brafman 2002a]. Como vimos no Capítulo 2, para realizar um teste de dominância entre duas tuplas t e t'

considerando uma teoria-pc Γ (isto é, testar se $t >_\Gamma t'$ ou se $t' >_\Gamma t$) é preciso encontrar uma cadeia de melhoramento de objetos entre as tuplas t e t'.

Em nosso trabalho propomos uma forma de realizar o teste de dominância entre tuplas com base em *listas de escopos*. A utilização do conceito de escopos tem como base o trabalho de [Agrawal et al. 1989], o qual propôs um método de calcular o fecho transitivo de uma relação binária baseado em listas de escopos (no trabalho de [Agrawal et al. 1989] os escopos são chamados de intervalos).

O que desejamos é comparar qualquer par de tuplas t, t' segundo uma teoria-pc Γ de forma direta (sem ter que encontrar uma cadeia de melhoramento de objetos entre t e t'). Se a ordem induzida por Γ fosse uma ordem fraca então poderiamos utilizar funções de *ranking* para comparar as tuplas, isto seria feito calculando as funções de $f_r(t)$ para t e $f_r(t')$ para t'. Como os resultados de tais funções são valores numéricos, a comparação entre t e t' se resumiria a comparação dos resultados de $f_r(t)$ e $f_r(t')$. Este tipo de comparação é ilustrado através da Figura 3.2(a).

Como a ordem induzida por uma teoria-pc Γ pode não ser uma ordem fraca, não podemos utilizar funções de *ranking* para testar se uma tupla domina outra, mas podemos utilizar listas de escopos para tal. A comparação utilizando listas de escopos faz uso do conceito de *abrangência* (Explicado em detalhes na Seção 3.2), desta forma a comparação entre duas tuplas t e t' é feita através da verificação de abrangência entre as listas de escopos l_t e $l_{t'}$ associadas as tuplas t e t', respectivamente. Este tipo de comparação é ilustrado através da Figura 3.2(b).

(a) Ordens fracas (b) Ordens não fracas

Figura 3.2: Comparação entre tuplas em ordens fracas e em ordens não fracas

Dadas uma teoria-pc Γ consistente e uma relação r sobre um esquema R, para obtermos as listas de escopos associadas às tuplas de r é preciso construir estruturas de grafo baseadas nas regras de preferências de Γ e nos domínios dos atributos de R. Como a obtenção de listas de escopos depende de tais estruturas, primeiramente apresentaremos a construção destas estruturas para que possamos definir as listas de escopos.

A Seção 3.1 apresenta a construção do grafo BTG (*Better-then graph*) e sua árvore de cobertura mínima. Em seguida, na Seção 3.2 demonstramos como obter os escopos associados aos nós da árvore de cobertura mínima do grafo BTG. A Seção 3.3 define as listas de escopos para as tuplas utilizando o grafo BTG e o conceito de escopo da Seção 3.2. Já na Seção 3.4 discutimos como obter uma árvore de cobertura mínima ótima para o grafo BTG. A Seção 3.5 apresenta uma otimização para o teste de dominância utilizando

listas de escopos. E finalmente, na Seção 3.6 apresentamos as considerações finais deste capítulo.

3.1 O Grafo BTG e sua Árvore de Cobertura Mínima

Um grafo BTG com respeito a uma teoria-pc Γ é dado pela Definição 3.1.

Definição 3.1. (Grafo BTG com respeito a uma teoria-pc Γ) Seja r uma relação sobre o esquema $R\ (A_1, ..., A_n)$ e Γ uma teoria-pc consistente contendo preferências sobre as tuplas de r. Um *grafo BTG com respeito a* Γ é um grafo dirigido $BTG_\Gamma(V, E)$, onde o conjunto de nós V do grafo e o conjunto de arestas E do grafo são definidos da seguinte maneira:

- $V = \mathcal{O} \cup \{vr\}$, onde vr é um nó virtual e $\mathcal{O} = \mathbf{Dom}(A_1) \times ... \times \mathbf{Dom}(A_n)$ é o conjunto de todas as tuplas formadas pelo produto cartesiano dos domínios dos atributos de R (como foi definido no Capítulo 2).
- $E = E' \cup E''$, onde $E' = \{(t, t') \mid t >_{\varphi_i} t' \text{ e } t, t' \in \mathcal{O} \text{ e } \varphi_i \in \Gamma\}$ e $E'' = \{(vr, t) \mid t, t' \in \mathcal{O}$ e $\nexists (t', t) \in E'\}$. E' é o conjunto das arestas obtidas através das comparações entre as tuplas de $\mathcal{O}$ utilizando as regras-pc de Γ e E' é o conjunto de arestas ligando as tuplas ótimas de $\mathcal{O}$ à raiz virtual vr do grafo, garantindo que o grafo BTG possua uma única componente conexa.

Em outras palavras, o grafo BTG_Γ é utilizado para representar a ordem induzida por uma teoria-pc Γ sobre as tuplas de $\mathcal{O}$, ou seja, um caminho do nó t para o nó t' indica que a tupla t é preferida à tupla t' segundo Γ. O Exemplo 3.2 mostra uma teoria-pc com seu respectivo grafo BTG.

Exemplo 3.2. Como exemplo vamos considerar novamente a relação r_1 da Figura 3.1 com os seguintes domínios para os atributos $\mathbf{Dom}(A) = \{a_1, a_2, a_3\}$, $\mathbf{Dom}(B) = \{b_1, b_2\}$ e $\mathbf{Dom}(C) = \{c_1, c_2\}$. Vamos considerar também a teoria-pc $\Gamma_2 = \{\varphi_1, \varphi_2, \varphi_3\}$, onde:

- $\varphi_1 : (A = a_3) > (A = a_2)$;
- $\varphi_2 : (A = a_3) \rightarrow (B = b_2) > (B = b_1)$;
- $\varphi_3 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

O grafo BTG_{Γ_2} é exibido da Figura 3.3.

É importante frisar que um grafo BTG sempre será acíclico pois representa a ordem induzida por uma teoria-pc consistente que, por definição, induz uma relação de ordem irreflexiva sobre $\mathcal{O}$. Após a construção do grafo BTG_Γ é obtida sua árvore de cobertura mínima T_Γ dada pela Definição 3.2.

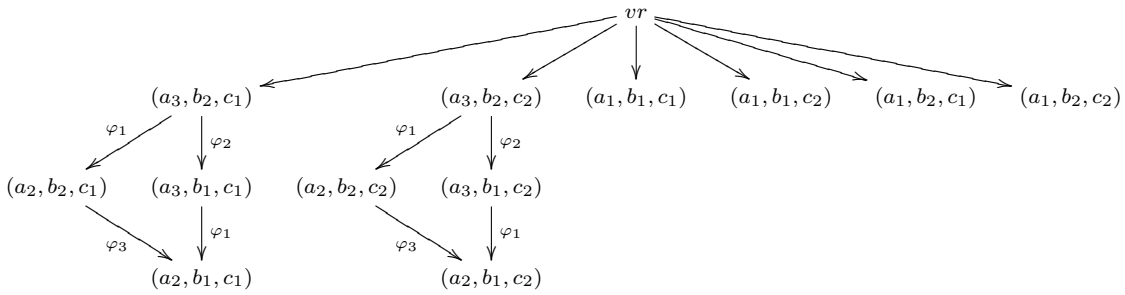
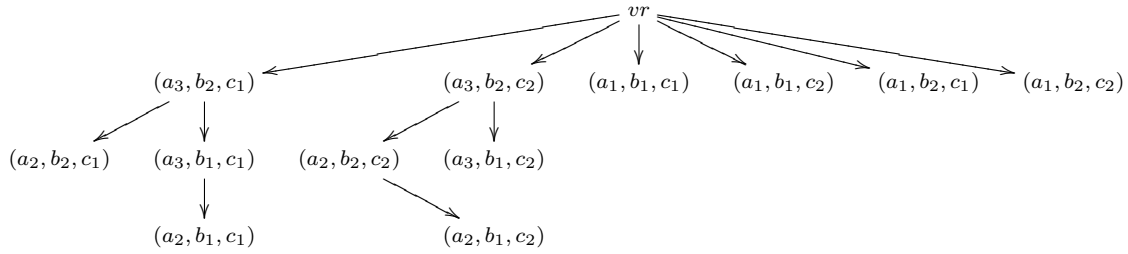


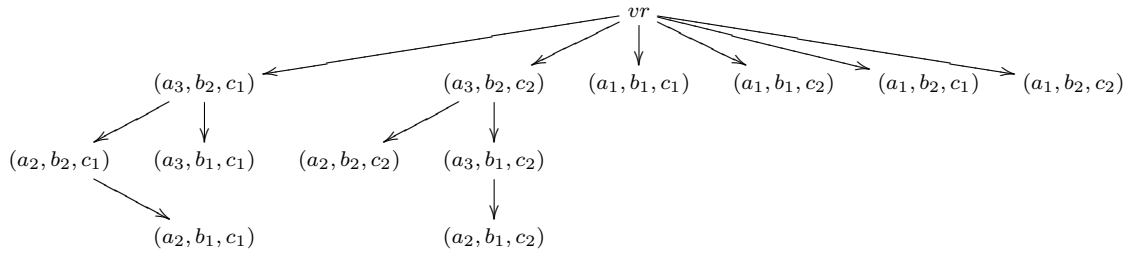
Figura 3.3: Grafo BTG_{Γ_2}

Definição 3.2. (Árvore de cobertura mínima) Seja $BTG_\Gamma(V, E)$ um grafo BTG com respeito a uma teoria-pc Γ consistente. Uma *árvore de cobertura mínima* T_Γ sobre BTG_Γ é um subgrafo de BTG_Γ contendo todos os nós de BTG_Γ e possuindo um único caminho de vr para os demais nós da árvore.

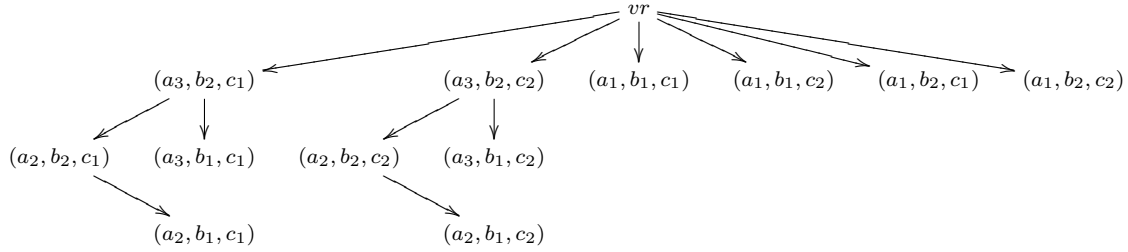
A Definição 3.2 garante que uma árvore de cobertura mínima T_Γ sobre um grafo BTG_Γ tem sempre o nó vr como raiz. Podemos notar também que um grafo BTG pode possuir mais de uma árvore de cobertura mínima. No caso do grafo BTG_{Γ_2} da Figura 3.3 existem quatro árvores de extensão exibidas nas Figuras 3.4(a), 3.4(b), 3.4(c) e 3.4(d). A princípio podemos considerar qualquer uma destas árvores de cobertura mínima, vamos considerar a árvore da Figura 3.4(a) para o grafo BTG_{Γ_2} e chamá-la de T_{Γ_2}.



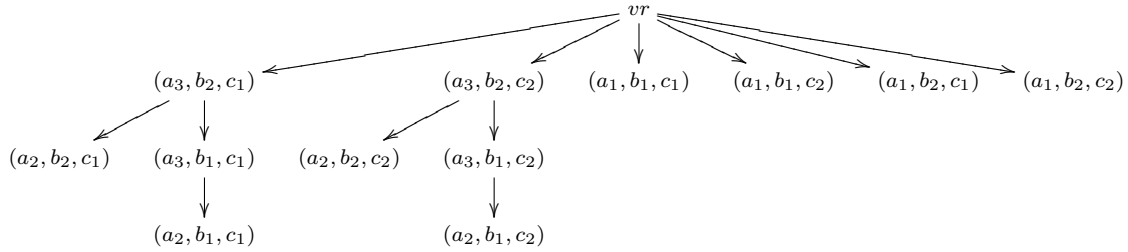
(a) Árvore de cobertura mínima 1



(b) Árvore de cobertura mínima 2



(c) Árvore de cobertura mínima 3



(d) Árvore de cobertura mínima 4

Figura 3.4: Árvores de extensão para o grafo BTG_{Γ_2}

3.2 Escopo de Tuplas com Respeito a T_Γ

Após a definição de grafo BTG e de árvore de cobertura mínima podemos começar a definir o que é o escopo de uma tupla. O escopo de uma tupla é composto por um *número de índice* e por um *número de pós-ordem*.

Os números de pós-ordem das tuplas representadas pelos nós de T_Γ são obtidos através de uma numeração incremental percorrendo T_Γ em profundidade, a numeração começa em 1 e é incrementada a cada nó (tupla) numerado, sendo que um nó só pode ser numerado quando seus filhos já foram numerados.

A Figura 3.5 mostra como é feita a numeração de pós-ordem para a árvore T_{Γ_2} da Figura 3.4(a). As arestas tracejadas descrevem o percurso realizado em T_{Γ_2} durante a

numeração.

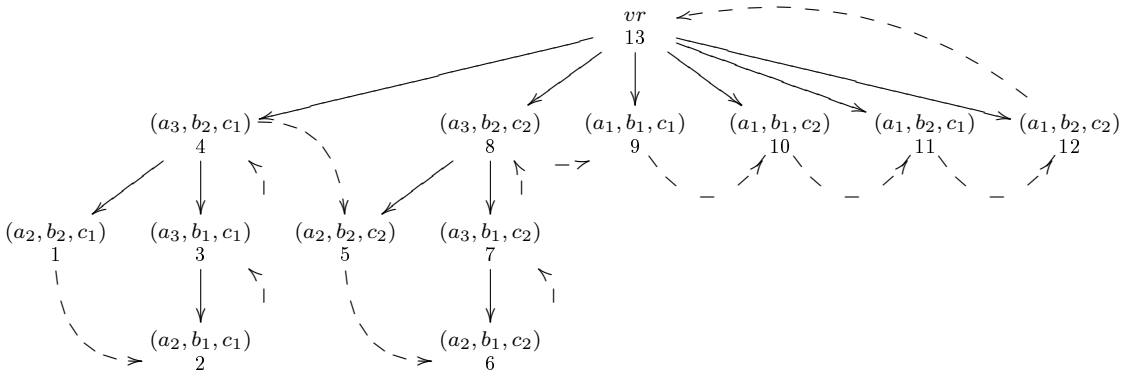


Figura 3.5: Árvore de cobertura mínima T_{Γ_2} com números de pós-ordem

Além do número de pós-ordem cada nó em T_Γ recebe um *número de índice* que consiste do menor número de pós-ordem de seus descendentes, no caso dos nós folhas (que não possuem descendentes) o número de índice é igual a seu próprio número de pós-ordem. Agora nós podemos afirmar o que é o escopo de uma tupla através da Definição 3.3.

Definição 3.3. (Escopo de uma tupla) Seja T_Γ uma árvore de cobertura mínima de um grafo BTG_Γ, onde t é um nó em T_Γ (que representa a tupla t). O *escopo* de t é $[i, j]$, onde i é o número de índice de t e j é o número de pós-ordem de t com relação a árvore T_Γ.

Na Figura 3.6 podemos ver a árvore T_{Γ_2} com os escopos associados aos nós. Após a definição de escopos podemos especificar como comparar dois escopos utilizado o conceito de abrangência, conforme a Definição 3.4.

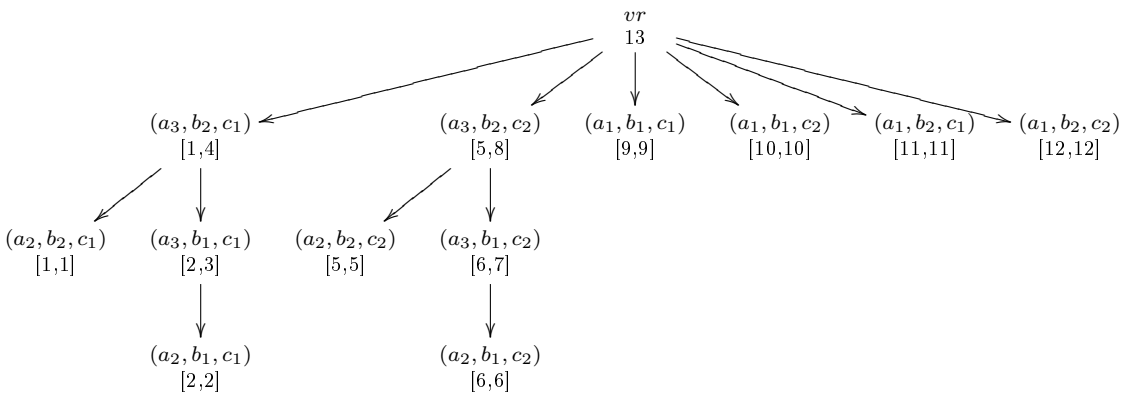


Figura 3.6: Árvore de cobertura mínima T_{Γ_2} com escopos associados

Definição 3.4. (Abrangência) Sejam t, t' duas tuplas tendo $[i, j]$ e $[i', j']$ como seus respectivos escopos. Dizemos que $[i, j]$ *abrange* $[i', j']$ se e somente se $i \leq j' < j$.

Como exemplo, vamos considerar os nós (a_3, b_2, c_1), (a_2, b_1, c_1) e (a_3, b_1, c_2) da árvore de cobertura mínima T_{Γ_2} e seus respectivos escopos $[1, 4]$, $[2, 2]$ e $[6, 7]$. Podemos ver que $[1, 4]$ abrange $[2, 2]$ e $[1, 4]$ não abrange $[6, 7]$. O conceito de abrangência está ligando com os caminhos entre os nós da árvore de cobertura mínima, como podemos ver no Lema 3.1.

Lema 3.1. ([Agrawal et al. 1989]) *Seja $[i, j]$ um escopo associado a um nó t e $[i', j']$ um escopo associado a um nó t'. Então existe um caminho de t para t' se e somente se $[i, j]$ abrange $[i', j']$.*

Considerando novamente os nós (a_3, b_2, c_1) e (a_2, b_1, c_1) da árvore de cobertura mínima T_{Γ_2}, o escopo de (a_3, b_2, c_1) abrange o escopo de (a_2, b_1, c_1) porque existe um caminho de (a_3, b_2, c_1) para (a_2, b_1, c_1) passando por (a_3, b_1, c_1), como podemos ver através da Figura 3.6.

3.3 Listas de Escopos de tuplas com respeito a BTG_Γ

Com os escopos associados aos nós é possível verificar diretamente a existência de um caminho entre qualquer par de nós da árvore de cobertura mínima T_Γ, conforme determina o Lema 3.1. No entanto, os escopos ligados às arestas de T_Γ, chamados de *escopos principais*, podem não considerar todas as arestas de BTG_Γ e nem todos os caminhos entre os nós podem estar representados.

Para que todos os caminhos entre os nós sejam representados é necessário obter os *escopos secundários* ligados às arestas de BTG_Γ que não estão presentes em T_Γ. Os escopos secundários são obtidos da seguinte maneira:

- Examina-se todos os nós de BTG_Γ em uma ordem topologicamente reversa (partindo dos nós folhas). Para toda aresta $(t, t') \in BTG_\Gamma$ tal que $(t, t') \notin T_\Gamma$:
  1. O escopo primário $[i, j]$ de t' é acrescentado como o escopo secundário $[i, j+1]$ de t;
  2. Todos os escopos secundários de t' tornam-se escopos secundários de t;
  3. Quando t possui dois escopos $[i, j]$ e $[i', j']$ tal que $i \leq i'$ e $j' \leq j$, o escopo $[i', j']$ é descartado;

No trabalho de [Agrawal et al. 1989], quando existe uma aresta (t, t') em BTG_Γ que não está presente em T_Γ, o escopo primário de t' é adicionado como escopo secundário de t sem qualquer alteração (isto é, sem o incremento do número de pós-ordem de t'). Entretanto isto contradiz o Lema 3.1 e não foi tratado no trabalho de [Agrawal et al. 1989]. Por este motivo, modificamos o escopo primário $[i, j]$ de t' para $[i, j+1]$ ao acrescentá-lo como escopo secundário de t. Esta modificação não causa nenhuma alteração na corretude da abrangência dos escopos, como é demonstrado pelo Lema 3.2.

Lema 3.2. *Seja BTG_Γ um grafo BTG associado a uma teoria-pc Γ, T_Γ uma árvore de cobertura mínima de BTG_Γ e (t, t') uma aresta de BTG_Γ que não está presente em T_Γ. O escopo primário $[i, j]$ de t' acrescentado como o escopo secundário $[i, j+1]$ a t mantém a abrangência correta para os escopos de BTG_Γ.*

Prova. É necessário mostrar que o escopo secundário $[i, j+1]$ adicionado a t acrescenta a abrangência necessária e que o escopo secundário $[i, j+1]$ adicionado a t não acrescenta abrangência desnecessária (e incorreta).

Pela Definição 3.3, temos que $[i, j+1]$ abrange $[i, j]$, pois $i \leq j < j+1$. Para os descendentes de t', temos que $[i, j]$ abrange $[i', j']$, onde $[i', j']$ é o escopo de algum descendente de t'. Assim temos que $[i, j+1]$ abrange $[i', j']$, pois $i \leq j' < j$ e, portanto $i \leq j' < j+1$.

Supomos por absurdo que $[i, j+1]$ abrange um escopo $[i'', j'']$ diferente dos escopos de t' e seus descendentes. Pelo Lema 3.1, temos que $i \leq j'' < j+1$ e $i \leq j < j+1$. Como supomos que $[i'', j'']$ é diferente dos escopos de t' e seus descendentes, temos que $j'' < i$ ou $j'' > j$. Se $j'' < i$, então $[i, j+1]$ não abrange $[i'', j'']$. Se $j'' > j$, então $j'' \geq j+1$ e portanto $[i, j+1]$ não abrange $[i'', j'']$. O que é um absurdo.

A Figura 3.7 exibe o grafo BTG_{Γ_2} já com todos os escopos (principais e secundários) associados aos nós, as arestas tracejadas são as arestas do grafo BTG_{Γ_2} que não estão presentes em T_{Γ_2} e os escopos secundários estão destacados em negrito.

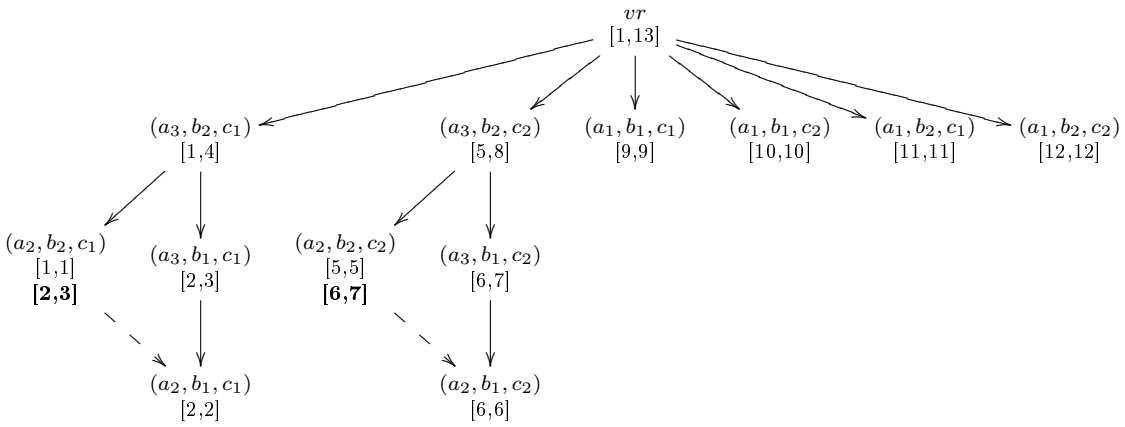


Figura 3.7: Grafo BTG_{Γ_2} com todos os escopos associados

Depois que os escopos secundários foram adicionados a T_Γ a representação da ordem induzida por Γ está completa, ou seja, todos os caminhos entre os nós de BTG_Γ estão representados pelos escopos. Agora podemos dizer o que é uma lista de escopos através da Definição 3.5.

Definição 3.5. (Lista de escopos) Dado um grafo BTG_Γ com relação a uma teoria-pc Γ, tal que BTG_Γ contém todos os escopos (principais e secundários) associados aos nós.

Uma *lista de escopos* l para um nó t de BTG_Γ é uma lista onde o primeiro elemento é o escopo principal de t e os demais elementos são os escopos secundários de t.

Com a definição de lista de escopos, agora podemos especificar a comparação de listas de escopos seguindo a noção de abrangência, dada pela Definição 3.6.

Definição 3.6. (Comparação entre listas de escopos) Dadas duas listas de escopos l e l' com respeito a um BTG_Γ, l abrange l' se e somente se algum escopo de l (principal ou secundário) abrange o escopo principal l'.

E através da Definição 3.7 podemos especificar como comparar duas tuplas utilizando suas listas de escopos.

Definição 3.7. (Comparação entre tuplas utilizando listas de escopos) Dadas uma teoria-pc Γ e duas tuplas $t, t' \in \mathcal{O}$ com suas respectivas listas de escopos l, l' com respeito a BTG_Γ, $t >_\Gamma t'$ se e somente se l abrange l'.

Quando uma lista de escopos l de uma tupla t abrange uma lista de escopos l' de uma tupla t' implicitamente existe uma cadeia de melhoramento de objetos de t' para t, desta forma a Definição 3.7 é validada pelo Teorema 2.5.

A princípio, o cálculo das listas de escopos pode parecer caro. Contudo este cálculo reduz drasticamente o custo do teste de dominância entre tuplas. Quando duas tuplas t e t' não podem ser comparadas diretamente (através de uma única regra-pc), é necessário descobrir uma cadeia de melhoramento de objetos entre t e t'. A grande vantagem da utilização das listas de escopos no teste de dominância está no fato de que as listas de escopos permitem comparar qualquer par de tuplas diretamente. O Exemplo 3.3 mostra uma comparação entre os testes de dominância com e sem o uso de listas de escopos.

Exemplo 3.3. Neste exemplo vamos considerar novamente a teoria-pc Γ_2 do Exemplo 3.2 e as tuplas (a_3, b_1, c_1) e (a_2, b_1, c_1) pertencentes a relação r_1 da Figura 3.1. O teste de dominância sem o uso de listas de escopos é feito da seguinte forma:

- Procura-se uma cadeia de melhoramento de objetos de (a_3, b_1, c_1) para (a_2, b_1, c_1);
  - O único melhoramento possível é através da regra-pc φ_4, chegando-se a tupla (a_3, b_2, c_1);
  - A partir da tupla (a_3, b_2, c_1) não é possível fazer melhoramentos, portando não existe uma cadeia de melhoramento de objetos de (a_3, b_1, c_1) para (a_2, b_1, c_1);
- Procura-se uma cadeia de melhoramento de objetos de (a_2, b_1, c_1) para (a_3, b_1, c_1);
  - De acordo com a regra φ_5, (a_2, b_1, c_1) pode ser melhorada para (a_2, b_2, c_1);
  - E pela regra φ_5, (a_2, b_2, c_1) pode ser melhorada para (a_3, b_2, c_1) e a partir desta tupla não é possível fazer melhoramentos;

– Considerando agora a regra φ_2, constatamos que $(a_3, b_1, c_1) >_{\varphi_2} (a_2, b_1, c_1)$ e portanto $(a_3, b_1, c_1) >_{\Gamma_2} (a_2, b_1, c_1)$;

Já utilizando as listas de escopos a comparação é feita de forma direta. A lista de escopos $l_{(a_3,b_1,c_1)} = ([2, 3])$ de (a_3, b_1, c_1) abrange a lista de escopos $l_{(a_2,b_1,c_1)} = ([1, 1], \mathbf{[2,2]})$ de (a_2, b_1, c_1) e portanto $(a_3, b_1, c_1) >_{\Gamma_2} (a_2, b_1, c_1)$. Se considerarmos as tuplas (a_2, b_1, c_1) e (a_2, b_1, c_2) a comparação sem o uso de listas de escopos é mais cara ainda pois estas tuplas são incomparáveis e serão gastos oito melhoramentos para chegarmos a esta conclusão.

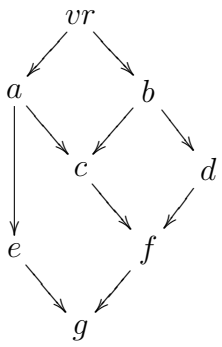
3.4 Árvore de Cobertura Mínima Ótima

Como mencionamos anteriormente, um grafo BTG pode ser coberto por mais de uma árvore de cobertura mínima, e dependendo da árvore escolhida o número de escopos pode aumentar ou diminuir. Por esta razão, a árvore de cobertura mínima escolhida deve ser ótima, ou seja, deve ser uma árvore que gere um número mínimo de escopos em cada nó. A obtenção de uma árvore de cobertura mínima ótima para um grafo BTG é feita pelo algoritmo **OptimumSpanningTree**[1] proposto no trabalho de [Agrawal et al. 1989] e apresentado com maiores detalhes no Capítulo 5.

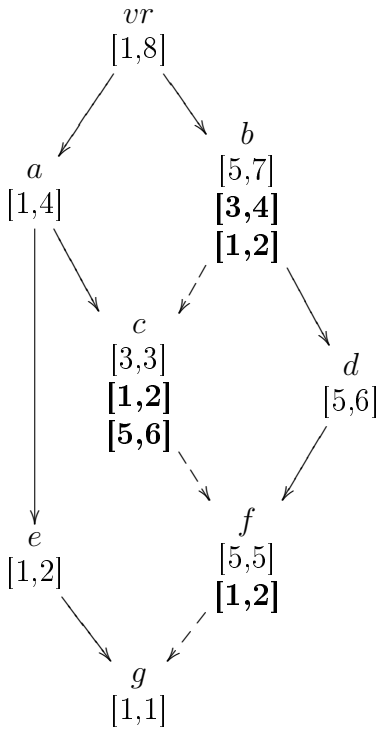
Um grafo BTG pode ter mais de uma árvore de cobertura mínima ótima e qualquer uma delas pode ser utilizada para calcular as listas de escopos. Como exemplo vamos considerar o grafo BTG da Figura 3.8(a). A Figura 3.8(b) exibe uma árvore de cobertura mínima não ótima, enquanto as Figura 3.8(c) e 3.8(d) exibem duas possíveis árvores de extensão ótimas para o grafo.

Deste ponto em diante, assumimos que as árvores de extensão escolhidas para os grafos BTG são sempre ótimas.

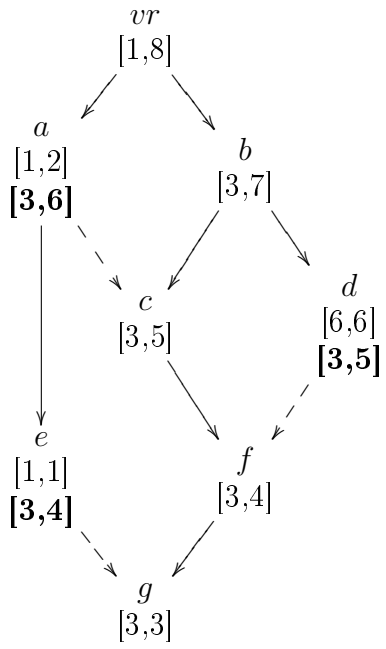
[1] O problema da árvore de cobertura mínima ótima é diferente do problema de árvore espalhada mínima, como poderá ser observado através do algoritmo **OptimumSpanningTree**.



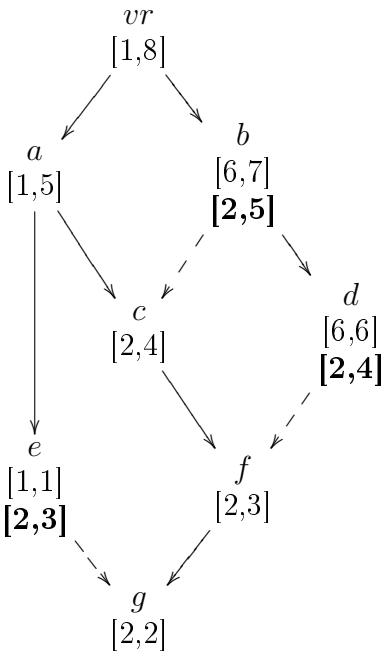
(a) Grafo BTG



(b) Árvore de cobertura mínima não ótima



(c) Árvore de cobertura mínima ótima 1



(d) Árvore de cobertura mínima ótima 2

Figura 3.8: Grafo BTG com árvores de extensão ótimas e não ótimas

3.5 Otimização

O grafo BTG_Γ com respeito a uma teoria-pc Γ possui um nó para cada tupla de $\mathcal{O}$. Como vimos anteriormente, o conjunto $\mathcal{O}$ é constituído pelo produto cartesiano dos domínios dos atributos de um esquema R de uma relação r. Desta maneira, se R possuir vários atributos ou se a amplitude de domínio dos atributos de R for muito grande, então o número de tuplas de $\mathcal{O}$ será bastante grande e, conseqüentemente, o grafo BTG_Γ ocupará um espaço de armazenamento considerável.

Com o objetivo de otimizar o método de obtenção das listas de escopos, vamos mostrar como um grafo BTG pode ser construído através de vários subgrafos BTG. Deste modo, o grafo BTG não será alterado, mas será necessário construir apenas uma parte do grafo que será *clonada* várias vezes.

Estas partes que serão clonadas são *subgrafos clones* obtidos a partir de uma *subgrafo essencial* dado pela Definição 3.8.

Definição 3.8. (Subgrafo essencial) Seja Γ uma teoria-pc associada a uma relação r sobre um esquema $R(A_1, ..., A_n)$. Seja $\mathbf{Attr}(\Gamma) = \{A_i \mid A_i \text{ocorre em } \Gamma\}$. Seja r' uma relação sobre o esquema $R'(A'_1, ..., A'_k)$ tal que $\mathbf{Attr}(\Gamma) = \{A'_1, ..., A'_k\}$. Um *subgrafo essencial* $SBTG_\Gamma$ é um grafo BTG com respeito a teoria-pc Γ associada a relação r'.

O exemplo a seguir mostra uma teoria-pc e seu respectivo subgrafo essencial.

Exemplo 3.4. Como exemplo vamos considerar a relação r_2 sobre o esquema $R2(A, B, C, D)$, onde $\mathbf{Dom}(A) = \{a_1, a_2\}$, $\mathbf{Dom}(B) = \{b_1, b_2\}$, $\mathbf{Dom}(C) = \{c_1, ..., c_{10}\}$ e $\mathbf{Dom}(D) = \{d_1, ..., d_{10}\}$ e a teoria-pc $\Gamma_3 = \{\varphi_1, \varphi_2\}$, onde:

- $\varphi_1 : (A = a_1) > (A = a_2)$;
- $\varphi_2 : (A = a_1) \rightarrow (B = b_2) > (B = b_1)$;

O subgrafo essencial $SBTG_{\Gamma_3}$ é exibido na Figura 3.9.

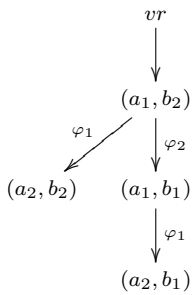


Figura 3.9: Subgrafo essencial $SBTG_{\Gamma_3}$

Podemos observar que o subgrafo essencial não considera o conjunto de atributos $A_{\neg\Gamma} = \mathbf{Attr}(R) - \mathbf{Attr}(\Gamma)$, onde $\mathbf{Attr}(\Gamma)$ é o conjunto de atributos da relação R. Isto é, o subgrafo essencial não considera os atributos de R que não ocorrem em Γ.

Os *subgrafos clones* com respeito a Γ são obtidos considerando as possíveis combinações de valores para os atributos de $A_{\neg\Gamma}$. Para cada combinação $(a_1, ..., a_m) \in \mathbf{Dom}(A''_1) \times ... \times \mathbf{Dom}(A''_m)$ é obtido um subgrafo clone $SBTG_\Gamma(a_1, ..., a_m)$, onde $A_{\neg\Gamma} = \{A''_1, ..., A''_m\}$. Além disto, todas as tuplas de um subgrafo clone $SBTG_\Gamma(a_1, ..., a_m)$ são completadas com as valores $(a_1, ..., a_m)$.

No caso da teoria-pc Γ_3 e da relação r_1 do Exemplo 3.4, $A_{\neg\Gamma} = \{C, D\}$. Portanto, deve existir um subgrafo clone para cada combinação em $\mathbf{Dom}(C) \times \mathbf{Dom}(D)$, o que resultará em cem combinações diferentes. Os subgrafos clones obtidos a partir do subgrafo essencial $SBTG_{\Gamma_3}$ são exibidos na Figura 3.10.

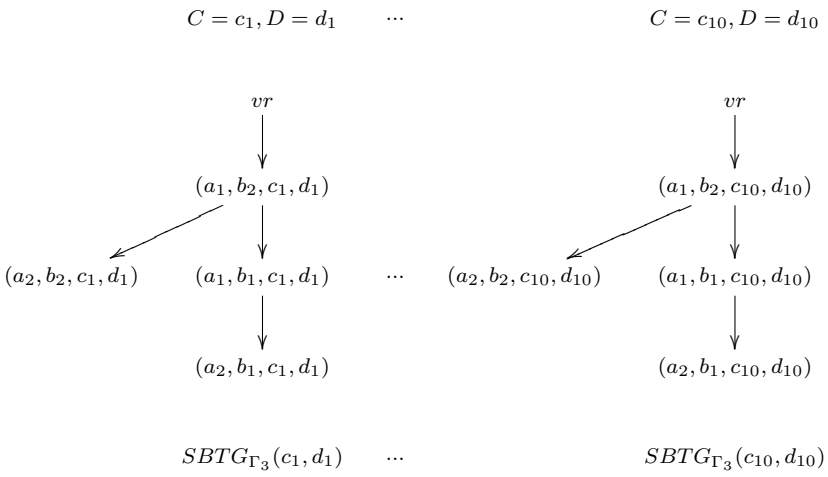


Figura 3.10: Clones para o subgrafo $SBTG_{\Gamma_3}$

Os subgrafos clones mantém o mesmo poder de representação do grafo BTG original, pois o grafo BTG original pode ser obtido através da união dos subgrafos clones, conforme determina o Lema

Lema 3.3. *Todo grafo BTG com respeito a uma teoria-pc* Γ *é a união de todos subgrafos clones com respeito a* Γ.

Prova. Seja Γ uma teoria-pc sobre uma relação $R(A_1, ..., A_n)$. Seja $BTG(V, E)_\Gamma$ o grafo BTG com respeito a Γ. Seja $\mathcal{B} = \{SBTG(V_1, E_1)_{\Gamma,1}, ..., SBTG(V_p, E_p)_{\Gamma,p}\}$ o conjunto com todos os subgrafos clones de Γ. Para que BTG_Γ seja a união dos subgrafos clones de $\mathcal{B}$ então $V = V_1 \cup .. \cup V_p$ e $E = E_1 \cup .. \cup E_p$.

Vamos provar que $V = V_1 \cup .. \cup V_p$. Sabemos que $V = \mathcal{O} \cup \{vr\}$ e $\mathcal{O} = \mathbf{Dom}(A_1) \times ... \times \mathbf{Dom}(A_n)$. Cada subgrafo $SBTG(V_j, E_j)_{\Gamma,j}$ todas as combinações para os atributos de $\mathbf{Attr}(\Gamma)$ e o nó vr. Existe um subgrafo clone para cada combinação dos atributos de $A_{\neg\Gamma}$. Portanto os nós de todos os subgrafos clones podem ser representados por $\mathbf{Dom}(A'_1) \times ... \times \mathbf{Dom}(A'_k) \times \mathbf{Dom}(A''_1) \times ... \times \mathbf{Dom}(A''_m) \cup \{vr\}$, onde $\mathbf{Attr}(\Gamma) = \{A'_1, ..., A'_k\}$ e $A_{\neg\Gamma} = \{A''_1, ..., A''_m\}$. Como $A_{\neg\Gamma} = \mathbf{Attr}(R) - \mathbf{Attr}(\Gamma)$, então os nós de todos os subgrafos clones podem ser representados por $\mathbf{Dom}(A_1) \times ... \times \mathbf{Dom}(A_n) \cup \{vr\}$. Logo, $V = V_1 \cup .. \cup V_p$.

Vamos provar que $E = E_1 \cup .. \cup E_p$. Sabemos que $E = E' \cup E''$, então temos que provar que $E' = E'_1 \cup .. \cup E'_p$ e $E'' = E''_1 \cup .. \cup E''_p$. Sabemos que $E' = \{(t,t') \mid t >_{\varphi_i} t'$ e $t,t' \in \mathcal{O}$ e $\varphi_i \in \Gamma\}$ e $E'' = \{(vr,t) \mid t \in \mathcal{O}$ e $\nexists t' \in \mathcal{O}$ tal que $(t',t) \in E'\}$, como $\mathcal{O} = V - \{vr\}$, temos que $E' = \{(t,t') \mid t >_{\varphi_i} t'$ e $t,t' \in (V - \{vr\})$ e $\varphi_i \in \Gamma\}$ e $E'' = \{(vr,t) \mid t \in (V - \{vr\})$ e $\nexists t' \in (V - \{vr\})$ tal que $(t',t) \in E'\}$. Para os subgrafos clones temos que $E'_j = \{(t,t') \mid t >_{\varphi_i} t'$ e $t,t' \in (V_j - \{vr\})$ e $\varphi_i \in \Gamma\}$ e $E''_j = \{(vr,t) \mid t \in (V_j - \{vr\})$ e $\nexists t' \in (V_j - \{vr\})$ tal que $(t',t) \in E'_j\}$. Como provamos que $V = V_1 \cup .. \cup V_p$, por indução temos que $E' = \bigcup_{1 \leq j \leq p} E'_j$ e $E'' = \bigcup_{1 \leq j \leq p} E''_j$ para todo $j \in \{1, ..., p\}$.

O grafo BTG_{Γ_3} gerado a partir da união dos subgrafos clones para teoria-pc do Exemplo 3.4 é apresentado na Figura 3.11.

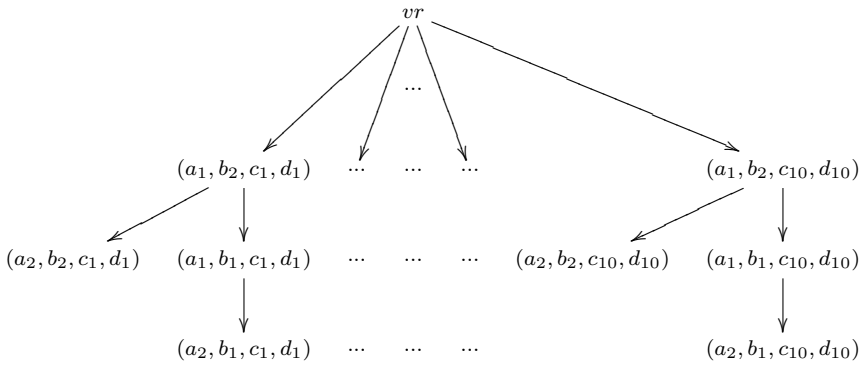


Figura 3.11: Grafo BTG_{Γ_3} gerado a partir dos clones de $SBTG_{\Gamma_3}$

Pelo Lema 3.3, o grafo BTG_Γ gerado a partir dos clones de $SBTG_\Gamma$ é idêntico ao grafo BTG_Γ construído de maneira convencional. Porém em uma implementação prática não é necessário construir todos os clones, podendo ser construído apenas o subgrafo $SBTG_\Gamma$ e na comparação de tuplas verificar se os atributos de $A_{\neg\Gamma}$ possuem valores idênticos.

A grande vantagem desta otimização está no fato de que o subgrafo $SBTG_\Gamma$ considera apenas os atributos A_Γ e apenas os valores que aparecem em Γ para os atributos A_Γ. Isto acontece porque as regras-pc consideradas neste trabalho sempre têm o formato $u \rightarrow (X = x) > (X = x')$.

Outro tipo otimização que deixamos como trabalho futuro é a redução do número de nós do grafo BTG considerado o conceito de dependência preferencial entre os atributos de uma teoria-pc. Por exemplo, vamos considerar os atributos A, B, C e D com seus respectivos domínios $\mathbf{Dom}(A) = \{a_1, a_2\}$, $\mathbf{Dom}(B) = \{b_1, b_2\}$, $\mathbf{Dom}(C) = \{c_1, c_2\}$ e $\mathbf{Dom}(D) = \{d_1, d_2\}$ e a teoria-pc

$$\Gamma_2 = \left\{ \begin{array}{l} \varphi_1 : (A = a_1) > (A = a_2), \\ \varphi_2 : (A = a_1) \rightarrow (B = b_1) > (B = b_2), \\ \varphi_3 : (C = c_1) > (C = c_2), \\ \varphi_4 : (C = c_1) \rightarrow (D = d_1) > (D = d_2) \end{array} \right\}.$$

Podemos notar que os atributos A e B são preferencialmente independentes de C e D através do grafo de dependência preferencial $G(\Gamma_2)$ exibido na Figura 3.12 que possui dois componentes disjuntos (um com os atributos A e B e outro com os atributos C e D).

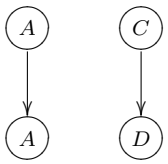


Figura 3.12: Grafo de dependência preferencial $G(\Gamma_2)$

O grafo BTG_{Γ_2} original possui dezesseis nós, mas podemos construir dois grafos BTG com quatro nós cada, um grafo $BTG_{\Gamma_2,A,B}$ considerando os atributos A e B e outro grafo $BTG_{\Gamma_2,C,D}$ considerando os atributos C e D. Podemos intuitivamente imaginar que para que uma tupla t seja preferida a uma tupla t' deve existir um caminho de t para t' tanto em $BTG_{\Gamma_2,A,B}$ quanto em $BTG_{\Gamma_2,C,D}$. Entretanto, é necessário verificar se esta propriedade é válida e se o custo para comparar tuplas desta maneira compensa a redução do número de nós do grafo BTG original.

3.6 Considerações Finais

Neste capítulo foram definidos os operadores **Best-E**, **Best-N** e **Best-D** que selecionam as tuplas de uma relação considerando um conjunto de preferências condicionais representadas através de uma teoria-pc, sendo que cada um destes operadores considera as preferências de uma maneira diferente ao selecionar as tuplas.

No caso do operador **Best-D** foi proposto o conceito de listas de escopos para que as tuplas de uma relação possam ser comparadas diretamente segundo as preferências de uma teoria-pc. A obtenção da listas de escopos é feita através de estruturas de grafos e árvores construídas a partir das preferências e do esquema da relação. Também neste capítulo, foi proposta uma otimização para a obtenção das listas de escopos.

Capítulo 4

Linguagem CPref-SQL

A linguagem *CPref-SQL* é uma extensão da linguagem SQL padrão que permite realizar consultas contendo preferências condicionais. A grande vantagem da linguagem *CPref-SQL* é a possibilidade de expressar preferências condicionais de maneira intuitiva através de um conjunto de regras de preferência, diferente da linguagem *Preference SQL* (onde as preferências têm que ser construídas através de diversos operadores) e do trabalho de [Chomicki 2003] (onde as preferências são expressas através de fórmulas lógicas de primeira ordem). Um bloco de consulta na linguagem *CPref-SQL* tem a seguinte forma:

```
SELECT <lista de atributos>
FROM <lista de relações>
WHERE condições (restrições hard)
ACCORDING TO [EXACT [NO EMPTY ]]
PREFERENCES regras de preferências (restrições soft)
GROUP BY atributos
HAVING condições de agregação
ORDER BY atributos
```

As preferências condicionais são incorporadas a uma consulta através da cláusula *ACCORDING TO PREFERENCES* e são expressas através de uma teoria-pc consistente seguindo as seguintes conversões:

- As regras-pc são conectadas através do conectivo *AND*;
- Uma regra-pc com condição vazia do tipo $(A = a_1) > (A = a_2)$ não sofre nenhuma conversão;
- Uma regra-pc com condição não vazia do tipo $(A_1 = a_1 \wedge ... \wedge A_n = a_n) \rightarrow (B = b_1) > (B = b_2)$ é convertida para *IF* $A_1 = a_1$ *AND ... AND* $A_n = a_n$ *THEN* $(B = b_1) > (B = b_2)$.

Uma consulta *CPref-SQL* contendo preferências condicionais é resolvida com a utilização dos operadores **Best-E**, **Best-N** e **Best-D**, o operador é definido de acordo com

as cláusulas *EXACT NO EMPTY* posicionadas imediatamente após as cláusulas *ACCORDING TO*. Quando é informada apenas a cláusula *EXACT*, o operador **Best-E** é utilizado para resolver a consulta, já as palavras *EXACT NO EMPTY* fazem com que o operador **Best-N** seja usado. Na ausência da cláusulas *EXACT NO EMPTY* o operador **Best-D** é adotado.

4.1 Exemplos de Consultas

Considerando a relação musicas sobre o esquema M (NOME, CANTOR, RITMO, DURACAO, EPOCA) e a relação cantores sobre o esquema C (CANTOR, GENERO) exibidas nas Figuras 4.1 e 4.2, respectivamente, vamos mostrar alguns exemplos de consultas na linguagem *CPref-SQL*.

	NOME	CANTOR	RITMO	DURACAO	EPOCA
t_1	Dias Melhores	Jota Quest	tranqüilo	longa	contemporânea
t_2	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
t_3	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
t_4	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
t_5	Hands Clean	Alanis Morissette	tranqüilo	media	contemporânea
t_6	Immortality	Celine Dion	tranqüilo	media	contemporânea
t_7	I could fall in love	Celine Dion	tranqüilo	breve	contemporânea
t_8	I'll try	Alan Jackson	tranqüilo	breve	contemporânea

Figura 4.1: Relação musicas sobre o esquema M (NOME, CANTOR, RITMO, DURACAO, EPOCA)

	CANTOR	GENERO
t_1	Jota Quest	rock
t_2	Alanis Morissette	pop
t_3	Sérgio Reis	sertanejo
t_4	Gino e Geno	sertanejo
t_5	Celine Dion	pop
t_6	Alan Jackson	country

Figura 4.2: cantores sobre o esquema C (CANTOR, GENERO)

O Exemplo 4.1 mostra uma consulta *CPref-SQL* correspondente a uma teoria-pc, o Exemplo 4.2 exibe uma consulta *CPref-SQL* aninhada e o Exemplo 4.3 mostra uma consulta utilizando funções de agregação.

Exemplo 4.1. Neste exemplo vamos mostrar como escrever uma consulta *CPref-SQL* a partir de uma teoria-pc, para isto considere a teoria-pc $\Gamma = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde:

- φ_1 : (GENERO = 'sertanejo') > (GENERO = 'rock');
- φ_2 : (GENERO = 'rock') > (GENERO = 'pop');
- φ_3 : (EPOCA = 'antiga') > (EPOCA = 'contemporânea');
- φ_4 : (GENERO = 'rock' $\wedge$ EPOCA = 'antiga') $\rightarrow$ (DURACAO = 'longa') > (DURACAO= 'breve')

Considerando as relações musicas e cantores podemos escrever a seguinte consulta *CPref-SQL* correspondente a teoria-pc Γ:

```
SELECT *
FROM musicas, cantores
WHERE musicas.CANTOR = cantores.CANTOR
ACCORDING TO EXACT PREFERENCES
(GENERO = 'sertanejo') > (GENERO = 'pop') AND
(GENERO = 'rock') > (GENERO = 'pop') AND
(EPOCA = 'antiga') > (EPOCA = 'contemporânea') AND
IF GENERO = 'rock' AND EPOCA = 'antiga' THEN
(DURACAO = 'longa') > (DURACAO= 'breve')
```

Como foi especificada a cláusula *EXACT* é usado o operador **Best-E** para resolver a consulta e as seguintes tuplas são retornadas:

- (Menino da Porteira, Sérgio Reis, tranqüilo, longa, antiga, sertanejo)
- (As Águas do São Francisco, Gino e Geno, tranqüilo, longa, antiga, sertanejo)

Exemplo 4.2. Supondo que um usuário deseje selecionar as músicas da relação musicas de acordo as seguintes preferências sobre os cantores da relação cantores:

- Os gêneros sertanejo e rock são preferidos ao gênero pop;
- Se o gênero for sertanejo o cantor "Gino e Geno" é preferido ao cantor "Sérgio Reis";
- Se o gênero for rock a cantora "Alanis Morissette" é preferida à cantora "Celine Dion".

Para selecionar as tuplas desejadas pelo usuário podemos utilizar uma consulta aninhada contendo as preferências sobre a relação C1 e usar o resultado desta consulta para selecionar as tuplas da relação M3, chegando a seguinte consulta:

```
SELECT *
FROM musicas
WHERE CANTOR IN (

     SELECT CANTOR
     FROM cantores
     ACCORDING TO EXACT PREFERENCES
     (GENERO = 'sertanejo') > (GENERO = 'pop') AND
     (GENERO = 'rock') > (GENERO = 'pop') AND
     IF GENERO = 'sertanejo' THEN
     (CANTOR = 'Gino e Geno') > (CANTOR = 'Sérgio Reis')
     IF GENERO = 'pop' THEN
     (CANTOR = 'Alanis Morissette') > (CANTOR = 'Celine Dion')

)
```

As tuplas retornadas por esta consulta são as tuplas t_1, t_3, t_4, t_5, t_8 da relação M3.

Exemplo 4.3. Supondo que um usuário deseja selecionar os cantores da relação M3 que possuam mais de uma música atendendo às seguintes preferências:

- Ritmo tranqüilo é preferido a ritmo agitado;
- Se o ritmo for tranqüilo então época contemporânea é melhor do que época antiga;
- Se o ritmo for agitado então época antiga é melhor do que época contemporânea.

Para selecionar as tuplas desejadas pelo usuário podemos utilizar a seguinte consulta:

```
SELECT CANTOR, COUNT(*)
FROM musicas
ACCORDING TO PREFERENCES
(RITMO = 'tranqüilo') > (RITMO = 'agitado')
IF RITMO = 'tranqüilo' THEN (EPOCA = 'antiga') > (EPOCA = 'contemporânea') AND
IF RITMO = 'agitado' THEN (EPOCA = 'contemporânea') > (EPOCA = 'antiga')
GROUP BY CANTOR
HAVING COUNT(*) > 1
```

A única tupla retorna da por esta consulta é a tupla ('Celine Dion', 2).

4.2 Processamento de Consultas na Linguagem CPref-SQL

Os planos de execução de consultas *CPref-SQL* são semelhantes aos planos de execução de consultas SQL padrão, sendo que os operadores **Best-E**, **Best-N** e **Best-D** são executados após a seleção e antes da projeção. Um plano de execução para uma consulta *CPref-SQL* básica (sem funções de agregação) é exibido na Figura 4.3. Os símbolos π, σ e $\bowtie$ denotam os operadores comuns da álgebra relacional *projeção*, *seleção* e *junção*, respectivamente.

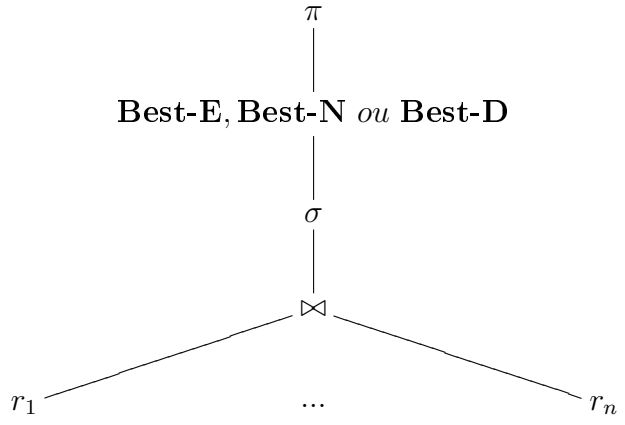


Figura 4.3: Plano de execução de uma consulta *CPref-SQL* básica

Quando os operadores **Best-E**, **Best-N** ou **Best-D** possuem preferências sobre atributos de apenas um esquema de relação, tais operadores podem ser resolvidos antes da junção para que o plano de execução tenha um desempenho melhor.

A utilização da propriedade anterior e de outras propriedades depende de análises sobre os números de índice dos atributos envolvidos, do tipo de algoritmo que os operadores foram implementados dentre outros fatores. Porém, como foi mencionado anteriormente este trabalho não abrange detalhes sobre otimização de execução de consultas e não serão abordados fatores como estes.

4.3 Considerações Finais

Neste capítulo foi especificada a sintaxe da linguagem de consulta para banco de dados *CPref-SQL*. A semântica da linguagem *CPref-SQL* foi ilustrada informalmente através de diversos exemplos de consultas.

Capítulo 5

Algoritmos e Implementação

Neste capítulo propomos os algoritmos **GetBest-E**, **GetBest-N** e **GetBest-D** para resolver os operadores de seleção de tuplas ótimas **Best-E**, **Best-N** e **Best-D**, respectivamente. Também neste capítulo são apresentados detalhes sobre a implementação de um protótipo para um fragmento da linguagem *CPref-SQL*. Além disto são exibidos resultados sobre alguns testes de desempenho realizados através do protótipo implementado.

5.1 Algoritmo GetBest-E

O algoritmo **GetBest-E** recebe como entrada uma relação r, uma teoria-pc Γ consistente e retorna as tuplas ótimas absolutas de acordo com Γ que estão presentes em r. O esquema do algoritmo **GetBest-E** é mostrado na Figura 5.1.

Figura 5.1: Esquema do algoritmo **GetBest-E**

Na Figura 5.2 é apresentado o pseudo-código para o algoritmo **GetBest-E**.

GetBest-E(r, Γ) :

1: $B =$ **BuildBest**(Γ) // Constrói as as tuplas ótimas absolutas
2: $S = select(r, B)$ // Obtém as tuplas ótimas presentes em r
3: **return** S

Figura 5.2: Algoritmo **GetBest-E**

O algoritmo **GetBest-E** utiliza a subrotina **BuildBest** para construir todas as tuplas ótimas absolutas de acordo com Γ e então verifica a existência destas tuplas na relação r.

A complexidade do algoritmo **GetBest-E** é de $O(n^3 + nm)$, onde $n = |\Gamma|$ (é o numero de regras da teoria-pc) e $m = |r|$ (é o número de tuplas da relação). Isto acontece porque

a complexidade da rotina **BuildBest** é de $O(n^3)$ (como é mostrado na próxima subseção) e o custo da função $select(r, B)$ é de $O(nm)$. A função $select(r, B)$ é responsável por varrer a relação r em busca das tuplas ótimas absolutas, uma teoria-pc com n regras pode gerar até n tuplas ótimas absolutas, portanto é preciso comparar cada uma das m tuplas da relação r com cada uma das n tuplas ótimas absolutas.

No Exemplo 5.1 são exibidas duas execuções para o algoritmo **GetBest-E**.

	NOME	CANTOR	GENERO	RITMO	DURACAO	EPOCA
t_1	As Águas do São Francisco	Gino e Geno	sertanejo	tranqüilo	longa	antiga
t_2	Menino da Porteira	Sérgio Reis	sertanejo	tranqüilo	longa	antiga
t_3	Panela Velha	Sérgio Reis	sertanejo	movimentado	breve	antiga
t_4	I could fall in love	Celine Dion	pop	tranqüilo	breve	contemporânea
t_5	I'll try	Alan Jackson	country	tranqüilo	breve	contemporânea
t_6	Mamma	Luciano Pavarotti	clássico	tranqüilo	longa	antiga
t_7	Canzoni Stonali	Andrea Bocelli	clássico	tranqüilo	longa	contemporânea
t_8	Dias Melhores	Jota Quest	rock	tranqüilo	longa	contemporânea
t_9	Dani Califórnia	Red Hot Chili Peppers	rock	movimentado	breve	contemporânea
t_{10}	Mentiras	Adriana Calcanhoto	mpb	tranqüilo	breve	contemporânea
t_{11}	Sossego	Tim Maia	mpb	tranqüilo	breve	antiga
t_{12}	Brasil	Antonio Carlos Jobim	mpb	tranqüilo	longa	antiga
t_{13}	Mal de Mim	Djavan	mpb	tranqüilo	breve	contemporânea
t_{14}	Pinga em Mim	Sérgio Reis	sertanejo	movimentado	longa	antiga
t_{15}	Apaixonado por você	Gino e Geno	sertanejo	movimentado	breve	contemporânea
t_{16}	Blusa Amarela	Gino e Geno	sertanejo	tranqüilo	breve	antiga

Figura 5.3: Relação musicas sobre o esquema M4 (NOME, CANTOR, GENERO, RITMO, DURACAO, EPOCA)

Exemplo 5.1. Levando em consideração a teoria-pc

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : (DURACAO = \text{`longa'}) \rightarrow (EPOCA = \text{`antiga'}) > (EPOCA = \text{`contemporânea'}), \\ \varphi_2 : (GENERO = \text{`sertanejo'}) > (GENERO = \text{`pop'}), \\ \varphi_3 : (GENERO = \text{`rock'}) > (GENERO = \text{`pop'}), \\ \varphi_4 : (GENERO = \text{`sertanejo'}) \rightarrow (DURACAO = \text{`longa'}) > (DURACAO = \text{`breve'}), \\ \varphi_5 : (GENERO = \text{`rock'}) \rightarrow (DURACAO = \text{`breve'}) > (DURACAO = \text{`longa'}) \end{array} \right\},$$

e a relação musicas sobre o esquema M4 (NOME, CANTOR, GENERO, RITMO, DURACAO, EPOCA) exibida na Figura 5.3, o algoritmo **GetBest-E** é executado da seguinte maneira:

- No primeiro passo são construídas as tuplas ótimas absolutas pelo algoritmo **BuildBest**,
$$B = \left\{ \begin{array}{l} (GENERO, \text{`sertanejo'}), (DURACAO, \text{`longa'}), (EPOCA, \text{`antiga'}), \\ (GENERO, \text{`rock'}), (DURACAO, \text{`breve'}) \end{array} \right\};$$
- Em seguida o algoritmo verifica quais destas tuplas estão presentes na relação musicas;
- Somente as tuplas t_1, t_2, t_3, t_9 e t_{14} são retornadas.

Considerando agora a teoria-pc

$$\Gamma' = \left\{ \begin{array}{l} \varphi_1 : (GENERO = \text{`pop'}) > (GENERO = \text{`sertanejo'}), \\ \varphi_2 : (GENERO = \text{`sertanejo'}) \rightarrow (DURACAO = \text{`breve'}) > (DURACAO = \text{`longa'}), \\ \varphi_3 : (GENERO = \text{`pop'}) \rightarrow (DURACAO = \text{`longa'}) > (DURACAO = \text{`breve'}) \end{array} \right\}$$

e novamente a relação musicas, a execução do algoritmo **GetBest-E** é feita como se segue:

- No primeiro passo são construídas as tuplas ótimas absolutas, $B = \{(GENERO, \text{'pop'}), (DURACAO, \text{'longa'})\}$;
- Em seguida o algoritmo verifica quais destas tuplas estão presentes na relação musicas;
- A relação musicas não possui nenhuma das tuplas ótimas, então o algoritmo retorna vazio;

5.1.1 Subrotina BuildBest

A subrotina **BuildBest** é um algoritmo baseado no trabalho de [Wilson 2004b] que constrói as tuplas ótimas absolutas a partir de uma teoria-pc Γ consistente. A construção das tuplas ótimas considera os valores dos atributos informados nas regras de preferência contidas na teoria-pc. As tuplas ótimas construídas pelo algoritmo **BuildBest** possuem a forma de conjuntos de atribuições (A, a), onde A é o atributo e a é o valor para o atributo. A subrotina **BuildBest** é exibida na Figura 5.4.

BuildBest(Γ) :

```
1:  S = ∅
2:  Γ' = ∅
3:  change = true
4:  while change do                          // Enquanto houver regras a se considerar
5:      change = false
6:      for all φ_i ∈ Γ do
7:          if ConsiderateRule(S, φ_i) then   // Verifica se a regra deve ser considerada
8:              Γ' = Γ' ∪ {φ_i}
9:              Γ = Γ − {φ_i}
10:         end if
11:     end for
12:     if Γ' ≠ ∅ then
13:         change = true
14:         S = UpdateBest(S, Γ')             // Atualiza as tuplas ótimas absolutas
15:         Γ' = ∅
16:     end if
17: end while
18: return S
```

Figura 5.4: Algoritmo **BuildBest**

A subrotina **ConsiderateRule** verifica se uma regra-pc φ é valida, isto ocorre quando φ possui uma condição vazia ou quando existe uma tupla em S que satisfaz a condição de φ. Já a subrotina **UpdateBest** modifica as tuplas ótimas absolutas de S considerando as

regras-pc contidas em Γ'. As subrotinas **ConsiderateRule** e **UpdateBest** são exibidas nas Figuras 5.5 e 5.6, respectivamente.

ConsiderateRule(S, φ) :

```
1: if EmptyCondition(φ_i) then                    // Verifica se a condição de φ é vazia
2:     return true
3: else
4:     for all s_i ∈ S do
5:         if validate(φ, s_i) then               // Verifica se a condição de φ é validada por s_i
6:             return true
7:         end if
8:     end for
9: end if
10: return false
```

Figura 5.5: Algoritmo **ConsiderateRule**

UpdateBest(S, Γ) :

```
1: for all φ_i ∈ Γ do
2:     v = bestValue(φ_i)                         // Obtém o melhor valor imposto pela regra φ_i
3:     if S = ∅ then
4:         S = {{(X_φi, v)}}
5:     else
6:         S' = S
7:         for all s_j ∈ S' do
8:             if validate(φ_i, s_j) then         // Verifica se s_j valida a condição de φ_i
9:                 S = S − {s_j}
10:                if X_φi ∈ Attr(s_j) then       // Verifica se s_j possui atribuição para X_φi
11:                    s'_j = s_j − {(X_φi, value(s_j, X_φi))}
12:                    s'_j = s'_j ∪ {(X_φi, v)}
13:                    S = S ∪ {s'_j}
14:                else
15:                    s_j = s_j ∪ {(X_φi, v)}
16:                end if
17:                S = S ∪ {s_j}
18:            end if
19:        end for
20:    end if
21: end for
22: return S
```

Figura 5.6: Algoritmo **UpdateBest**

Para melhor entendimento da subrotina **BuildBest** uma execução passo a passo do mesmo é descrita no Exemplo 5.2.

Exemplo 5.2. Supondo a teoria-pc

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : (D = l) \rightarrow (E = a) > (E = c), \\ \varphi_2 : (G = s) > (G = p), \\ \varphi_3 : (G = r) > (G = p), \\ \varphi_4 : (G = s) \rightarrow (D = l) > (D = b), \\ \varphi_5 : (G = r) \rightarrow (D = b) > (D = l) \end{array} \right\},$$

na qual G, E e D representam os atributos Gênero, Época e Duração e l, b, a, c, s, r e p, representam os valores longa, breve, antiga, contemporânea, sertanejo, rock e pop, respectivamente. O algoritmo **BuildBest** é executado da seguinte maneira:

- Primeira repetição do laço principal:
  - Regras a serem consideradas: φ_2 e φ_3;
  - Atualização das condições:
    - $S = \{s_1, s_2\}$, onde $s_1 = \{(G, s)\}$ e $s_2 = \{(G, r)\}$
- Segunda repetição do laço principal:
  - Regras a serem consideradas: φ_4 e φ_5;
  - Atualização das condições:
    - $S = \{s_1, s_2\}$, onde $s_1 = \{(G, s), (D, l)\}$ e $s_2 = \{(G, r), (D, b)\}$
- Terceira repetição do laço principal:
  - Regras a serem consideradas: φ_1;
  - Atualização das condições:
    - $S = \{s_1, s_2\}$, onde $s_1 = \{(G, s), (D, l), (E, a)\}$ e $s_2 = \{(G, r), (D, b)\}$
- Não há mais alterações de condições;
- Retorna $S = \{\{(G, s), (D, l), (E, a)\}, \{(G, r), (D, b)\}\}$

É importante notar que se a subrotina **BuildBest** deve receber uma teoria-pc sem regras-pc com condição vazia ou nenhuma tupla ótima absoluta será retornada. Por exemplo, se consideramos a teoria-pc

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : (G = s) \rightarrow (D = l) > (D = b), \\ \varphi_2 : (D = l) \rightarrow (E = a) > (E = c) \end{array} \right\},$$

a regra-pc φ_1 nunca será atendida pois sua condição depende do atributo A que não possui nenhuma preferência especificada. É preciso pesquisar um solução para a subrotina **BuildBest** tratar este tipo de teoria-pc.

5.2 Algoritmo GetBest-N

O algoritmo **GetBest-N** recebe uma relação r, uma teoria-pc Γ consistente e retorna as tuplas ótimas absolutas de acordo com Γ ou a própria relação r se nenhuma das tuplas

ótimas absolutas existir em r. O esquema do algoritmo **GetBest-N** é mostrado na Figura 5.7.

Figura 5.7: Esquema do algoritmo **GetBest-N**

O algoritmo **GetBest-N** utiliza o algoritmo **GetBest-E** como uma subrotina, se o algoritmo **GetBest-E** retornar vazio então o algoritmo **GetBest-N** retorna todas as tuplas da relação de entrada. Na Figura 5.8 é apresentado o pseudo-código para o algoritmo **GetBest-N**.

GetBest-N(r, Γ) :

```
1: B = GetBest-E(r, Γ)          // Obtém as tuplas ótimas em r de acordo com Γ
2: if B = ∅ then
3:     return r
4: else
5:     return B
6: end if
```

Figura 5.8: Algoritmo **GetBest-N**

A complexidade do algoritmo **GetBest-N** é de $O(n^3 + nm)$, já que este algoritmo pode ser visto como uma variação do algoritmo **GetBest-E** onde é feita apenas uma varredura a mais na relação quando nenhuma tupla ótima absoluta é encontrada.

No Exemplo 5.3 é exibida uma execução mais detalhada do algoritmo **GetBest-N**.

Exemplo 5.3. Levando em consideração a teoria-pc Γ do Exemplo 5.1 e a relação musicas exibida na Figura 5.3, o algoritmo **GetBest-N** é executado da seguinte maneira:

- As tuplas ótimas de acordo com Γ são obtidas através do algoritmo **GetBest-E**;
- Como o algoritmo **GetBest-E** é usado, são retornadas as tuplas t_1, t_2, t_3, t_9 e t_{14}, (conforme o Exemplo 5.1);

Se for considerada a teoria-pc Γ' do Exemplo 5.1, o algoritmo detecta que nenhuma das tuplas ótimas está presente na relação musicas e retorna todas as tuplas desta relação;

5.3 Algoritmo GetBest-D

O algoritmo **GetBest-D** baseia-se no algoritmo *block-nested-loops (BNL)* proposto no trabalho de [Börzsönyi et al. 2001] e utiliza o conceito de listas de escopos apresentado no

Capítulo 3 para realizar comparações entre tuplas. O algoritmo **GetBest-D** recebe como entrada uma relação r, uma teoria-pc Γ consistente e retorna as tuplas dominantes em r de acordo com Γ. O esquema da Figura 5.9 expõe a idéia geral do algoritmo **GetBest-D**.

Figura 5.9: Esquema do algoritmo **GetBest-D**

O pseudo-código para algoritmo **GetBest-D** é exibido na Figura 5.10, podemos observar que é utilizada a subrotina **GetScopeLists** (que será apresentada mais adiante) para obter as listas de escopos. Esta rotina não possui a otimização discutida no Capítulo 3, mas como foi mencionado anteriormente, em uma implementação prática tal otimização pode ser praticada para uma maior eficiência.

GetBest-D(r, Γ) :

```
1:  W = ∅
2:  L = GetScopeLists(r, Γ)                          // Obtém as listas de escopos
3:  for all t ∈ r do                                 // Laço para cada tupla da relação
4:      if W = ∅ then
5:          W = {t}
6:      else
7:          l = scopeList(t, L)                      // Obtém a lista de escopos associada a tupla t
8:          ignore = false
9:          for all t' ∈ W do                        // Laço para cada tupla da solução temporária
10:             if (¬ignore) then                    // Verifica se t já foi dominada por uma tupla de W
11:                 l' = scopeList(t', L)
12:                 if (subsume(l', l)) then         // Verifica se t' domina t
13:                     ignore = true                // Ignora a tupla t
14:                 else if subsume(l, l') then
15:                     W = W − {t'}                 // Remove a tupla t' da solução temporária
16:                 end if
17:             end if
18:         end for
19:         if (¬ignore) then                        // Verifica se t deve ser ignorada
20:             W = W ∪ {t}
21:         end if
22:     end if
23: end for
24: return W
```

Figura 5.10: Algoritmo **GetBest-D**

O algoritmo **GetBest-D** possui a complexidade de $O(nm^2 + m^7)$, pois a subrotina **GetScopeLists** possui esta complexidade que é a operação de maior custo realizada pelo algoritmo.

Uma execução do algoritmo **GetBest-D** é dada pelo Exemplo 5.4.

	GENERO	EPOCA	DURACAO
t_1	pop	contemporânea	breve
t_2	rock	antiga	longa
t_3	rock	contemporânea	longa
t_4	rock	contemporânea	breve
t_5	pop	contemporânea	longa
t_6	sertanejo	contemporânea	longa
t_7	pop	antiga	breve
t_8	pop	antiga	longa
t_9	sertanejo	contemporânea	breve
t_{10}	sertanejo	antiga	breve

Figura 5.11: Relação musicas sobre o esquema M5 (GENERO, RITMO, DURACAO, EPOCA)

Exemplo 5.4. A execução do algoritmo **GetBest-D** recebendo como entrada a teoria-pc

$$\Gamma = \left\{ \begin{array}{l} \varphi_1 : (GENERO = \text{`sertanejo'}) > (GENERO = \text{`rock'}) \\ \varphi_2 : (EPOCA = \text{`antiga'}) > (EPOCA = \text{`contemporânea'}) \\ \varphi_3 : (GENERO = \text{`sertanejo'}) \rightarrow (DURACAO = \text{`longa'}) > (DURACAO = \text{`breve'}) \\ \varphi_4 : (GENERO = \text{`rock'} \wedge E = \text{`contemporânea'}) \rightarrow (DURACAO = \text{`breve'}) > (DURACAO = \text{`longa'}) \end{array} \right\}$$

e a relação musicas sobre o esquema M5 (GENERO, RITMO, DURACAO, EPOCA) exibida na Figura 5.11 é realizada conforme os passos seguintes:

- Inicialmente o conjunto de tuplas W encontra-se vazio;
- A subrotina **GetScopeLists** obtém as listas de escopos;
- A relação musicas é varrida e suas tuplas são comparadas usando suas listas de escopos;
  - A tupla t_1 é adicionada a W;
  - A t_2 que possui a lista de escopos ($[7, 7]$) não domina t_1 presente em W (que possui a lista de escopos ($[11, 11]$)), portanto t_2 é adicionada a W;
  - A tupla t_3 que possui a lista de escopos ($[1, 1]$) é adicionada a W;
  - A tupla t_4 que possui a lista de escopos ($[1, 2]$) domina a tupla t_3, portanto t_3 é removida de W e t_4 é adicionada a W;
  - A tupla t_5 que possui a lista de escopos ($[9, 9]$) é adicionada a W;
  - A tupla t_6 que possui a lista de escopos ($[1, 4]$) domina a tupla t_4, portanto t_4 é removida de W e t_6 é adicionada a W;
  - A tupla t_7 que possui a lista de escopos ($[11, 12]$) domina a tupla t_1, portanto t_1 é removida de W e t_7 é adicionada a W;
  - A tupla t_8 que possui a lista de escopos ($[9, 10]$) domina a tupla t_5, portanto t_5 é removida de W e t_8 é adicionada a W;
  - A tupla t_9 que possui a lista de escopos ($[1, 3]$) é dominada por t_6 e portanto é descartada;
  - A tupla t_{10} que possui a lista de escopos ($[5, 6]$, **[1,4]**) é adicionada a W;

- O algoritmo retorna as tuplas t_2, t_6, t_7 e t_{10}.

5.3.1 Algoritmos para Obtenção das Listas de Escopos

A subrotina **GetScopeLists** utilizada pelo algoritmo **GetBest-D** é um algoritmo que recebe como entrada uma teoria-pc consistente e retorna um conjunto contendo triplas do tipo $(t, [i, j], ei)$, onde t é uma tupla, $[i, j]$ é um escopo de t e ei indica o tipo de escopo, 1 para principal e 2 para secundário.

A Figura 5.12 exibe a subrotina **GetScopeLists**.

GetScopeLists(r, Γ) :

1: $BTG_\Gamma =$ **BuildBTG**(r, Γ) // Constrói o grafo BTG
2: $T_\Gamma =$ **OptimumSpanningTree**(BTG_Γ) // Constrói a árvore de cobertura mínima ótima
3: $vr = root(T_\Gamma)$ // Obtêm a raiz de T_Γ
4: $L =$ **GetMainScopes**$(T_\Gamma, vr, 1, \emptyset)$ // Obtém os escopos principais
5: $V_{BTG_\Gamma} = ReverseTopSort(V_{BTG_\Gamma}, BTG_\Gamma)$ // Ordena os nós de BTG_Γ em sua ordem topológica reversa
6: **for all** $t' \in V_{BTG_\Gamma}$ **do** // Laço seguindo a ordem topológica reversa de BTG_Γ
7: **for all** $(t, t') \in E_{BTG_\Gamma} \mid (t, t') \notin E_{T_\Gamma}$ **do** // Arestas de BTG_Γ não consideradas em T_Γ
8: $L = \{(t, 2, [i, j]) \mid (t', 1, [i, j] \in L)\}$ // Obtém os escopos secundários
9: $L = L \cup \{(t, 2, [i, j]) \mid (t', 2, [i, j] \in L)\}$
10: **for all** $(t, x, [i, j]), (t, y, [i', j']) \in L$ **do** // Para cada par de escopos de t
11: **if** $i \leq i'$ **and** $j' \leq j$ **then**
12: $L = L - \{(t, y, [i', j'])\}$
13: **else if** $i' \leq i$ **and** $j \leq j'$ **then**
14: $L = L - \{(t, x, [i, j])\}$
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: **return** L

Figura 5.12: Algoritmo **GetScopeLists**

A subrotina **BuildBTG** é responsável por construir o grafo BTG, seu pseudo-código é exibido na Figura 5.13. É importante ressaltar que a rotina **BuildBTG** constrói o grafo BTG tendo como nós apenas as tuplas da relação r (além do nó vr). Isto não afeta o conceito de grafo BTG dado no Capítulo 3, mas é importante para que seja consumido menos espaço para armazenamento do grafo.

A subrotina **OptimumSpanningTree** é um algoritmo que foi proposto no trabalho de [Agrawal et al. 1989] com o objetivo de construir uma árvore de cobertura mínima ótima para um grafo BTG, seu pseudo-código é exibido na Figura 5.14.

A subrotina **GetMainScopes** exibida na Figura 5.15 é responsável por obter os escopos principais de um nó t e seus descendentes. Esta subrotina recebe uma árvore de

BuildBTG(r, Γ) :

```
1:  O = buildObjets(r)          // Constrói o conjunto de objetos O usando a relação r
2:  V = O ∪ {vr}                // Nós do grafo
3:  E = ∅                       // Arestas do grafo
4:  for all t, t' ∈ V do
5:      for all φ_i ∈ Γ do      // Laço para todas as regras de Γ
6:          if t >_{φ_i} t' then   // Verifica se t é preferido a t' de acordo com φ_i
7:              E = E ∪ {(t, t')}
8:          else if t' >_{φ_i} t then
9:              E = E ∪ {(t', t)}
10:         end if
11:     end for
12: end for
13: for all t ∈ O do            // Adiciona o nó virtual como raiz
14:     if ((t', t) ∉ E | t' ∈ O) then   // Verifica se t não recebe arestas
15:         E = E ∪ {(vr, t)}
16:     end if
17: end for
18: return G = (V, E)           // Retorna grafo com nós e arestas
```

Figura 5.13: Algoritmo **BuildBTG**

OptimumSpanningTree(BTG_Γ) :

```
1:  T_Γ = BTG_Γ                 // Inicialmente a árvore é igual ao grafo BTG
2:  for all t ∈ V_{T_Γ} do
3:      P_t = ∅                 // Inicializa o conjunto de predecessores de cada nó
4:  end for
5:  V_{T_Γ} = TopSort(V_{T_Γ}, T_Γ)   // Ordena os nós de T_Γ em sua ordem topológica
6:  for all t ∈ V_{T_Γ} do      // Laço seguindo a ordem topológica de T_Γ
7:      for all (t', t), (t'', t) ∈ E_{T_Γ} do   // Arestas que chegam em t
8:          if |P_{t'}| > |P_{t''}| then   // Prevalece o maior número de predecessores
9:              E_{T_Γ} = E_{T_Γ} − {(t'', t)}
10:             P_t = P_t ∪ {t'}
11:         else
12:             E_{T_Γ} = E_{T_Γ} − {(t', t)}
13:             P_t = P_t ∪ {t''}
14:         end if
15:     end for
16: end for
17: return I
```

Figura 5.14: Algoritmo **OptimumSpanningTree**

cobertura mínima T, um nó $t \in T$, um número de pós-ordem inicial p, um conjunto de escopos L e trabalha de forma recursiva, parando quando t e todos seus descendentes recebem seus escopos principais.

GetMainScopes(T, t, p, L) :

1: $i = p$ // O número de índice é igual ao menor número de pós-ordem
2: **for all** $(t, t') \in E_T$ **do**
3: $L' =$ **GetMainScopes**(T, t', p, L) // Chamada recursiva para o filho t' de t
4: $p = MaxPost(\{t'' \mid (t, t'') \in E_T\}) + 1$ // O número de pós-ordem de t será igual ao maior número de pós-ordem de seus filhos acrescido de um
5: **end for**
6: $L = L \cup L' \cup \{(t, 1, [i, p])\}$ // Adiciona o escopo de t
7: **return** L

Figura 5.15: Algoritmo **GetMainScopes**

A subrotina mais cara contida dentro da subrotina **GetScopeLists** é a subrotina **BuildBTG**. A complexidade da subrotina **BuildBTG** é de $O(nm^2 + m^4)$. Além disto a subrotina **GetScopeLists** possui três laços para cada nó das estruturas de grafos aninhados com dois laços para cada arestas destas estrutura. O número de vértices é no máximo m (já que podemos construir o grafo BTG contendo apenas as tuplas da relação) e o número de arestas é no máximo m^2. Considerando estes laços temos uma complexidade de $O(m^7)$. Desta maneira, temos que a complexidade da subrotina **GetScopeLists** é de $O(nm^2 + m^7)$.

O Exemplo 5.5 demonstra uma execução mais detalhada para o algoritmo **GetScopeLists**.

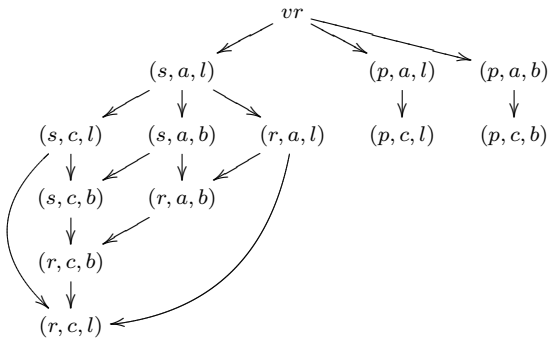
Exemplo 5.5. Neste exemplo vamos considerar a relação musicas sobre o esquema M(GENERO, EPOCA, DURACAO) e a teoria-pc $\Gamma = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde:

- $\varphi_1 : (G = s) > (G = r)$;
- $\varphi_2 : (E = a) > (E = c)$;
- $\varphi_3 : (G = s) \rightarrow (D = l) > (D = b)$;
- $\varphi_4 : (G = r \wedge E = c) \rightarrow (D = b) > (D = l)$;
- $\varphi_5 : (G = r \wedge E = a) \rightarrow (D = l) > (D = b)$;

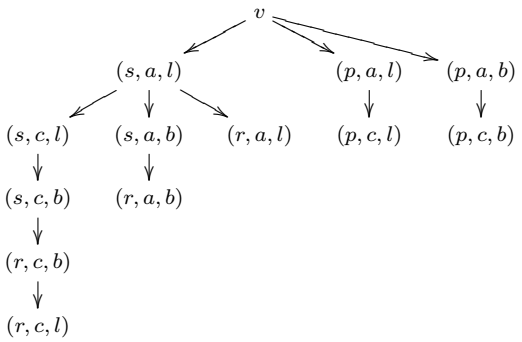
G corresponde ao atributo GENERO e $\mathbf{Dom}(G) = \{s, r, p\}$, D corresponde ao atributo DURACAO e $\mathbf{Dom}(D) = \{l, b\}$, E corresponde ao atributo EPOCA e $\mathbf{Dom}(E) = \{a, c\}$. As letras s, r, p, a, c, l e b correspondem aos valores sertanejo, rock, pop, antiga, contemporânea, longa e breve, respectivamente. Uma execução do algoritmo **GetScopeLists** recebendo como entrada a relação musicas e a teoria-pc Γ e feita da seguinte maneira:

1. A rotina **BuildBTG** constrói o grafo BTG_Γ, apresentado na Figura 5.16(a);

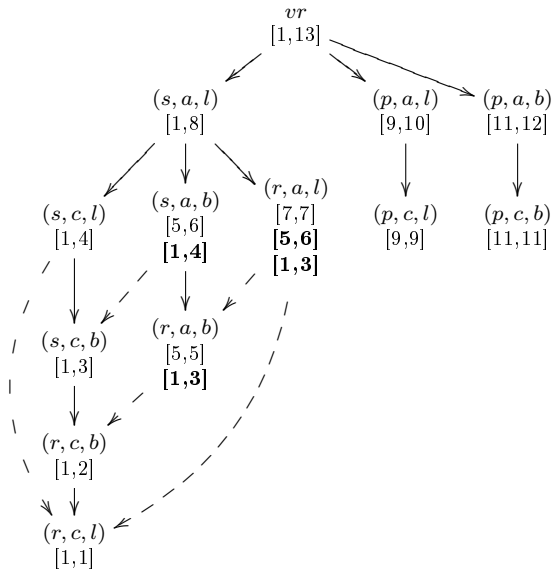
2. A rotina **OptimumSpanningTree** obtém a árvore de cobertura mínima ótima T_Γ para BTG_Γ, que é exibida na Figura 5.16(b);
3. A rotina **GetMainScopes** atribui os escopos principais e o restante do algoritmo atribui os escopos secundários, obtendo as listas de escopos que são mostradas através da Figura 5.16(c).



(a) Grafo BTG_Γ construído pela rotina **BuildBTG**



(b) Árvore de cobertura mínima T_Γ obtida pela rotina **OptimumSpanningTree**



(c) Grafo BTG_Γ com as listas de escopos obtidas pelo algoritmo **GetScopeLists**

Figura 5.16: Exemplo de execução do algorimo **GetScopeLists**

5.4 Implementação do Protótipo CPref-SQL

No decorrer deste trabalho foi implementado um protótipo para um fragmento da linguagem *CPref-SQL* de forma integrada ao Sistema de Gerenciamento de Banco de Dados (SGBD) PostgreSQL. O protótipo é composto por um interpretador e uma biblioteca dinâmica denominada **preflib**.

O interpretador efetua uma conversão sobre uma consulta *CPref-SQL* fazendo com que o SGBD utilize a biblioteca **preflib** para resolver os operadores de seleção de tuplas ótimas. O resultado dos operadores é devolvido ao SGBD que, por sua vez, retorna o resultado da consulta ao interpretador. Este processo de execução de consultas *CPref-SQL* é exibido na Figura 5.17.

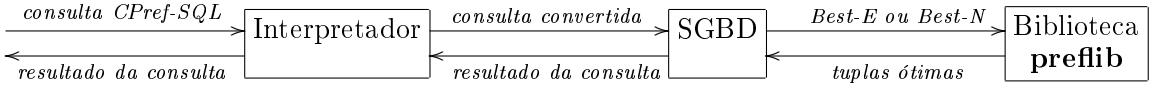


Figura 5.17: Processo de execução de consultas *CPref-SQL*

Quando as consultas não possuem nenhuma preferência o interpretador não faz a conversão e a consulta é executada inteiramente pelo SGBD.

Atualmente os algoritmos **GetBest-E** e **GetBest-N** encontram-se implementados dentro da biblioteca **preflib**.

A linguagem de programação utilizada na implementação do interpretador e da biblioteca **preflib** foi a linguagem C devido a necessidade de compatibilidade com o SGBD PostgreSQL. A escolha pelo SGBD PostgreSQL foi feita principalmente porque o PostgreSQL é um SGBD com diversos recursos presentes em SGBD's comerciais, é um SGBD altamente expansível e também é um software livre.

A implementação do protótipo na forma de um interpretador e de uma biblioteca dinâmica permite que o PostgreSQL seja atualizado sem que mudanças drásticas sejam feitas na implementação.

5.5 Testes com o Protótipo CPref-SQL

Com o objetivo de verificar o desempenho dos algoritmos implementados foram feitos dois tipos de testes. O primeiro teste baseia-se no fato de que os algoritmos **GetBest-E** e **GetBest-N** utilizam a subrotina **BuildBest** para construir as tuplas ótimas absolutas a partir das preferências especificadas. Desta forma podemos buscar as tuplas ótimas absolutas através de uma consulta SQL padrão utilizando *restrições hard*, ou seja, podemos buscar as tuplas ótimas absolutas através de uma consulta SQL padrão contendo as condições necessárias na cláusula WHERE.

Como exemplo, vamos considerar a seguinte consulta *CPref-SQL*:

```
SELECT *
FROM MUSICAS
ACCORDING TO EXACT PREFERENCES
(GENERO = 'sertanejo') > (GENERO = 'pop') AND
(GENERO = 'rock') > (GENERO = 'pop') AND
IF GENERO = 'sertanejo' THEN
(EPOCA = 'antiga') > (EPOCA = 'contemporânea') AND
IF GENERO = 'rock' AND EPOCA = 'contemporânea' THEN
(DURACAO = 'longa') > (DURACAO= 'breve').
```

As condições desejadas para as tuplas ótimas absolutas desta consulta são GENERO = 'sertanejo' e EPOCA = 'antiga' ou GENERO = 'rock' e EPOCA = 'antiga' e DURACAO = 'longa', neste caso as tuplas ótimas podem ser obtidas através da seguinte consulta na linguagem SQL padrão:

```
SELECT *
FROM MUSICAS
WHERE (GENERO = 'sertanejo' AND EPOCA = 'antiga') OR
(GENERO = 'rock' AND EPOCA = 'contemporânea' AND DURACAO = 'longa').
```

O primeiro tipo de teste compara o tempo de execução de uma consulta *CPref-SQL* com o tempo de execução de uma consulta SQL padrão contendo as condições para as tuplas ótimas absolutas. Os resultados retornados por ambas consultas são idênticos, mas a consulta *CPref-SQL* gasta um pouco mais de tempo como mostra a Figura 5.18, isto acontece devido a comunicação entre o SGBD e a biblioteca **preflib** que é necessária para resolver consultas *CPref-SQL*.

Este primeiro tipo de teste foi realizado sobre uma relação de filmes do *Internet Movie Database (IMDB)* contendo 248.722 tuplas. Como todas as consultas *CPref-SQL* retornaram alguma tupla ótima absoluta, os tempos de execução dos algoritmos **GetBest-E** e **GetBest-N** foram mesmos. No caso de consultas cujas tuplas ótimas absolutas não estão presentes na relação, o tempo gasto pelo algoritmo **GetBest-N** será proporcional ao número de tuplas contidas na relação, pois este algoritmo é a implementação do operador **Best-N** que retorna todas as tuplas da relação quando as tuplas ótimas absolutas não são encontradas.

O segundo tipo de teste foi realizado para avaliar o desempenho da subrotina **BuildBest** no que diz respeito a interpretação das consultas *CPref-SQL*, ou seja, a obtenção das condições desejadas para as tuplas ótimas absolutas a partir das contidas em uma consulta *CPref-SQL*.

Foram efetuados testes com consultas possuindo de dez a mil regras-pc. Em todas estas consultas as regras-pc foram arranjadas de forma que a regra-pc na posição i só será considerada depois que a regra-pc na posição $i+1$ for considerada, isto reflete o pior caso de interpretação das regras-pc pela subrotina **BuildBest**. Os resultados dos testes são exibidos na Figura 5.19.

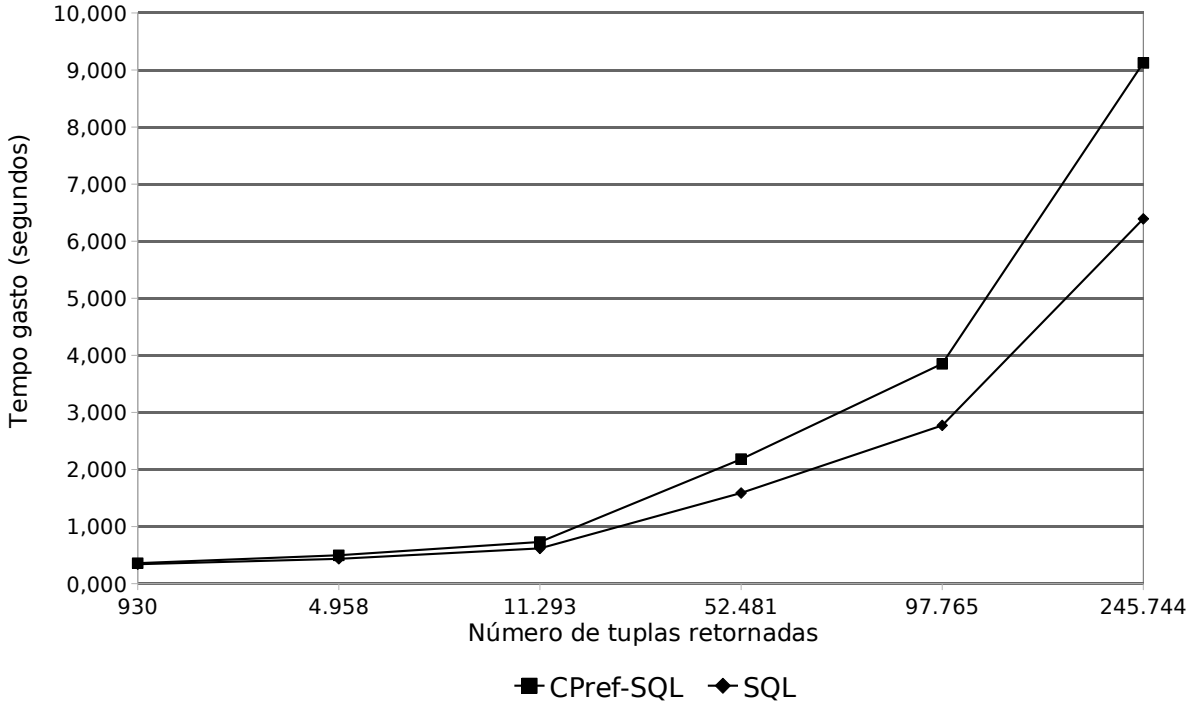


Figura 5.18: Teste para comparação de desempenho entre consultas SQL e *CPref-SQL*

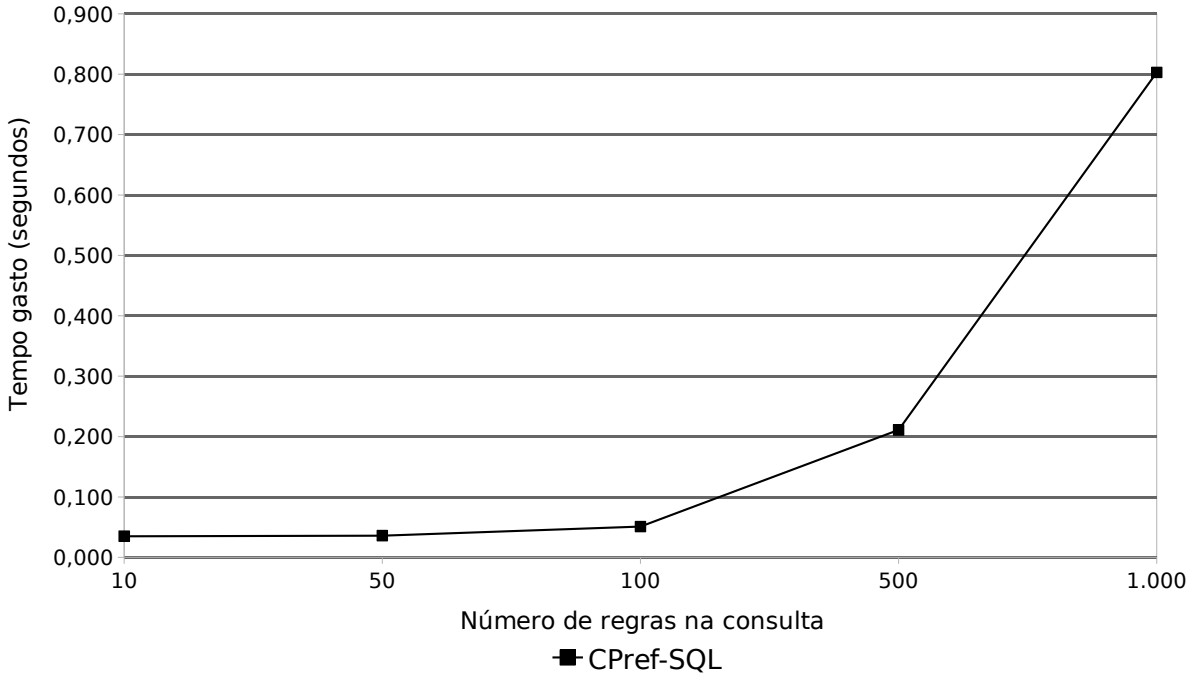


Figura 5.19: Teste de desempenho do algoritmo **BuildBest**

Como pode ser visto, o tempo de execução do algoritmo cresce em relação ao número de regras da consulta, mas o desempenho do algoritmo **BuildBest** é satisfatório pois raramente um usuário criará uma consulta contendo mais de mil regras-pc.

Todos os testes foram executados em um computador com processador Intel Core 2 Duo 4300 (1.8 GHz) com 2 Gigabytes de memória. Utilizamos o sistema operacional Linux (Distribuição Debian 4.0) e o SGBD PostgreSQL 8.1.11.

5.6 Considerações Finais

Neste capítulo apresentamos os algoritmos **GetBest-E**, **GetBest-N** e **GetBest-D** para resolver os operadores de seleção de tuplas ótimas **Best-E**, **Best-N** e **Best-D**, respectivamente. No caso do algoritmo **GetBest-D** foi utilizado o conceito de listas de escopos proposto no Capítulo 3.

Também neste capítulo descrevemos a implementação de um protótipo para um fragmento da linguagem *CPref-SQL*, através do qual realizamos alguns testes que constataram um desempenho satisfatório para os algoritmos implementados.

Considerações Finais da Parte I

Na primeira parte deste trabalho apresentamos diversos conceitos e propriedades relacionados à preferências sobre objetos (tuplas), abordamos os principais trabalhos sobre o tratamento de preferências e a incorporaração de preferências em linguagens de consultas para banco de dados.

Especificamos a linguagem *CPref-SQL* através de uma extensão da álgebra relacional convencional feita com o acréscimo dos operadores **Best-E**, **Best-N** e **Best-D**. No caso do operador **Best-D** utilizamos o conceito de conceito de listas de escopos para realizar o teste de dominância entre tuplas.

Para resolver os operadores **Best-E**, **Best-N** e **Best-D** foram propostos os algoritmos **GetBest-E**, **GetBest-N** e **GetBest-D**, respectivamente. Descrevemos como foi implementado um protótipo para um fragmento da linguagem *CPref-SQL* e através deste protótipo realizamos alguns testes onde constatamos um desempenho satisfatório para os algoritmos implementados.

Parte II

Preferências sobre Seqüências de Objetos

Capítulo 6

Fundamentos Teóricos e Trabalhos Correlatos

A primeira parte desta dissertação concentrou-se no tratamento de preferências condicionais sobre objetos simples (tuplas) através da linguagem de consulta *CPref-SQL*. Todavia há certos tipos de aplicações que necessitam de um tratamento de preferências sobre estruturas mais complexas como *conjuntos de objetos* ou *seqüências de objetos* [de Amo e Giacometti 2008]. De agora em diante iremos focar no tratamento de preferências sobre seqüências de objetos, também chamadas de *preferências temporais*, e na incorporaração deste tipo preferência no contexto de Banco de Dados.

As preferências temporais estão relacionadas principalmente com a disposição dos objetos de uma seqüência, ou seja, as preferências sobre os atributos de um objeto podem ser afetadas pela posição do objeto em relação aos demais objetos da seqüência. O Exemplo 6.1 exibe alguns tipos de preferências que envolvem esta noção temporal.

Exemplo 6.1. Neste exemplo vamos empregar algumas preferências temporais para exprimir características desejáveis em uma lista de músicas. Vamos considerar as seguintes preferências especificadas por um usuário:

1. Para músicas sertanejas uma duração longa é melhor do que uma duração breve, mas para músicas pop uma duração breve é melhor do que uma duração longa.
2. É melhor que a lista de música inicie com uma música sertaneja do que com uma música pop.
3. Se a música anterior foi pop é melhor que a próxima seja sertaneja, mas se a música anterior foi sertaneja é melhor que a próxima seja pop.

Inicialmente na Seção 6.1 discutiremos brevemente sobre duas abordagens no tratamento de preferências sobre conjuntos de objetos. Na Seção 6.2 é apresentado o formalismo lógico *TPref* para representação de preferências temporais na forma de preferências condicionais temporais. E a Seção 6.4 expõe nossas considerações finais do capítulo.

6.1 Preferências sobre Conjuntos de Objetos

Uma forma de trabalhar com preferências sobre *conjuntos* de objetos foi desenvolvida por [Brafman et al. 2006b], onde se relata como reduzir uma especificação de preferências sobre conjuntos de objetos para uma especificação de preferências sobre objetos simples. Esta especificação de preferências é feita tratando cada conjunto de objetos como um vetor, onde cada posição do vetor é uma propriedade desejada para o conjunto de objetos e assim cada vetor pode ser visto como um objeto simples. O Exemplo 6.2 mostra como é feita uma redução do problema de tratamento de preferências sobre conjuntos de objetos para o problema de tratamento de preferências sobre objetos simples.

Exemplo 6.2. Supondo os dois conjuntos de músicas exibidos na Figura 6.1(a) e na Figura 6.1(b). Vamos considerar que um usuário especifique as seguintes preferências sobre conjuntos:

1. Conjuntos com músicas movimentadas são preferidos a conjuntos com músicas tranqüilas;
2. Conjuntos de músicas com maior diversidade de gênero são melhores;
3. O ritmo das músicas é mais importante do que a diversidade de gênero;

Para o primeiro conjunto pode-se associar o vetor $(m, 2)$ e para o segundo conjunto pode-se associar o vetor $(t, 3)$. O primeiro elemento do vetor é o ritmo das músicas e o segundo elemento informa a diversidade de gênero do conjunto. O primeiro vetor é melhor do que o segundo considerando o ritmo e o segundo vetor é melhor do que o primeiro considerando a diversidade de gênero. Como o usuário especificou que o ritmo é mais importante do que a diversidade de gênero o primeiro vetor é o melhor e, conseqüentemente, o primeiro conjunto de músicas é o melhor.

movimentado	sertaneja
movimentado	pop
movimentado	sertaneja

(a) Conjunto 1

tranqüilo	sertaneja
tranqüilo	rock
tranqüilo	pop

(b) Conjunto 2

Figura 6.1: Conjuntos de músicas

Outro trabalho importante voltado para o tratamento de preferências sobre conjunto de objetos foi a linguagem *DD-PREF* proposta em [des Jardins e Wagstaff 2005]. Esta linguagem permite expressar preferências sobre a diversidade e a qualidade dos atributos dos objetos contidos em conjuntos. Por exemplo, comparando os conjuntos 1 e 2 (apresentados nas Figuras 6.1(a) e 6.1(b), respectivamente) no que diz respeito a diversidade de gênero o conjunto 2 é melhor do que o conjunto 1, mas considerando a qualidade de gênero voltada para sertanejo o conjunto 1 é melhor do que o conjunto 2.

Também nos trabalhos de [Brafman et al. 2006b] e [des Jardins e Wagstaff 2005] foram desenvolvidos algoritmos para otimização de subconjuntos de objetos, ou seja, estes algoritmos tentam obter o subconjunto ótimo atendendo certas preferências dentro de um conjunto maior de objetos.

6.2 Preferências sobre Seqüências de Objetos: O Formalismo Lógico TPref

O estudo de preferências sobre seqüências de objetos é bastante útil quando precisamos considerar a disposição de objetos dentro de uma lista. Vamos considerar novamente o Exemplo 6.1 e as listas de músicas exibidas nas Figuras 6.2(a) e 6.2(b). Intuitivamente podemos ver que a segunda lista de músicas é preferida à primeira de acordo com as preferências do usuário.

Posição	Duração	Gênero
1	breve	sertanejo
2	longa	rock
3	longa	sertaneja

(a) Lista de músicas 1

Posição	Duração	Gênero
1	longa	sertanejo
2	breve	pop
3	longa	sertanejo

(b) Listas de músicas 2

Figura 6.2: Listas de músicas

Como a representação e raciocínio com preferências sobre seqüências de objetos apresenta propriedades bem específicas, precisamos de um formalismo adequado para este tipo de tarefa. O formalismo que adotamos neste trabalho é o formalismo *TPref* proposto por [de Amo e Giacometti 2007]. O formalismo lógico *TPref* teve como base o trabalho de [Wilson 2004b] e foi estendido para suportar preferências condicionais temporais. Este formalismo apresenta importantes propriedades para o desenvolvimento de algoritmos capazes de solucionar diversos problemas ligados a preferências condicionais temporais.

Inicialmente na Subseção 6.2.1 são apresentadas as condições temporais que fazem parte das regras de preferências do formalismo lógico *TPref*. A seguir, a Subseção 6.2.2 define uma teoria de preferência condicional temporal que é utilizada para representar um conjunto de preferências temporais. Posteriormente, a Subseção 6.2.3 exibe como esta teoria impõe uma relação de ordem sobre um conjunto de seqüências e a Subseção 6.2.4 apresenta as condições necessárias para que uma teoria de preferência condicional temporal seja consistente.

6.2.1 Condições Temporais

No tratamento de preferências condicionais temporais, assim como no tratamento de preferências condicionais, é considerado um conjunto de atributos $A = \{X_1, ..., X_n\}$, onde $\mathbf{Dom}(X_i)$ denota o domínio de X_i, ou seja, os possíveis valores que podem ser assumidos por X_i. O conjunto $\mathbf{Dom}(X_1) \times ... \times \mathbf{Dom}(X_n)$ é denotado por $\mathcal{O}$ e representa o conjunto de objetos formados por todas as combinações de valores possíveis para os atributos presentes em A. Se $o = (x_1, ..., x_n)$ é um objeto então $o[X_i]$ denota o valor do atributo X_i no objeto o.

O formalismo lógico *TPref* faz uso de um conjunto de regras para a representação de preferências condicionais temporais, cada uma destas regras possui uma *condição temporal* que, em determinadas posições de uma seqüência, torna a regra válida ou não.

As condições temporais são expressas por meio de uma adaptação da Lógica Temporal Proposicional (LTP) introduzida em [Prior 1967], visto que intuitivamente cada posição de uma seqüência pode ser percebida como um estado no tempo. Na LTP as fórmulas básicas são variáveis proposicionais $P_1, ..., P_n$. No formalismo lógico TPref as fórmulas básicas ou proposições têm a forma $A = a_i$, onde A é um atributo e a_i é um possível valor para A. Como as fórmulas básicas assumem um formato particular no formalismo lógico *TPref*, a lógica usada mantém as propriedades da LTP, mas recebe o nome de Lógica Temporal Simples (LTS) apenas por uma questão de diferenciação. Sendo assim uma condição temporal é uma fórmula LTS dada pela Definição 6.1.

Definição 6.1. (Fórmula LTS [de Amo e Giacometti 2007]) Uma *fórmula LTS* é definida como se segue:

1. **true** e **false** são fórmulas LTS;
2. Se P é uma proposição, então P é uma fórmula LTS;
3. Se P_1 e P_2 são fórmulas LTS, então $P_1 \wedge P_2$, $P_1 \vee P_2$ e $\neg P_1$ são fórmulas LTS.
4. Se P_1 e P_2 são fórmulas LTS, então P_1 **Until** P_2 e P_1 **Since** P_2 são fórmulas LTS.

As condições temporais serão *satisfeitas* levando em consideração uma seqüência de objetos dada pela Definição 6.2.

Definição 6.2. (Seqüência de objetos [de Amo e Giacometti 2007]) Uma *seqüência de objetos* é uma estrutura que consiste de um conjunto de objetos $\{o_1, ..., o_k\}$ com uma ordenação temporal $o_1 < ... < o_k$ que significa que o_i precede o_{i+1}.

Uma seqüência de objetos s pode ser denotada simplesmente por $s = \langle o_1, ..., o_k \rangle$ e $|s| = k$ denota a dimensão temporal (tamanho) de s, ou seja, o número de objetos que s possui. Denotamos por $\mathbf{Seq}(\mathcal{O})$ o conjunto de todas as possíveis seqüências constituídas

pelos objetos pertencentes a $\mathcal{O}$. Já a notação $\mathbf{Seq}_n(\mathcal{O})$ representa o conjunto de seqüências com dimensão temporal n em $\mathbf{Seq}(\mathcal{O})$.

Uma fórmula $F \in LTS$ é satisfeita por uma seqüência de objetos $s = \langle o_1, ..., o_k \rangle$ em um estado $i \in \{1, ..., k\}$, denotado por $(s, i) \models F$ quando:

1. $(s, i) \models (A = a)$ se e somente se $o_i[A] = a$;
2. $(s, i) \models F \wedge G$ se e somente se $(s, i) \models F$ e $(s, i) \models G$;
3. $(s, i) \models F \vee G$ se e somente se $(s, i) \models F$ ou $(s, i) \models G$;
4. $(s, i) \models \neg F$ se e somente se $(s, i) \not\models F$;
5. $(s, i) \models F$ **Until** G se e somente se existe j tal que $i < j \leq |s|$ e $(s, j) \models G$ e para todo k tal que $i \leq k < j$ tem-se $(s, k) \models F$;
6. $(s, i) \models F$ **Since** G se e somente se existe j tal que $1 \leq j < i$ e $(s, j) \models G$ e para todo k tal que $j < k \leq i$ tem-se $(s, k) \models F$;

Uma seqüência s satisfaz uma fórmula LTS F, denotado por $s \models F$, se $(s, i) \models F$, onde $i = |s|$. A fórmula **true** é satisfeita por qualquer seqüência e a fórmula **false** não é satisfeita por nenhuma seqüência. Duas fórmulas F e G são *equivalentes* (*last-equivalentes* ou *l-equivalentes*) se e somente se para toda seqüência s, $(s \models F \iff s \models G)$. Duas fórmulas F e G são *globalmente equivalentes* se e somente se para qualquer seqüência s, $((s, i) \models F \iff (s, i) \models G)$ para todo $i \in \{1, ..., |s|\}$.

A partir das fórmulas LTS definidas anteriormente, são definidas as seguintes fórmulas derivadas:

- **Prev** F equivale a **false Since** F e referencia a posição anterior, ou seja, **Prev** F é satisfeita no estado i quando F for satisfeita na posição imediatamente anterior $(i - 1)$;
- **Next** F equivale a **false Until** F e referencia a próxima posição, ou seja, **Next** F é satisfeita no estado i quando F for satisfeita na posição imediatamente posterior $(i + 1)$;
- **First** equivale a $\neg$ **Prev true** e referencia a primeira posição, ou seja, **First** é satisfeita no estado i somente quando a posição i for a primeira posição $(i = 1)$;
- **Last** equivale a $\neg$ **Next true** e referencia a última posição, ou seja, **Last** é satisfeita no estado i somente quando a posição i for a última posição;
- $\blacklozenge F$ equivale a **true Since** F e referencia alguma posição anterior, ou seja, $\blacklozenge F$ é satisfeita no estado i quando F for satisfeita em alguma posição anterior a i;
- $\lozenge F$ equivale a **true Until** F e referencia alguma posição posterior, ou seja, $\lozenge F$ é satisfeita no estado i quando F for satisfeita em alguma posição posterior a i;

- $\blacksquare F$ equivale a $\neg\blacklozenge\neg F$ e referencia todas as posições anteriores, ou seja, $\blacksquare F$ é satisfeita no estado i quando F for satisfeita em todas as posições anteriores a i;
- $\Box F$ equivale a $\neg\Diamond\neg F$ e referencia todas as posições posteriores, ou seja, $\Box F$ é satisfeita no estado i quando F for satisfeita em todas as posições posteriores a i.

Uma propriedade muito importante verificada em fórmulas *LTP*, e conseqüentemente em fórmulas *LTS*, é a propriedade de *separação*, a qual garante que toda fórmula *LTP* é globalmente equivalente a uma combinação booleana de *fórmulas puras de presente, de passado e de futuro* cujos conceitos são dados na Definição 6.3. O Exemplo 6.3 mostra algumas fórmulas puras.

Definição 6.3. (Fórmulas puras de presente, fórmulas puras de passado e fórmulas puras de futuro [de Amo e Giacometti 2007]) Uma *fórmula pura de presente* é uma fórmula definida como se segue:

1. Uma proposição $X = a$ é uma fórmula pura de presente;
2. Uma combinação booleana de fórmulas puras de presente é uma fórmula pura de presente.

Uma *fórmula pura de passado* é uma fórmula definida como se segue:

1. Se F e G são fórmulas puras de presente então F **Since** G é uma fórmula pura de passado;
2. Uma combinação booleana de fórmulas puras de passado é uma fórmula pura de passado.

Uma *fórmula pura de futuro* é uma fórmula definida como se segue:

1. Se F e G são fórmulas pura de presente então F **Until** G é uma fórmula pura de futuro;
2. Uma combinação booleana de fórmulas puras de futuro é uma fórmula pura de futuro.

Exemplo 6.3. Como exemplo de fórmula puras de passado podemos considerar (**Prev** F)$\wedge$ (**First**) $\wedge$ ($\blacklozenge G$) $\wedge$ ($\blacksquare H$), uma vez que as fórmulas **Prev** F, **First**, $\blacklozenge G$ e $\blacksquare H$ são todas fórmulas derivadas de **Since**. Como exemplo de fórmula puras de futuro podemos considerar (**Next** F) $\wedge$ (**Last**) $\wedge$ ($\Diamond G$) $\wedge$ ($\Box H$), uma vez que as fórmulas **Next** F, **Last**, $\Diamond G$ e $\Box H$ são todas fórmulas derivadas de **Until**.

A propriedade de separação é validada pelo Teorema 6.1 que é demonstrado detalhadamente no trabalho de [Gabbay 1987]. Entretanto, devemos lembrar que esta propriedade não é trivial, existindo problemas em aberto com respeito a complexidade da separação de fórmulas LTP [Hodkinson e Reynolds 2005].

Teorema 6.1. (Teorema da Separação [Gabbay 1987]) *Seja F uma fórmula LTS, então F é globalmente equivalente a uma fórmula separada.*

Uma fórmula F está *separada* se F está no formato $F_1 \vee ... \vee F_n$, onde F_i possui o formato $F_i^0 \wedge F_i^- \wedge F_i^+$, sendo que F_i^0 é uma fórmula pura de presente, F_i^- é uma fórmula pura de passado e F_i^+ é uma fórmula pura de futuro.

Vamos considerar, por exemplo, a fórmula $(\mathbf{Prev}\ F) \wedge \mathbf{Last} \wedge (A = a) \vee (\Diamond G) \wedge \mathbf{First} \wedge (B = b)$. Esta fórmula apresenta dois componentes separados $(\mathbf{Prev}\ F) \wedge \mathbf{Last} \wedge (A = a)$ e $(\Diamond G) \wedge \mathbf{First} \wedge (B = b)$. No primeiro componente temos a fórmula pura de presente $(A = a)$, a fórmula pura de passado $(\mathbf{Prev}\ F)$ e a fórmula pura de futuro **Last**. No segundo componente temos a fórmula pura de presente $(B = b)$, a fórmula pura de passado **First** e a fórmula pura de futuro $(\Diamond G)$.

6.2.2 Teoria de Preferência Condicional Temporal

A especificação de preferências temporais é realizada através de uma *teoria de preferência condicional temporal* que, por sua vez, é composta por um conjunto de *regras de preferência condicional temporal*. Os conceitos de regra de preferência condicional temporal e de teoria de preferência condicional temporal são dados pelas Definições 6.4 e 6.5, respectivamente.

Definição 6.4. (Regra de preferência condicional temporal (regra-pct) [de Amo e Giacometti 2007]) Seja A um conjunto de atributos. Uma *regra de preferência condicional temporal*, ou simplesmente *regra-pct*, é uma expressão na forma $\varphi : F \rightarrow (X = x) > (X = x')$, onde $X \in A$, $x, x' \in \mathbf{Dom}(X)$ e F é uma fórmula *LTS* separada. Uma regra-pct *simples* é uma regra-pct onde a condição temporal contém um único componente da disjunção, assim uma regra-pct é equivalente a um conjunto de regras-pct simples.

Como exemplo de regra-pct podemos considerar $\varphi : (\mathbf{Prev}(C = c)) \wedge (A = a) \vee \mathbf{First} \wedge (B = b) \rightarrow (D = d) > (D = d')$. Esta regra-pct não é simples, mas equivale ao seguinte conjunto de regras-pct simples:

- $\varphi_1 : (\mathbf{Prev}(C = c)) \wedge (A = a) \rightarrow (D = d) > (D = d')$;
- $\varphi_2 : \mathbf{First} \wedge (B = b) \rightarrow (D = d) > (D = d')$.

Os elementos que aparecem em uma regra-pct φ podem ser referenciados através de certas notações. A notação $F_\varphi^- \wedge F_\varphi^0 \wedge F_\varphi^+$ indica as fórmulas que aparecem na condição temporal, $(X_\varphi = x_\varphi) > (X_\varphi = x'_\varphi)$ denota que o valor x é preferido ao valor x' para o atributo X na regra φ e $\mathbf{Attr}(F)$ denota o conjunto de atributos que aparecem na condição temporal F. Por exemplo, na regra-pct $\varphi_1 : (\mathbf{Prev}(C = c)) \wedge (A = a) \rightarrow (D = d) > (D = d')$, temos:

- $F_{\varphi_1}^- : \mathbf{Prev}(C = c)$, $F_{\varphi_1}^0 : (A = a)$;

- $\mathbf{Attr}(F)_{\varphi_1} = \{A, C\}$, $\mathbf{Attr}(F^-_{\varphi_1}) = \{C\}$, $\mathbf{Attr}(F^0_{\varphi_1}) = \{A\}$;
- $(D_{\varphi_1} = d_{\varphi_1}) > (D_{\varphi_1} = d'_{\varphi_1})$, a regra-pct φ_1 indica que o valor d é preferido ao valor d' para o atributo D.

Definição 6.5. (Teoria de preferência condicional temporal [de Amo e Giacometti 2007]) Uma *teoria de preferência condicional temporal*, ou simplesmente *teoria-pct*, é um conjunto finito de regras-pct simples $F \rightarrow (X = x) > (X = x')$, onde $X \notin \mathbf{Attr}(F^0)$.

O Exemplo 6.4 mostra uma representação de preferências condicionais temporais através de uma teoria-pct.

Exemplo 6.4. As preferências temporais especificadas no Exemplo 6.1 podem ser representadas pela teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$, onde:

- $\varphi_1 : (\mathit{Gênero} = \text{`sertanejo'}) \rightarrow (\mathit{Duração} = \text{`longa'}) > (\mathit{Duração} = \text{`breve'})$;
- $\varphi_2 : (\mathit{Gênero} = \text{`pop'}) \rightarrow (\mathit{Duração} = \text{`breve'}) > (\mathit{Duração} = \text{`longa'})$;
- $\varphi_3 : \mathbf{First} \rightarrow (\mathit{Gênero} = \text{`sertanejo'}) > (\mathit{Gênero} = \text{`pop'})$;
- $\varphi_4 : \mathbf{Prev}(\mathit{Gênero}=\text{`pop'}) \rightarrow (\mathit{Gênero} = \text{`sertanejo'}) > (\mathit{Gênero} = \text{`pop'})$;
- $\varphi_5 : \mathbf{Prev}(\mathit{Gênero} = \text{`sertanejo'}) \rightarrow (\mathit{Gênero} = \text{`pop'}) > (\mathit{Gênero} = \text{`sertanejo'})$.

6.2.3 Ordem Induzida por uma Teoria-pct

Uma teoria-pct Φ induz implicitamente uma ordem parcial estrita sobre as seqüências de $\mathbf{Seq}(\mathcal{O})$, permitindo assim que tais seqüências possam ser comparadas através de Φ. Cada regra-pct φ induz uma relação de ordem R_φ sobre $\mathbf{Seq}(\mathcal{O})$. Esta relação possibilita comparar duas seqüências que diferem em apenas uma posição e em apenas um atributo, conforme é mostrado na Definição 6.6.

Definição 6.6. (Comparação de seqüências de mesmo tamanho que diferem em uma única posição [de Amo e Giacometti 2007]) Dadas duas seqüências $s = \langle o_1, ..., o_n \rangle$ e $s' = \langle o'_1, ..., o'_n \rangle$ e uma regra-pct $\varphi : F \rightarrow (X = x) > (X = x')$ então $sR_\varphi s'$ se e somente se existe $j \in \{1, ..., n\}$ tal que:

1. $o_j \neq o'_j$ e $o_i = o'_i$ para todo $i \in \{1, ..., n\} \setminus \{j\}$;
2. $(s, j) \models F_\varphi$ e $(s', j) \models F_\varphi$;
3. $o_j[X_\varphi] = x_\varphi$ e $o'_j[X_\varphi] = x'_\varphi$;
4. Para todo $Y \in A \setminus \{X_\varphi\}$, $o_j[Y] = o'_j[Y]$.

Como exemplo vamos considerar as seqüências $s_1 = \langle (a_1, b_1), (a_2, b_2), (a_1, b_3) \rangle$, $s_2 = \langle (a_1, b_1), (a_2, b_1), (a_1, b_3) \rangle$ e a regra-pct $\varphi_1 : (\mathbf{Prev}(A = a_1)) \rightarrow (B = b_2) > (B = b_1)$. A única posição diferente nas duas seqüências é a segunda posição. A condição da regra-pct φ_1 é satisfeitas por ambas seqüências nesta posição. O único atributo com valores

diferentes na segunda posição para ambas seqüências é o atributo B. A regra-pct φ_1 indica que o valor b_2 é preferido ao valor b_1 para o atributo B, portanto $s_1 R_{\varphi_1} s_2$.

A comparação de seqüências que são diferentes em mais de uma posição é feita considerando o fecho transitivo da união das relações de ordem R_φ para todas as regras-pct φ de Φ. Isto é, uma seqüência s é preferida a uma seqüência s' de acordo com Φ se $s >_\Phi s'$, onde $>_\Phi$ é o fecho transitivo de R_Φ, sendo que R_Φ denota o conjunto $\bigcup_{\varphi \in \Phi} R_\varphi$. Utilizando o conceito de *Cadeia de melhoramento de seqüências* da Definição 6.7, o Lema 6.1 apresenta a condição necessária e suficiente para que duas seqüências de mesmo tamanho sejam comparadas de acordo com Φ e o Exemplo 6.5 demonstra como é feita uma comparação entre seqüências utilizando este conceito.

Definição 6.7. (Cadeia de melhoramento de seqüências [de Amo e Giacometti 2007]) Sejam s e s' duas seqüências de tamanho n. Existe uma *cadeia de melhoramento de seqüências* de s' para s de acordo com Φ se existe um conjunto de seqüências $\{s_1, ..., s_{p+1}\}$ e um conjunto de regras-pct $\{\varphi_1, ..., \varphi_p\}$ em Φ tal que $s_1 = s', s_{p+1} = s$ e $s_k R_{\varphi_k} s_{k+1}$ para todo $k \in \{1, ..., p\}$.

Lema 6.1. ([de Amo e Giacometti 2007]) *Seja Φ uma teoria-pct. Sejam s e s' duas seqüências de tamanho n. Então $s >_\Phi s'$ se e somente se existe uma cadeia de melhoramento de seqüências de s' para s.*

Exemplo 6.5. Neste exemplo vamos considerar as seqüências $s = \langle (a_1, b_1), (a_2, b_2), (a_1, b_3) \rangle$, $s' = \langle (a_1, b_1), (a_2, b_1), (a_1, b_1) \rangle$ e a teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$, onde:

- $\varphi_1 : (\mathbf{Prev}(A = a_1)) \rightarrow (B = b_2) > (B = b_1)$;
- $\varphi_2 : (\mathbf{Prev}(A = a_2)) \rightarrow (B = b_2) > (B = b_1)$;
- $\varphi_3 : (\mathbf{Prev}(A = a_2)) \rightarrow (B = b_3) > (B = b_2)$.

Podemos concluir que $s >_\Phi s'$ pela existência da cadeia de melhoramento de seqüências $s_1 R_{\varphi_1} s_2 R_{\varphi_2} s_3 R_{\varphi_3} s_4$, onde:

- $s_1 = s'$;
- $s_2 = \langle (a_1, b_1), (a_2, b_2), (a_1, b_1) \rangle$;
- $s_3 = \langle (a_1, b_1), (a_2, b_2), (a_1, b_2) \rangle$;
- $s_4 = s$;

Até agora, foram apresentados diversos conceitos e propriedades importantes principalmente quanto à representação e comparação de seqüências. Contudo, as propriedades apresentadas são válidas apenas para teorias-pct *consistentes* [de Amo e Giacometti 2008]. O conceito de consistência é introduzido na Definição 6.8.

Definição 6.8. (Teoria-pct consistente [de Amo e Giacometti 2007]) Seja Φ uma teoria-pct, Φ é *consistente* se e somente se $>_\Phi$ é irreflexiva, isto é, $>_\Phi$ é uma ordem parcial estrita sobre $\mathbf{Seq}(\mathcal{O})$(lembrando que, por definição $>_\Phi$ é transitiva e que transitividade e irreflexividade implica em anti-simetria).

6.2.4 Teste de Consistência

No trabalho de [de Amo e Giacometti 2007] a consistência de uma teoria-pct Φ é verificada quando Φ possui apenas regras-pct cujas condições são conjunções de fórmulas *LTS* puras de presente e de passado, ou seja, para toda regra-pct $\varphi \in \Phi$, $F_\varphi = F^-_\varphi \wedge F^o_\varphi$. O conjunto de todas as teorias-pct com regras-pct neste formato é denotado por **TPref***. O teste de consistência de uma teoria-pct **TPref*** se reduz a um certo número de testes de consistência de teorias-pc (não temporais).

As teorias-pc necessárias para o teste de consistência são obtidas com base nas regras-pct de Φ que são válidas em cada posição de uma seqüência $s \in \mathbf{Seq}(\mathcal{O})$. Mais precisamente, dada uma teoria-pct Φ e uma seqüência $s \in \mathbf{Seq}(\mathcal{O})$, é definida uma teoria-pc $\Gamma_j(\Phi, s) = \{\varphi^0 \mid \varphi \in \Phi \wedge (s, j) \models F^-_\varphi \wedge F^+_\varphi\}$ para todo inteiro $j \in \{1, ..., |s|\}$, onde $\varphi^0 : F^0_\varphi \rightarrow (X_\varphi = x_\varphi) > (X_\varphi = x'_\varphi)$ é uma regra-pc obtida a partir da regra-pct $\varphi : F^-_\varphi \wedge F^0_\varphi \wedge F^+_\varphi \rightarrow (X_\varphi = x_\varphi) > (X_\varphi = x'_\varphi)$. Intuitivamente, $\Gamma_j(\Phi, s)$ é o conjunto de componentes de presente das condições das regras-pct cujos componentes de passado e futuro são satisfeitos pela seqüência s na posição j.

A garantia de que uma teoria-pct Φ é consistente é fornecida pelo Teorema 6.2, lembrando que o teorema é válido apenas para teorias-pct em **TPref*** e portanto, no decorrer deste trabalho, é usado o termo *teoria-pct consistente* para referenciar este tipo de teoria-pct. Apesar de considerarmos apenas este tipo de teoria-pct, é possível tratar teorias-pct com regras cujas condições possuem componentes de presente e de futuro simultaneamente (mas não passado e futuro simultaneamente. O Exemplo 6.6 mostra a utilização do Teorema 6.2 para verificar a consistência de uma teoria-pct $\Phi \in$ **TPref***.

Teorema 6.2. ([de Amo e Giacometti 2007]) *Seja Φ uma teoria-pct tal que para toda regra-pct $\varphi \in \Phi, \varphi \in$ **TPref***. A teoria-pct Φ é consistente se e somente se para toda seqüência s de tamanho $k > 0$, $\Gamma_k(\Phi, s)$ é consistente.*

Exemplo 6.6. Seja uma teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$, onde:

- φ_1 : **First** $\rightarrow$ (*Gênero = 'sertanejo'*) > (*Gênero='pop'*);
- φ_2 : **Prev** (*Gênero = 'sertanejo'*) $\rightarrow$ (*Gênero = 'pop'*) > (*Gênero='sertanejo'*);
- φ_3 : **Prev** (*Gênero='pop'*) $\rightarrow$ (*Gênero = 'sertanejo'*) > (*Gênero='pop'*).

Considerando **Dom**(Gênero) = {'sertanejo', 'pop'}, podemos verificar o seguinte:

- φ_1 sempre é válida para a primeira posição de qualquer seqüência $s \in \mathbf{Seq}(\mathcal{O})$. Desta maneira, temos que $\Gamma_1 = \{\varphi'_1 : (Gênero = 'sertanejo') > (Gênero='pop')\}$ é consistente (como vimos no Capítulo 2);
- φ_2 sempre é válida para a posição $i > 1$ de qualquer seqüência $s \in \mathbf{Seq}(\mathcal{O})$, onde $|s| > 1$ e o_{i-1}[Gênero] = 'sertanejo'. Desta maneira, temos que $\Gamma_i = \{\varphi'_2 : (Gênero = 'pop') > (Gênero='sertanejo')\}$ é consistente;

- φ_3 sempre é válida para a posição $j > 1$ de qualquer seqüência $s \in \mathbf{Seq}(\mathcal{O})$, onde $|s| > 1$ e o_{j-1}[Gênero] = 'pop'. Desta maneira, temos que $\Gamma_j = \{\varphi'_3 : (Gênero = \text{'}sertanejo\text{'}) > (Gênero = \text{'}pop\text{'})\}$ é consistente;

Como φ_2 é válida somente quando φ_1 e φ_3 não são válidas podemos concluir que a teoria-pct Φ é consistente, pois $\Gamma_k(\Phi, s)$ é consistente para toda seqüência s de tamanho $k > 0$.

6.2.5 Comparação de Seqüências com Tamanhos Distintos

Em inúmeras situações é necessário realizar comparações entre seqüências que possuem tamanhos diferentes, principalmente no contexto de Banco de Dados (onde podemos ter seqüências de diversos tamanhos armazenadas em uma relação), como veremos nos capítulos posteriores.

Antes de definirmos como comparar duas seqüências com tamanhos distintos de acordo com uma teoria-pct, precisamos entender o Lema 6.2. Este lema utiliza o conceito de *remover o último objeto* dado pela Definição 6.9 e define a condição necessária para que duas seqüências s e s' satisfaçam $s >_\Phi s'$, onde Φ é uma teoria-pct em **TPref***.

Definição 6.9. (Remover o último objeto [de Amo e Giacometti 2007]) Seja $s = \langle o_1, ..., o_n \rangle$ uma seqüência em $\mathbf{Seq}_n(\mathcal{O})$. A operação *remover o último objeto* é definida por: $ruo(s) = \langle o_1, ..., o_{n-1} \rangle$.

Lema 6.2. ([de Amo e Giacometti 2007]) *Seja Φ uma teoria-pct tal que para toda $\varphi \in \Phi, \varphi \in$ **TPref***. Para todo par de seqüências $s = \langle o_1, ..., o_{n+1} \rangle$ e $s' = \langle o'_1, ..., o'_{n+1} \rangle$ em $\boldsymbol{Seq}_{n+1}(\mathcal{O})$ com $n > 0$, se $s >_\Phi s'$, então $ruo(s) >_\Phi ruo(s')$, ou $ruo(s) = ruo(s')$ e $o_{n+1} >_\Gamma o'_{n+1}$ onde $\Gamma = \Gamma_{n+1}(\Phi, s) = \Gamma_{n+1}(\Phi, s')$.*

O Lema 6.2 reduz o problema de comparar duas seqüências para o problema de comparar dois objetos. Além disto, podemos enxergar a comparação entre duas seqüências de maneira análoga a comparação lexicográfica entre duas palavras. Desta maneira, podemos utilizar esta idéia para comparar duas seqüências de tamanhos distintos, veja a Definição 6.10.

Definição 6.10. (Comparação de seqüências com tamanhos distintos) Seja Φ uma teoria-pct. Sejam s e s' duas seqüências tais que $|s| = n$, $|s'| = m$ e $n > m$. Dizemos que $s >_\Phi s'$, se $prefixo(s, m) >_\Phi s'$, onde $prefixo(s, m)$ é uma seqüência com as m primeiras posições de s.

O Exemplo 6.7 demonstra algumas comparações de seqüências utilizando os conceitos definidos até o momento.

Exemplo 6.7. Considerando a teoria-pct do Exemplo 6.4 e as seqüências:
$s_1 = \langle$(sertanejo, breve), (pop, longa), (sertanejo, longa)$\rangle$,
$s_2 = \langle$(sertanejo, breve), (pop, breve)$\rangle$ e

$s_3 = \langle$(sertanejo, longa), (pop, breve)$\rangle$.
Podemos constatar que $s_3 >_\Phi s_2$ pois $s_3 R_{\varphi_1} s_2$ e também que $s_2 >_\Phi s_1$ pois $s_2 R_{\varphi_1} prefixo(s_1, 2)$, onde $prefixo(s_1, 2) = \langle$(sertanejo, breve), (pop, longa)$\rangle$. Além disto, $s_3 >_\Phi s_1$ pois há uma *cadeia de melhoramento de seqüências* de s_1 para s_3.

6.3 Comparação de Seqüências Utilizando o Conceito de Listas de Escopos

Através do Lema 6.2 percebemos que o problema da comparação de seqüências segundo uma teoria-pct consistente se reduz ao problema da comparação de objetos (tuplas) segundo uma teoria-pc consistente. Desta maneira, a comparação de seqüências segundo uma teoria-pct consistente pode fazer uso do conceito de listas de escopos apresentado no Capítulo 3. A comparação de duas seqüências s e s' de acordo com uma teoria-pct Φ consistente utilizando listas de escopos é feita da seguinte maneira:

- Procuramos a primeira posição em que s é diferente de s';
- Seja k esta posição, obtemos a teoria-pc $\Gamma_k = \Gamma_k(\Phi, s) = \Gamma_k(\Phi, s')$;
- Sejam t_k o objeto na posição k de s e t'_k o objeto na posição k de s', comparamos t_k e t'_k através de suas listas de escopos obtidas a partir de Γ_k
- Quando não existe uma posição em que s é diferente de s' ou quando t_k é incomparável a t'_k, então as seqüências s e s' são incomparáveis.

O Exemplo 6.8 demonstra como comparar duas seqüências utilizando o conceito de listas de escopos.

Exemplo 6.8. Neste exemplo vamos considerar os atributos A e B com seus respectivos domínios $\mathbf{Dom}(A) = \{a_1, a_2\}$ e $\mathbf{Dom}(B) = \{b_1, b_2\}$. Vamos considerar também a teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$, onde:

- $\varphi_1 : \mathbf{First} \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_2 : \mathbf{Prev}(A = a_1) \rightarrow (A = a_2) > (A = a_1)$;
- $\varphi_3 : \mathbf{Prev}(A = a_2) \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_4 : (A = a_1) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

A comparação entre as seqüências $s = \langle(a_1, b_2), (a_2, b_2)\rangle$ e $s' = \langle(a_1, b_2), (a_1, b_1), (a_3, b_1)\rangle$ é realizada da seguinte maneira:

- A primeira posição em que s difere de s' é a posição 2;
- Obtemos a teoria-pc $\Gamma_2 = \{\varphi'_2, \varphi'_4, \varphi'_5\}$, onde:

- $\varphi'_2 : (A = a_2) > (A = a_1)$;
- $\varphi'_4 : (A = a_1) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi'_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$;

- Obtemos a lista de escopos $[1, 4]$ para (a_2, b_2) (segunda posição de s) e a lista de escopos $[1, 1]$ para (a_1, b_1) (segunda posição de s') através do grafo BTG_{Γ_2} da Figura 6.3;
- Como $[1, 4]$ abrange $[1, 1]$, temos que $(a_2, b_2) >_{\Gamma_2} (a_1, b_1)$, conseqüentemente $s >_\Phi s'$.

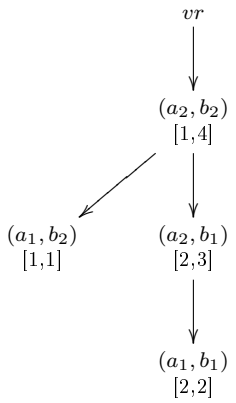


Figura 6.3: Grafo BTG_{Γ_2} para a posição 2

6.4 Considerações Finais

Este capítulo abordou o tratamento de preferências sobre estruturas mais complexas do que objetos simples. Foram discutidos brevemente trabalhos sobre o tratamento de preferências sobre conjunto de objetos. E principalmente, foi apresentado com maiores detalhes o formalismo lógico *TPref* para a representação de preferências condicionais temporais, bem como propriedades importantes que possibilitam a comparação de seqüências e a verificação da consistência de um conjunto de regras de preferências temporais.

Além disto, foi demonstrado como utilizar o conceito de listas de escopos em conjunto com o formalismo lógico *TPref* para a comparação de seqüências de acordo com uma teoria-pct.

Capítulo 7

Álgebra TPrefAR

No próximo capítulo iremos especificar a linguagem de consulta para banco de dados *TPref-SQL* para que consultas contendo preferências condicionais temporais possam ser resolvidas. A linguagem *TPref-SQL* será fundamentada em uma nova álgebra relacional com operadores específicos para tratar operações sobre seqüências de tuplas denominada *álgebra TPrefAR* e em um modelo relacional com relações próprias para armazenar seqüências de tuplas.

No modelo relacional convencional um esquema relacional $R(A_1, ..., A_n)$ é definido para que tuplas sobre os atributos $A_1, ..., A_n$ possam ser armazenadas. Entretanto, para armazenarmos seqüências precisamos identificar cada seqüência individualmente e identificar cada posição de uma seqüência.

No modelo relacional convencional podemos utilizar relações sobre um esquema relacional $R(\mathcal{ID}, \mathcal{P}, A_1, ..., A_k)$, onde $\mathcal{ID}$ é o *atributo identificador de seqüência* e $\mathcal{P}$ é o *atributo identificador de posição*. O atributo $\mathcal{ID}$ possui um valor único para cada seqüência e o atributo $\mathcal{P}$ representa a posição de cada tupla na seqüência. Os atributos $A_1, ..., A_k$ são os atributos que efetivamente representam os dados das tuplas das seqüências, por esta razão são chamados de *atributos de dados*.

O conjunto finito de esquemas relacionais $\mathbb{R} = \{R_1, ..., R_n\}$ recebe o nome de *esquema de banco de dados*. Uma *instância de banco de dados* sobre um esquema $\mathbb{R}$ é uma função que associa a cada esquema relacional R_i uma relação r_i sobre este esquema relacional.

O modelo relacional de seqüências que vamos considerar utiliza relações de seqüências sobre *esquemas relacionais de seqüências* para a representação de seqüências. Em um *esquema relacional de seqüências* não mencionamos os atributos $\mathcal{ID}$ e $\mathcal{P}$, pois as informações sobre estes atributos estão explícitas nas seqüências. Uma *instância relacional de seqüências* sobre um esquema relacional de seqüências $R(A_1, ..., A_k)$ é um conjunto de seqüências, onde cada elemento da seqüência é uma tupla sobre o esquema $R(A_1, ..., A_k)$.

O Exemplo 7.1 mostra como as seqüências são instanciadas no modelo relacional de seqüências e no modelo relacional convencional.

Exemplo 7.1. Neste exemplo vamos mostrar como as seqüências $s_1 =< (a_1, b_1), (a_2, b_2), (a_3, b_3) >$ e $s_2 =< (a_1, b_1), (a_4, b_4) >$ são armazenadas nos dois modelos relacionais discutidos. A Figura 7.1(a) apresenta a instância r sobre o esquema relacional convencional $R(\mathcal{ID}, \mathcal{P}, A, B)$, já a Figura 7.1(b) apresenta a instância relacional de seqüências r' sobre o esquema relacional de seqüências $R'(A, B)$. Sendo que ambas instâncias armazenam as seqüências s_1 e s_2.

$\mathcal{ID}$	$\mathcal{P}$	A	B
1	1	a_1	b_1
1	2	a_2	b_2
1	3	a_3	b_3
2	1	a_1	b_1
2	2	a_4	b_4

(a) Instância r

$$\left\{ \begin{array}{l} < (a_1, b_1), (a_2, b_2), (a_3, b_3) >, \\ < (a_1, b_1), (a_4, b_4) > \end{array} \right\}$$

(b) Instância r'

Figura 7.1: Instâncias de relações no modelo relacional convencional e no modelo relacional de seqüências

O conjunto finito de esquemas relacionais de seqüências $\mathbb{R} = \{R_1, ..., R_n\}$ recebe o nome de *esquema de banco de dados de seqüências*. Uma *instância de banco de dados de seqüências* sobre um esquema $\mathbb{R}$ é uma função que associa uma relação de seqüências r_i a cada esquema relacional de seqüências R_i.

A Proposição 7.1 estabelece a equivalência entre os dois modelos relacionais.

Proposição 7.1. *Seja* $\mathbb{R} = \{R_1(\mathcal{ID}, \mathcal{P}, A_1, ..., A_k), ...\}$ *um esquema de banco de dados e* $\mathbb{R}' = \{R_1(A_1, ..., A_k), ...\}$ *o esquema associado sem os atributos* $\mathcal{ID}$ *e* $\mathcal{P}$*. Então para toda instância de* $\mathbb{R}$ *existe* uma única *instância de banco de dados de seqüências sobre* $\mathbb{R}'$ *e vice-versa: para cada instância de banco de dados de seqüências sobre* $\mathbb{R}'$*, existe uma única instância de banco de dados de tuplas convencionais sobre o esquema de banco de dados* $\mathbb{R}$*.*

De agora em diante, faremos uso dos dois modelos relacionais indistintivamente, sabendo que a passagem de uma modelagem para outra é feita de maneira inequívoca como indicado na Proposição 7.1.

A linguagem de consultas *TPref-SQL* que vamos introduzir no próximo capítulo permite executar consultas sobre bancos de dados de seqüências. Esta linguagem tem uma álgebra relacional associada, a álgebra *TPrefAR*, cuja sintaxe e semântica discutiremos no restante deste capítulo. A álgebra *TPrefAR* é constituída de onze operadores elementares que são executados sobre relações de seqüências. Estes operadores podem ser unários (operam sobre uma única relação de seqüências) e binários (operam sobre duas relações de seqüências produzindo uma nova relação de seqüências). Os operadores elementares são os seguintes:

1. Projeção (operação unária): permite projetar cada tupla das seqüências de uma relação sobre um determinado conjunto de atributos.

2. Seleção (operação unária): permite selecionar seqüências de uma relação de seqüências satisfazendo certas condições simples envolvendo o atributo $\mathcal{P}$ e os atributos de dados.
3. União (operação binária): permite unir dois conjuntos de seqüências sobre o mesmo esquema relacional.
4. Diferença (operação binária): permite extrair as seqüências que pertencem a um conjunto de seqüências e que não pertencem a outro conjunto de seqüências.
5. Produto Cartesiano (operação binária): permite realizar o produto cartesiano de seqüências produzindo um conjunto de seqüências com tuplas sobre um conjunto de atributos constituído da união dos atributos envolvidos nas seqüências de entrada. Os esquemas relacionais das relações de entrada devem ser disjuntos.
6. Renomeação (operação unária): permite renomear os atributos do esquema relacional de seqüências de entrada.
7. Subseqüência (operação unária): permite extrair a i-ésima tupla de uma seqüencia.
8. Concatenação (operação binária): permite concatenar cada seqüencia da primeira relação com cada seqüencia da segunda relação, produzindo um conjunto de seqüências mais longas.
9. BestSeqs-E (operação unária): permite selecionar as seqüências preferidas de acordo com uma teoria-tcp, utilizando a semântica "ou as melhores absolutas ou nada".
10. BestSeqs-N (operação unária): permite selecionar as seqüências preferidas de acordo com uma teoria-tcp, utilizando a semântica "ou as melhores absolutas, ou o que tiver".
11. BestSeqs-D (operação unária): permite selecionar as seqüências preferidas de acordo com uma teoria-tcp, utilizando a semântica "somente as melhores seqüências relativas à relação de entrada".

Na álgebra *TPrefAR* é possível realizar consultas sobre um banco de dados de seqüências através de expressões recursivas compostas de *operadores* e relações de seqüências. Estas expressões são denominadas *expressões da álgebra TPrefAR* e são definidas de maneira indutiva através da Definição 7.1.

Definição 7.1. (Expressão da álgebra *TPrefAR*) Seja $\mathbb{R} = \{R_1, ..., R_n\}$ um esquema de banco de dados de seqüências. Uma *expressão da álgebra TPrefAR* é definida da seguinte maneira:

- Se r_i é uma relação sobre um esquema relacional $R_i \in \mathbb{R}$, então r_i é uma expressão da álgebra *TPrefAR*;
- Se E é uma expressão da álgebra *TPrefAR* e *op* é um operador unário da álgebra *TPrefAR*, então $op(E)$ é uma expressão da álgebra *TPrefAR*;

- Se E_1 e E_2 são expressões da álgebra *TPrefAR* e *op* é um operador binário da álgebra *TPrefAR*, então (E_1) *op* (E_2) é uma expressão da álgebra *TPrefAR*.

O resultado de uma expressão da álgebra *TPrefAR* é sempre uma relação de seqüências que é obtida de acordo com os operadores contidos na expressão. Os operadores da álgebra *TPrefAR* são definidos da Seção 7.1 até a Seção 7.10.3 e na Seção 7.11 são expostas as considerações finais sobre este capítulo.

7.1 Operador de Projeção

O operador de projeção é um operador unário representado pelo símbolo π que permite extrair atributos de dados de uma relação. A expressão

$$\pi_{A_1,...,A_k}(r)$$

retorna uma relação r' cujo esquema relacional é $r'(A_1, ..., A_k)$, sendo que os atributos $\{A_1, ..., A_k\}$ devem obrigatoriamente ser atributos de dados do esquema da relação r. A relação r' possui as mesmas seqüências pertencentes a r, entretanto apenas os valores para os atributos $A_1, ..., A_k$ estão presentes em r'.

Por exemplo, considerando a relação r sobre o esquema $R3(A, B)$ apresentada na Figura 7.2(a), a expressão

$$\pi_A(r)$$

produz uma relação contendo apenas o atributo de dados A, como é exibido na Figura 7.2(b).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_2, b_2), (a_3, b_3) \rangle, \\ \langle (a_1, b_1), (a_4, b_4) \rangle \end{array} \right\}$$

(a) Relação r sobre o esquema $R3(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1), (a_2), (a_3) \rangle, \\ \langle (a_1), (a_4) \rangle \end{array} \right\}$$

(b) Relação produzida pela expressão $\pi_A(r)$

Figura 7.2: Aplicação do operador de projeção

7.2 Operador de Renomeação

O operador de renomeação é um operador unário representado pelo símbolo ρ que permite alterar o esquema relacional de uma relação (seu nome e os nomes de seus atributos de dados). A expressão

$$\rho_{S(A_1 \to A'_1,...,A_n \to A'_n)}(r)$$

retorna a relação r sob o nome s e com os atributos $A_1, ..., A_n$ renomeados para $A'_1, ..., A'_n$, respectivamente. Onde $A_1, ..., A_n$ são atributos de dados do esquema da relação r.

7.3 Operador de Seleção

O operador de seleção é um operador unário representado pelo símbolo σ que permite selecionar seqüências de tuplas de uma relação que atendam certas condições. As condições informadas para o operador de seleção devem ser obrigatoriamente atendidas pelas seqüências retornadas, por este motivo estas condições são chamadas de *restrições hard*. Uma expressão do tipo

$$\sigma_c(r)$$

retorna uma relação com o mesmo esquema relacional de r, porém as seqüências retornadas são apenas aquelas que atendam as condições c. Estas condições são combinações booleanas de *condições básicas*. Uma *condição básica*, por sua vez, é uma expressão do tipo $(\mathcal{P}\ op\ p) \wedge (A_i\ op\ v)$, onde A_i é um atributo de dado do esquema de r, $p \in \mathbb{N}$, $v \in \mathbf{Dom}(A_i)$ e $op \in \{=, \neq, <, \leq, >, \geq\}$

A utilização do operador de seleção sobre uma relação é ilustrada no Exemplo 7.2.

Exemplo 7.2. Considerando a relação r sobre o esquema $R4(A, B)$ exibida na Figura 7.3(a). A expressão

$$\sigma_{(\mathcal{P}=2\ \wedge\ A=a_3)}(r)$$

obtêm as seqüências que possuam o atributo A com valor igual a a_3 na posição 2. O resultado desta expressão é exibido na Figura 7.3(b).

$$\left\{ \begin{array}{l} \langle (a_1, b_1) \rangle, \\ \langle (a_1, b_1), (a_2, b_2) \rangle, \\ \langle (a_1, b_1), (a_3, b_1) \rangle \end{array} \right\}$$

(a) Relação r sobre o esquema $R4(A, B)$

$$\{\ \langle (a_1, b_1), (a_3, b_1) \rangle\ \}$$

(b) Relação produzida pela expressão $\sigma_{(\mathcal{P}=2\ \wedge\ A=a_3)}(r)$

Figura 7.3: Aplicação do operador de seleção

7.4 Operador de União

O operador de união é um operador binário que recebe duas relações e constrói uma nova relação contendo todas as seqüências de tuplas contidas nas duas relações recebidas. Uma operação de união $r_1 \cup r_2$ é válida somente quando as relações r_1 e r_2 são *compatíveis*, ou seja, quando r_1 e r_2 possuem o mesmo esquema relacional. O esquema relacional retornado será igual aos esquemas relacionais das duas relações recebidas.

Como exemplo, vamos considerar as relações r_5 sobre o esquema $R5(A, B)$ e r_6 sobre o esquema $R6(A, B)$ exibidas nas Figuras 7.4(a) e 7.4(a), respectivamente. A expressão $R5 \cup R6$ resultará na relação apresentada pela Figura 7.4(c).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle, \\ \langle (a_4, b_3) \rangle \end{array} \right\}$$

(a) Relação r_5 sobre o esquema $R5(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1, b_2) \rangle, \\ \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_4, b_1) \rangle \end{array} \right\}$$

(b) Relação r_6 sobre o esquema $R6(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle, \\ \langle (a_4, b_3) \rangle, \\ \langle (a_4, b_1) \rangle \end{array} \right\}$$

(c) Relação produzida pela expressão $r_5 \cup r_6$

Figura 7.4: Aplicação do operador de união

7.5 Operador de Diferença

O operador de diferença é um operador binário representado pelo sinal $-$ que permite obter as seqüências que pertencem a uma relação e não pertencem a outra relação. O operador de diferença necessita da mesma imposição que o operador de união, isto é, as duas relações recebidas pelo operador devem ser compatíveis. O esquema relacional retornado pelo operador de diferença será o mesmo esquema relacional das duas relações recebidas.

Como exemplo, vamos considerar novamente as relações r_5 e r_6 exibidas nas Figuras 7.4(a) e 7.4(b), respectivamente. A expressão $r_5 - r_6$ resultará na relação exibida pela Figura 7.5.

$$\left\{ \begin{array}{l} \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle, \\ \langle (a_4, b_3) \rangle \end{array} \right\}$$

Figura 7.5: Relação produzida pela expressão $r_5 - r_6$

7.6 Operador de Produto Cartesiano

O operador de produto cartesiano é um operador binário representado pelo símbolo $\times$ que permite juntar duas relações para obter uma nova relação contendo a combinação das seqüências das duas relações recebidas.

A operação de produto cartesiano $r_1 \times r_2$ é realizada combinando as posições de cada seqüência da relação r_1 com as respectivas posições de cada seqüência da relação r_2. Quando duas seqüências possuem tamanhos diferentes, o tamanho da menor seqüência é considerado e as posições excedentes são desconsideradas.

A relação resultante possui, além dos atributos $\mathcal{ID}$ e $\mathcal{P}$, todos os atributos de dados do esquema da relação r_1 (na mesma ordem em que eles ocorrem) seguidos pelos atributos de dados do esquema da relação r_2 (na mesma ordem em que eles ocorrem). Se houver algum atributo de dados A_i comum aos esquemas de r_1 e r_2, a segunda ocorrência do atributo A_i é renomeada para A_i', para que não ocorra conflito de nomes.

No Exemplo 7.3 pode ser visto como o operador de produto cartesiano é aplicado em duas relações.

Exemplo 7.3. Sejam as relações r_7 sobre o esquema $R7(A, B)$ apresentada na Figura 7.6(a) e r_8 sobre o esquema $R8$ ($\mathcal{ID}$, $\mathcal{P}$, A, C) apresentada na Figura 7.6(b). A expressão $r_7 \times r_8$ produz a relação r_9 sobre o esquema $R9(\mathcal{ID}$, $\mathcal{P}$, A, B, A', $C)$ apresentada da Figura 7.6(c).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_2, b_1) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle, \\ \langle (a_4, b_3) \rangle \end{array} \right\}$$

(a) Relação r_7 sobre o esquema $R7(A, B)$

$$\left\{ \begin{array}{l} \langle (a_2, c_1), (a_1, c_2), (a_3, c_3) \rangle, \\ \langle (a_4, c_1) \rangle \end{array} \right\}$$

(b) Relação r_8 sobre o esquema $R8$ ($\mathcal{ID}$, $\mathcal{P}$, A, C)

$$\left\{ \begin{array}{l} \langle (a_1, b_1, a_2, c_1), (a_2, b_1, a_1, c_2) \rangle, \\ \langle (a_1, b_1, a_4, c_1) \rangle, \\ \langle (a_3, b_1, a_2, c_1), (a_3, b_2, a_1, c_2), (a_3, b_3, a_3, c_3) \rangle, \\ \langle (a_3, b_1, a_4, c_1) \rangle, \\ \langle (a_4, b_3, a_2, c_1) \rangle, \\ \langle (a_4, b_3, a_4, c_1) \rangle \end{array} \right\}$$

(c) Relação r_9 sobre o esquema $R9(\mathcal{ID}, \mathcal{P}, A, B, A', C)$

Figura 7.6: Aplicação do operador de produto cartesiano

7.7 Operador de Subseqüência Simples

O operador de subseqüência simples é um operador unário que permite obter subseqüências de tamanho um a partir das seqüências de uma relação recebida. A expressão

$$SUBSEQUENCES_i(r)$$

produz uma nova relação contendo subseqüências de tamanho um obtidas a partir das seqüências de r. Para cada seqüência s de r é obtida uma subseqüências s' contendo a tupla da posição i de s. Caso alguma seqüência não possua a posição i, tal seqüência é desconsiderada. O esquema relacional da nova relação é o mesmo da relação recebida.

Como exemplo, vamos considerar a relação r_{10} sobre o esquema $R_{10}(A, B)$ exibida na Figura 7.7(a) e a expressão

$$SUBSEQUENCES_2(r_{10}).$$

O resultado desta expressão é a relação exibida na Figura 7.7(b).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_4, b_3), (a_4, b_1) \rangle \end{array} \right\}$$

(a) Relação r_{10} sobre o esquema $R_{10}(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_2) \rangle \end{array} \right\}$$

(b) Relação produzida pela expressão $SUBSEQUENCES_2(r_{10})$

Figura 7.7: Aplicação do operador de subseqüência

7.8 Operador de Concatenação de Seqüências

O operador de concatenação de seqüências é um operador binário que permite concatenar seqüências de duas relações. A expressão

$$r_1 \; CONCATENATION \; r_2$$

concatena cada seqüência da relação r_1 com cada seqüência da relação r_2. Desta maneira se r_1 possui m seqüências e r_2 possui n seqüências, a relação resultante possuirá $m \times n$ seqüências. A concatenação de duas seqüências $s = \langle t_1, ..., t_k \rangle$ e $s' = \langle t'_1, ..., t'_{k'} \rangle$ dá origem a uma nova seqüência $s'' = \langle t_1, ..., t_k, t'_1, ..., t'_{k'} \rangle$ contendo as tuplas de s seguidas pelas tuplas de s'.

A operação de concatenação de seqüências necessita da mesma imposição que a operação de união, isto é, as relações r_1 e r_2 devem ser compatíveis. O esquema relacional retornado é o mesmo esquema relacional das duas relações recebidas.

Como exemplo, vamos considerar as relações r_{12} e r_{13} exibidas nas Figuras 7.8(a) e 7.8(b), respectivamente. A expressão r_{12} $CONCATENATION$ r_{13} resulta na relação exibida pela Figura 7.8(c).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle \end{array} \right\}$$
(a) Relação r_{12}

$$\left\{ \begin{array}{l} \langle (a_1, b_2) \rangle, \\ \langle (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$
(b) Relação r_{13}

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2), (a_1, b_2) \rangle, \\ \langle (a_1, b_1), (a_1, b_2), (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3), (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3), (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$
(c) Relação produzida pela expressão r_{12} $CONCATENATION$ r_{13}

Figura 7.8: Aplicação do operador de concatenação de seqüências

7.9 Operadores de Seleção de Seqüências Ótimas

No Capítulo 3 consideramos três semânticas diferentes para a seleção das tuplas ótimas de uma relação de acordo com um conjunto de preferências e definimos um operador para cada uma destas semânticas no modelo relacional convencional. De forma análoga, na álgebra *TPrefAR* podemos considerar três semânticas diferentes para selecionar as seqüências de uma relação se seqüências considerando um conjunto de preferências. Estas preferências também podem ser chamadas de *restrições soft*, já os operadores de seleção de seqüências ótimas *tentam* atender as preferências, mas as seqüências retornadas podem atendem *parcialmente* às preferências ou não. No caso da álgebra *TPrefAR* vamos considerar um conjunto de preferências condicionais temporais expressas por meio de uma teoria-pct. Os operadores para seleção de seqüências ótimas são: **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**.

Operador **BestSeqs-E**$_\Phi(r)$ obtém as seqüências ótimas *absolutas* de acordo com uma teoria-pct Φ que estão presentes em uma relação r, ou seja, obtém as seqüências de tuplas que atendem exatamente as preferências de Φ e estão presentes em r. Quando nenhuma das seqüências ótimas absolutas segundo Φ está presente em r, o operador **BestSeqs-E** retorna uma relação vazia. O esquema relacional retornado é idêntico ao esquema relacional da relação recebida.

Como exemplo vamos considerar a relação r sobre o esquema $R_{15}(A, B)$ exibida na Figura 7.9(a) e a teoria-pct $\Phi_1 = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$, onde:

- $\varphi_1 : \mathbf{First} \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_2 : \mathbf{Prev}(A = a_1) \rightarrow (A = a_2) > (A = a_1)$;
- $\varphi_3 : \mathbf{Prev}(A = a_2) \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_4 : \mathbf{Prev}(A = a_2) \rightarrow (A = a_3) > (A = a_1)$;
- $\varphi_5 : (A = a_1) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi_6 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

Neste caso, a expressão **BestSeqs-E**$_{\Phi_1}(r)$ retorna a relação exibida na Figura 7.9(b), pois a relação r possui uma seqüência ótima absoluta segundo Φ_1. Entretanto, se consideramos a teoria-pct $\Phi_2 = \{\varphi'_1, \varphi'_2, \varphi'_3, \varphi'_4, \varphi'_5\}$, onde:

- $\varphi'_1 : \mathbf{First} \rightarrow (A = a_3) > (A = a_2)$;
- $\varphi'_2 : \mathbf{Prev}(A = a_3) \rightarrow (A = a_2) > (A = a_3)$;
- $\varphi'_3 : \mathbf{Prev}(A = a_2) \rightarrow (A = a_3) > (A = a_2)$;
- $\varphi'_4 : (A = a_3) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi'_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

A expressão **BestSeqs-E**$_{\Phi_2}(R_{15})$ retorna uma relação vazia, já que a relação $R15$ não possui nenhuma seqüência ótima absoluta segundo Φ_2.

$$\left\{ \begin{array}{l} s_1 = \langle (a_2, b_1), (a_1, b_2), (a_1, b_1), (a_1, b_2) \rangle, \\ s_2 = \langle (a_1, b_1), (a_2, b_2), (a_1, b_1) \rangle, \\ s_3 = \langle (a_2, b_1), (a_3, b_2), (a_3, b_3), (a_1, b_2) \rangle, \\ s_4 = \langle (a_1, b_1), (a_2, b_2), (a_3, b_1), (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$

(a) Relação r sobre o esquema $R_{15}(A, B)$

$$\{\ s_2 = \langle (a_1, b_1), (a_2, b_2), (a_3, b_2) \rangle\ \}$$

(b) Relação produzida pela expressão **BestSeqs-E**$_{\Phi_1}(r)$

Figura 7.9: Aplicação do operador **BestSeqs-E**

Operador **BestSeqs-N**$_\Phi(r)$ tenta obter as seqüências ótimas absolutas de acordo com Φ que estão presentes em uma relação r. Se nenhuma das seqüências ótimas absolutas estiver presente em r, então todas as seqüências de r são retornadas.

Como exemplo vamos considerar novamente a relação r exibida na Figura 7.9(a) e a teoria-pct Φ_1. Neste caso, a expressão **BestSeqs-N**$_{\Phi_1}(r)$ retorna a relação exibida na

Figura 7.9(b) (assim como aconteceu com o operador **BestSeqs-E**). No caso da expressão **BestSeqs-N**$_{\Phi_2}(r)$, a relação r não possui nenhuma seqüência ótima absoluta segundo Φ_2. Deste modo todas as seqüências de r são retornadas. Tanto o operador **BestSeqs-E** quanto o operador **BestSeqs-N** utilizam o algoritmo **BuildSeqs** para construir as seqüências ótimas absolutas que será apresentado com maiores detalhes no Capítulo 9.

Operador **BestSeqs-D**$_{\Phi}(r)$ retorna o conjunto de seqüências dominantes em r de acordo com Φ, ou seja,

$$\textbf{BestSeqs-D}_{\Phi}(r) = \{s \in r \mid \nexists s' \in r, s' >_{\Phi} s\}.$$

Isto significa que para toda seqüência s retornada pelo operador **BestSeqs-D** não existe nenhuma outra seqüência em r que seja preferida a s de acordo com Φ. Como exemplo vamos considerar a expressão **BestSeqs-D**$_{\Phi_2}(r)$, o resultado desta expressão é exibido na Figura 7.10.

$$\left\{ \begin{array}{l} s_3 = \langle (a_2, b_1), (a_3, b_2), (a_3, b_3), (a_1, b_2) \rangle, \\ s_4 = \langle (a_1, b_1), (a_2, b_2), (a_3, b_1), (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$

Figura 7.10: Relação produzida pela expressão **BestSeqs-D**$_{\Phi_2}(r)$

As seqüências s_3 e s_4 são as seqüências dominantes em r, pois $s_3 >_{\Phi_2} s_1$, $s_4 >_{\Phi_2} s_2$ e s_3 e s_4 são incomparáveis de acordo com Φ_2.

Mais detalhes sobre como as seqüências dominantes são obtidas pelo operador **BestSeqs-D** são apresentados no Capítulo 9 através do algoritmo **GetBestSeqs-D**.

7.10 Operadores Derivados

Além dos operadores elementares a álgebra *TPrefAR* possui alguns *operadores derivados* que podem ser obtidos através de operadores elementares. Estes operadores são especificados devido a sua comum utilização. Além disto, expressões mais simples da álgebra *TPrefAR* podem ser escritas utilizando estes operadores.

7.10.1 Operador de Subseqüência

O operador de subseqüência é um operador unário que permite obter subseqüências a partir das seqüências de uma relação recebida. A expressão

$$SUBSEQUENCES_{i,n}(r)$$

obtém uma nova relação contendo subseqüências a partir das seqüências de r. Para cada seqüência s de r é obtida uma subseqüência s' começando pela tupla i de s e considerando

n tuplas a partir de i. Caso alguma seqüência não possua todas as n tuplas a partir de i, são consideradas apenas as tuplas disponíveis. O esquema relacional da nova relação é o mesmo da relação recebida. O operador de subseqüência pode ser obtido utilizando o operador de concatenação e o subseqüência simples diversas vezes.

Como exemplo, vamos considerar a relação r sobre o esquema relacional $R_{10}(A, B)$ exibida na Figura 7.11(a) e a expressão

$$SUBSEQUENCES_{1,2}(r).$$

O resultado desta expressão é a relação exibida na Figura 7.11(b).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_4, b_3), (a_4, b_1) \rangle \end{array} \right\}$$

(a) Relação r sobre o esquema relacional $R_{10}(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2) \rangle \end{array} \right\}$$

(b) Relação produzida pela expressão $SUBSEQUENCES_{1,2}(r)$

Figura 7.11: Aplicação do operador de subseqüência

7.10.2 Operador de Interseção

O operador de interseção é um operador binário representado pelo símbolo $\cap$ que recebe duas relações r_1 e r_2 e retorna uma nova relação contendo as seqüências que estão presentes tanto em r_1 quanto em r_2. A interseção necessita da mesma imposição que a união, isto é, as relações r_1 e r_2 devem ser compatíveis. O esquema relacional retornado será o mesmo das relações recebidas.

O operador de interseção pode ser expresso através do operador de diferença da seguinte maneira $r_1 \cap r_2 = r_1 - (r_1 - r_2)$.

Como exemplo, vamos considerar as relações r_{17} e r_{18} exibidas nas Figuras 7.12(a) e 7.12(b), respectivamente. A expressão $r_{17} \cap r_{18}$ resulta na relação mostrada na Figura 7.12(c).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_3, b_1), (a_3, b_2), (a_3, b_3) \rangle, \\ \langle (a_4, b_3) \rangle \end{array} \right\}$$

(a) Relação r_{17}

$$\left\{ \begin{array}{l} \langle (a_4, b_1) \rangle, \\ \langle (a_1, b_2) \rangle, \\ \langle (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$

(b) Relação r_{18}

$$\left\{ \begin{array}{l} \langle (a_1, b_2) \rangle, \\ \langle (a_1, b_1), (a_1, b_2) \rangle \end{array} \right\}$$

(c) Relação produzida pela expressão $r_{17} \cap r_{18}$

Figura 7.12: Aplicação do operador de interseção

7.10.3 Operador de Junção

O operador de junção é um operador binário representado pelo símbolo $\bowtie$ que permite juntar duas relações para obter uma nova relação. A junção $r_1 \bowtie r_2$ da álgebra $TPrefAR$ é

semelhante a junção do modelo relacional convencional, ou seja, deve existir um conjunto de atributos de dados X não vazio que deve ser comum às relações r_1 e r_2.

A operação de junção $r_1 \bowtie r_2$ é realizada combinando as posições de cada seqüência da relação r_1 com as respectivas posições de cada seqüência da relação r_2. A combinação de duas posições é possível quando ambas posições possuem os mesmos valores para o conjunto de atributos X. Como as duas posições possuem os mesmos valores para o conjunto de atributos X, este conjunto de atributos aparecerá apenas uma vez na posição combinada. Quando duas seqüências possuem tamanhos diferentes, o tamanho da menor seqüência é considerado e as posições excedentes são desconsideradas.

O esquema relacional da relação produzida pela expressão $r_1 \bowtie r_2$ possuirá todos os atributos de dados do esquema de r_1 (na mesma ordem em que eles aparecem) e seguidos pelos atributos de dados do esquema R_2 (na mesma ordem em que eles aparecem) exceto pelos atributos de dados do esquema de r_2 que já apareceram em r_1.

Quando duas relações não possuem um conjunto de atributos comum pode ser usada a operação de renomeação antes da operação de junção. Repare que a operação de junção pode ser obtida utilizando-se a operação de produto cartesiano nas relações recebidas, seguida de uma seleção sobre o conjunto de atributos X, e finalmente, com uma projeção.

No Exemplo 7.4 pode ser visto como o operador de junção é aplicado em duas relações.

Exemplo 7.4. Neste exemplo consideraremos as relações r_{19} sobre o esquema $R_{19}(A, B)$ e r_{20} sobre o esquema R_{20} $(\mathcal{ID}, \mathcal{P}, A, C)$ apresentadas nas Figuras 7.13(a) e 7.13(b), respectivamente. A expressão $r_{19} \bowtie r_{20}$ resulta na relação r_{21} sobre o esquema $R_{21}(A, B, C)$ exibida na Figura 7.13(c).

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_1, b_2) \rangle, \\ \langle (a_4, b_2) \rangle, \\ \langle (a_1, b_3) \rangle \end{array} \right\}$$

(a) Relação r_{19} sobre o esquema $R_{19}(A, B)$

$$\left\{ \begin{array}{l} \langle (a_1, c_1), (a_4, c_3) \rangle, \\ \langle (a_1, c_4), (a_1, c_2), (a_1, c_3) \rangle \end{array} \right\}$$

(b) Relação r_{20} sobre o esquema $R_{20}(A, C)$

$$\left\{ \begin{array}{l} \langle (a_1, b_1, c_1) \rangle, \\ \langle (a_1, b_1, c_4), (a_1, b_2, c_2) \rangle, \\ \langle (a_1, b_3, c_1) \rangle, \\ \langle (a_1, b_3, c_4) \rangle \end{array} \right\}$$

(c) Relação r_{21} sobre o esquema $R_{21}(A, B, C)$

Figura 7.13: Aplicação do operador de junção

7.10.4 Operadores de Construção de Seqüências Ótimas

Como vimos na subseção anterior os operadores de seleção de seqüências ótimas são capazes de selecionar determinadas seqüências de uma relação atendendo certas preferências temporais. Outra tarefa interessante envolvendo preferências temporais é a construção de seqüências ótimas a partir de tuplas (objetos simples). Este tipo de operador é útil, por exemplo, quando um usuário possui uma coleção de músicas armazenadas e deseja construir uma lista de execução com estas músicas de acordo com suas preferências.

A construção de seqüências ótimas é feita de forma incremental, ou seja, uma seqüência de tamanho $k+1$ é construída acrescentando uma posição a uma seqüência de tamanho k.

Para cada posição i é considerado um subconjunto de preferências Γ_i e esta posição é preenchida com as tuplas que atendem Γ_i. Desta maneira, podemos enxergar a obtenção de cada posição como uma aplicação do operador **Best** proposto no Capítulo 3. Como o operador **Best** possui três semânticas distintas, vamos definir três operadores de construção de seqüências ótimas, são eles: **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D**.

Todos os três operadores **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D** retornam um esquema relacional idêntico ao da relação recebida e possuem a mesma sintaxe:

- **BuildBestSeqs-E**$_{\Phi,d,n}(r)$.
- **BuildBestSeqs-N**$_{\Phi,d,n}(r)$.
- **BuildBestSeqs-D**$_{\Phi,d,n}(r)$.

As seqüências ótimas são construídas de acordo com as preferências representadas pela teoria-pct Φ e devem possuir um tamanho igual a d. O parâmetro n é utilizando para limitar o número de seqüências a ser construído, pois dependendo da relação r, da teoria-pct Φ e do tamanho n pode ser construída uma quantidade gigantesca de seqüências e muitas vezes o usuários não deseja receber este tipo de resposta.

Quando $n = 0$ são construídas todas as seqüências possíveis e quando $n > 0$ são construídas no máximo n seqüências (em certas situações não é possível construir todas as n seqüências). Apesar dos operadores **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D** possuírem a mesma sintaxe, cada operador possui uma semântica diferente ao considerar as preferências e construir as seqüências ótimas.

Como uma seqüência de tamanho um pode ser vista como uma tupla, a construção de uma seqüência ótima considera uma seqüência de tamanho um para cada posição. Quando uma relação r possui seqüências com tamanho maior que um, as informações temporais são desconsideradas durante a construção. Isto é, se r possui uma seqüência s com tamanho igual a n, com $n > 1$, s é vista como n seqüências de tamanho um.

Os operadores de construção de seqüências ótimas podem ser obtidos através dos operadores **SUBSEQUENCES**, dos operadores de seleção de seqüências ótimas e do operador de concatenação. Com o operador de **SUBSEQUENCES** podemos quebrar as seqüências de uma relação em seqüências de tamanho um. Em seguida podemos selecionar estas seqüências de tamanho um para cada posição da seqüência a ser construída através dos operadores de seleção de seqüências ótimas. Os operadores de seleção de seqüências ótimas quando aplicados sobre uma relação de seqüências de tamanho um, considerando uma teoria-pct contendo regras cujas condições possuem apenas componentes de presente,

funcionam de forma equivalente ao operador **Best** apresentado no Capítulo 3. Podemos construir as seqüências ótimas combinando as posições obtidas pelos operadores de seleção de seqüências ótimas utilizando o operador **CONCATENATION**.

A construção de seqüências ótimas através do operador **BuildBestSeqs-E** considera as tuplas ótimas absolutas presentes em r de acordo com as preferências de cada posição. Como exemplo vamos considerar a relação r_{16} exibida na Figura 7.14(a) e a expressão

$$\textbf{BuildBestSeqs-E}_{\Phi_1,3,0}(r_{16}),$$

onde $\Phi_1 = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$ e:

- $\varphi_1 : \textbf{First} \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_2 : \textbf{Prev}(A = a_1) \rightarrow (A = a_2) > (A = a_1)$;
- $\varphi_3 : \textbf{Prev}(A = a_2) \rightarrow (A = a_1) > (A = a_2)$;
- $\varphi_4 : (A = a_1) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

O resultado da expressão é a relação da Figura 7.14(b).

$$\left\{ \begin{array}{l} \langle (a_2, b_2), (a_3, b_2), (a_1, b_1), (a_2, b_2) \rangle, \\ \langle (a_1, b_1), (a_3, b_2) \rangle, \\ \langle (a_2, b_2) \rangle \end{array} \right\}$$

(a) Relação r_{16}

$$\{ \ \langle (a_1, b_1), (a_2, b_2), (a_1, b_1) \rangle \ \}$$

(b) Relação produzida pela expressão **BuildBestSeqs-E**$_{\Phi_1,3,0}(R16)$

Figura 7.14: Aplicação do operador **BuildBestSeqs-E**

Quando a relação não possui nenhuma tupla ótima absoluta para alguma posição da seqüência a ser construída, o operador **BuildBestSeqs-E** retorna uma relação vazia. Como exemplo vamos considerar a expressão

$$\textbf{BuildBestSeqs-E}_{\Phi_2,3,0}(r_{16}),$$

onde $\Phi_2 = \{\varphi'_1, \varphi'_2, \varphi'_3, \varphi'_4, \varphi'_5\}$ e:

- $\varphi'_1 : \textbf{First} \rightarrow (A = a_2) > (A = a_3)$;
- $\varphi'_2 : \textbf{Prev}(A = a_3) \rightarrow (A = a_2) > (A = a_3)$;
- $\varphi'_3 : \textbf{Prev}(A = a_2) \rightarrow (A = a_3) > (A = a_2)$;
- $\varphi'_4 : (A = a_3) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi'_5 : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

Neste caso o operador **BuildBestSeqs-E** não construirá nenhuma seqüência ótima, pois a relação r_{16} não possui as tuplas ótimas absolutas para determinadas posições.

O operador **BuildBestSeqs-N** tenta construir as seqüências ótimas considerando as tuplas ótimas absolutas presentes em uma relação r de acordo com uma teoria-pct Φ. Quando uma posição não possui nenhuma tupla ótima absoluta presente em r, o operador considera todas as tuplas de r para esta posição.

Como exemplo vamos considerar a expressão

$$\textbf{BuildBestSeqs-N}_{\Phi_1,3,0}(r_{16}),$$

neste caso o operador **BuildBestSeqs-D** retornará o mesmo resultado do operador **BuildBestSeqs-E**, pois para todas as posições existem tuplas ótimas absolutas presentes em r_{16}. Mas se consideramos a expressão

$$\textbf{BuildBestSeqs-N}_{\Phi_2,3,0}r_{16}),$$

O operador **BuildBestSeqs-N** retornará a relação exibida na Figura 7.15, preenchendo as posições que não possuem tuplas ótimas absolutas com qualquer tupla de r_{16}.

$$\left\{\begin{array}{c}
\langle(a_2,b_2),(a_1,b_1),(a_1,b_1)\rangle,\\
\langle(a_2,b_2),(a_1,b_1),(a_2,b_2)\rangle,\\
\langle(a_2,b_2),(a_1,b_1),(a_3,b_2)\rangle,\\
\langle(a_2,b_2),(a_2,b_2),(a_1,b_1)\rangle,\\
\langle(a_2,b_2),(a_2,b_2),(a_2,b_2)\rangle,\\
\langle(a_2,b_2),(a_2,b_2),(a_3,b_2)\rangle,\\
\langle(a_2,b_2),(a_3,b_2),(a_2,b_2)\rangle
\end{array}\right\}$$

Figura 7.15: Relação produzida pela expressão $\textbf{BuildBestSeqs-N}_{\Phi_2,3,0}(r_{16})$

Uma seqüência construída pelo operador **BuildBestSeqs-D** possui em cada posição i uma tupla de r que é dominante segundo Γ_i. Ou seja, para cada posição i são consideradas as tuplas dominantes de r de acordo com as preferências relativas a posição i. Como exemplo vamos considerar a expressão

$$\textbf{BuildBestSeqs-D}_{\Phi_3,3,0}(r_{16}),$$

onde $\Phi_3 = \{\varphi_1'', \varphi_2'', \varphi_3'', \varphi_4'', \varphi_5''\}$ e:

- $\varphi_1'' : \textbf{First} \rightarrow (A = a_3) > (A = a_2)$;
- $\varphi_2'' : \textbf{Prev}(A = a_3) \rightarrow (A = a_2) > (A = a_3)$;
- $\varphi_3'' : \textbf{Prev}(A = a_2) \rightarrow (A = a_3) > (A = a_2)$;
- $\varphi_4'' : (A = a_3) \rightarrow (B = b_1) > (B = b_2)$;
- $\varphi_5'' : (A = a_2) \rightarrow (B = b_2) > (B = b_1)$.

Neste caso, o operador **BuildBestSeqs-D** retornará a relação exibida na Figura 7.16.

$$\left\{ \begin{array}{l} \langle (a_1, b_1), (a_2, b_2), (a_1, b_1) \rangle, \\ \langle (a_1, b_1), (a_2, b_2), (a_3, b_2) \rangle, \\ \langle (a_3, b_2), (a_2, b_2), (a_1, b_1) \rangle, \\ \langle (a_3, b_2), (a_2, b_2), (a_3, b_2) \rangle \end{array} \right\}$$

Figura 7.16: Relação produzida pela expressão $\mathbf{BuildBestSeqs\text{-}D}_{\Phi_{3,3,0}}(r_{16})$

No Capítulo 9 daremos mais detalhes sobre a construção de seqüências ótimas pelos operadores **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D** através dos algoritmos propostos para estes operadores.

7.11 Considerações Finais

Neste capítulo foi proposto um modelo relacional composto por relações especiais para armazenar seqüências de tuplas denominadas relações de seqüências. Para realizar operações sobre este tipo de relação for definida uma a álgebra *TPrefAR*.

A álgebra *TPrefAR* é composta de diversos operadores que realizam operações semelhantes a álgebra relacional e operadores capazes de tratar preferências condicionais temporais, entretanto estes operadores foram definidos para realizar operações exclusivamente sobre seqüências.

Capítulo 8

A Linguagem de Consulta TPref-SQL

De posse da álgebra *TPrefAR* definida no Capítulo 7 será especificada a linguagem de consulta *TPref-SQL*. A especificação desta linguagem permitirá a formulação de consultas sobre relações de seqüências. Desta maneira, diversas operações sobre seqüências, inclusive seleção e construção de seqüências ótimas, poderão ser realizadas através de consultas *TPref-SQL*.

A correspondência entre consultas na linguagem *TPref-SQL* e expressões da álgebra *TPrefAR* é de extrema importância para que sejam traçados *planos de execução de consultas* otimizados, ou seja, a obtenção da melhor maneira de se executar uma consulta.

Apesar de tarefas de otimização de planos de execução estarem fora do contexto deste trabalho, neste capítulo será mostrado como consultas na linguagem *TPref-SQL* podem ser decompostas em operadores da álgebra *TPrefAR*, para que trabalhos futuros possam utilizar estas informações.

No decorrer deste capítulo serão apresentadados diversos exemplos de consultas *TPref-SQL* que utilizarão as seguintes relações:

- Relação musicas sobre o esquema M(NOME, CANTOR, RITMO, DURACAO, EPOCA) exibida na Figura 8.1;
- Relação nacionais sobre o esquema N(NOME, CANTOR, RITMO, DURACAO, EPOCA) exibida na Figura 8.2;
- Relação internacionais sobre o esquema I(NOME, CANTOR, RITMO, DURACAO, EPOCA) exibida na Figura 8.3;
- Relação listas sobre o esquema L(NOME, CANTOR, RITMO, DURACAO, EPOCA) exibida na Figura 8.4;
- Relação cantores sobre o esquema C(CANTOR, GENERO) exibida na Figura 8.5.

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Dias Melhores	Jota Quest	tranqüilo	longa	contemporânea
2	1	Sossego	Tim Maia	tranqüilo	media	antiga
3	1	Brasil	Antonio Carlos Jobim	tranqüilo	longa	antiga
4	1	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
5	1	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
6	1	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
7	1	Hands Clean	Alanis Morissette	tranqüilo	media	contemporânea
8	1	Immortality	Celine Dion	tranqüilo	media	contemporânea
9	1	I could fall in love	Celine Dion	tranqüilo	breve	contemporânea
10	1	I'll try	Alan Jackson	tranqüilo	breve	contemporânea

Figura 8.1: Relação musicas sobre o esquema M(NOME, CANTOR, RITMO, DURACAO, EPOCA)

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Dias Melhores	Jota Quest	tranqüilo	longa	contemporânea
2	1	Sambassim	Djavan	tranqüilo	media	contemporânea
3	1	Mentiras	Adriana Calcanhoto	tranqüilo	breve	contemporânea
4	1	Sossego	Tim Maia	tranqüilo	media	antiga
5	1	Brasil	Antonio Carlos Jobim	tranqüilo	longa	antiga
6	1	Mal de Mim	Djavan	tranqüilo	breve	contemporânea
7	1	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
8	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
9	1	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
10	1	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
11	1	Blusa Amarela	Gino e Geno	tranqüilo	breve	antiga
12	1	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea

Figura 8.2: Relação nacionais sobre o esquema N(NOME, CANTOR, RITMO, DURACAO, EPOCA)

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Dani Califórnia	Red Hot Chili Peppers	movimentado	media	contemporânea
2	1	Crazy	Alanis Morissette	movimentado	breve	contemporânea
3	1	Big Mistake	Alanis Morissette	tranqüilo	breve	contemporânea
4	1	Hands Clean	Alanis Morissette	tranqüilo	media	contemporânea
5	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
6	1	Immortality	Celine Dion	tranqüilo	media	contemporânea
7	1	I could fall in love	Celine Dion	tranqüilo	breve	contemporânea
8	1	Given up	Linkin Park	movimentado	breve	contemporânea
9	1	I'll try	Alan Jackson	tranqüilo	breve	contemporânea
10	1	When love comes around	Alan Jackson	tranqüilo	longa	contemporânea
11	1	Drive	Alan Jackson	movimentado	longa	antiga

Figura 8.3: Relação internacionais sobre o esquema I(NOME, CANTOR, RITMO, DURACAO, EPOCA)

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
1	2	Panela Velha	Sérgio Reis	movimentado	breve	antiga
1	3	I'll try	Alan Jackson	tranqüilo	breve	contemporânea
2	1	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
2	2	Drive	Alan Jackson	movimentado	longa	antiga
2	3	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
3	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
3	2	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
3	3	I'll try	Alan Jackson	tranqüilo	breve	contemporânea
3	4	Drive	Alan Jackson	movimentado	longa	antiga
3	5	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
4	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
4	2	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
4	3	Dani Califórnia	Red Hot Chili Peppers	movimentado	media	contemporânea
5	1	Given up	Linkin Park	movimentado	breve	contemporânea
5	2	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
5	3	Crazy	Alanis Morissette	movimentado	breve	contemporânea
6	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
6	2	Dani Califórnia	Red Hot Chili Peppers	movimentado	media	contemporânea
6	3	Crazy	Alanis Morissette	movimentado	breve	contemporânea
6	4	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
6	5	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
7	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
7	2	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
8	1	Sossego	Tim Maia	tranqüilo	media	antiga
8	2	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
9	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
9	2	Sossego	Tim Maia	tranqüilo	media	antiga
9	3	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga

Figura 8.4: Relação listas sobre o esquema L(NOME, CANTOR, RITMO, DURACAO, EPOCA)

$\mathcal{ID}$	$\mathcal{P}$	CANTOR	GENERO
1	1	Jota Quest	rock
2	1	Red Hot Chili Peppers	rock
3	1	Alanis Morissette	rock
4	1	Linkin Park	rock
5	1	Djavan	mpb
6	1	Adriana Calcanhoto	mpb
7	1	Tim Maia	mpb
8	1	Antonio Carlos Jobim	mpb
9	1	Djavan	mpb
10	1	Sérgio Reis	sertanejo
11	1	Gino e Geno	sertanejo
12	1	Celine Dion	pop
13	1	Alan Jackson	country

Figura 8.5: Relação cantores sobre o esquema C(CANTOR, GENERO)

8.1 Sintaxe

Basicamente o bloco de uma consulta *TPref-SQL* possui a seguinte sintaxe:

```
SELECT [BUILDBESTSEQS | SUBSEQUENCES] <atributos>
FROM <relações>
WHERE <condições>
[[MAX m | START s] POSITIONS p]
ACCORDING TO [EXACT [NO EMPTY ]]
PREFERENCES <preferências>
[[UNION | EXCEPT | INTERSECT | CONCATENATION]
SELECT ...]
```

Cada uma das cláusulas de uma consulta são explicadas mais detalhadamente das subseções que se seguem.

8.1.1 As Cláusulas SELECT e FROM

A cláusula *SELECT* corresponde à operação de projeção e deve ser usada em conjunto com a cláusula *FROM* para indicar sobre qual relação será realizada a projeção. Basicamente uma consulta do tipo:

SELECT $A_1, ..., A_k$
FROM r

corresponde a expressão $\pi_{A_1,...,A_k}(r)$ na álgebra *TPrefAR*.

O símbolo * pode ser usado para retornar todos os atributos de uma relação, por exemplo, a obtenção de todos os atributos da relação MUSICAS pode ser feita com a consulta:

```
SELECT *
FROM MUSICAS.
```

Quando a cláusula *FROM* recebe mais de uma relação a operação de produto cartesiano é realizada por padrão. Basicamente uma consulta do tipo:

SELECT $A_1, ..., A_k$
FROM $r_1, ..., r_n$

corresponde a expressão $\pi_{A_1,...,A_k}(r_1 \times ... \times r_n)$ na álgebra *TPrefAR*.

Entretanto a cláusula *FROM* pode realizar a operação de junção também, isto é feito através de uma determinada condição colocada na cláusula *WHERE*. Basicamente uma consulta do tipo:

SELECT $A_1, ..., A_k$
FROM r_1, r_2
WHERE $r_1.A_n = r_2.A_n$

corresponde a expressão $\pi_{A_1,...,A_k}(r_1 \bowtie_{A_n} r_2)$ na álgebra *TPrefAR*, onde $\bowtie_{A_n}$ indica a junção entre r_1 e r_2 através do atributo A_n.

O Exemplo 8.1 mostra uma consulta realizando uma junção entre duas tabelas.

Exemplo 8.1. Neste exemplo vamos considerar a consulta:

```
SELECT NOME, GENERO
FROM NACIONAIS, CANTORES
WHERE MUSICAS.CANTOR = CANTORES.CANTOR
```

que obtém o nome das músicas da relação NACIONAIS com seus respectivos gêneros. O resultado desta consulta é a relação r_{22} exibida na Figura 8.6.

$\mathcal{ID}$	$\mathcal{P}$	NOME	GENERO
1	1	Dias Melhores	rock
2	1	Sossego	mpb
3	1	Brasil	mpb
4	1	Menino da Porteira	sertanejo
5	1	As Águas do São Francisco	sertanejo
6	1	Apaixonado por você	sertanejo
7	1	Hands Clean	rock
8	1	Immortality	pop
9	1	I could fall in love	pop
10	1	I'll try	Country

Figura 8.6: Relação r_{22}

8.1.2 A Cláusula WHERE

A cláusula *WHERE* em conjunto com as cláusulas *SELECT* e *FROM* corresponde a operação de seleção da álgebra *TPrefAR*. O formato das *condições* é semelhante ao formato das fórmulas aceitas pela operação de seleção considerando algumas conversões.

São aceitas fórmulas booleanas formadas pelos conectivos *AND*, *OR* e *NOT* (que correspondem exatamente aos conectivos $\wedge$, $\vee$ e $\neg$ da operação de seleção) e por fórmulas básicas equivalentes às condições básicas definidas para a operação de seleção. Uma condição ($\mathcal{P}$ *op* $p \wedge A$ *op* a) para a operação de seleção equivale a fórmula (P *op* p *AND* A *op* a) para a cláusula *WHERE*.

Por exemplo, para obtermos as seqüências da tabela LISTAS com músicas tranqüilas na posição dois ou com músicas contemporâneas em qualquer posição podemos usar a consulta:

```
SELECT *
FROM LISTAS
WHERE (P = 2 AND RITMO = 'tranqüilo')
OR (P >= 1 AND EPOCA = 'contemporânea').
```

O resultado desta consulta é a relação r_{23} exibida na Figura 8.7.

8.1.3 A Cláusula AS

A renomeação de atributos e relações é feita com a cláusula *AS* da seguinte forma: *nome_antigo AS nome_nome*. Como exemplo vamos considerar a consulta:

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
5	1	Given up	Linkin Park	movimentado	breve	contemporânea
5	2	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
5	3	Crazy	Alanis Morissette	movimentado	breve	contemporânea
6	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
6	2	Dani Califórnia	Red Hot Chili Peppers	movimentado	media	contemporânea
6	3	Crazy	Alanis Morissette	movimentado	breve	contemporânea
6	4	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
6	5	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
7	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
7	2	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
8	1	Sossego	Tim Maia	tranqüilo	media	antiga
8	2	As Águas do São Francisco	Gino e Geno	tranqüilo	longa	antiga
9	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
9	2	Sossego	Tim Maia	tranqüilo	media	antiga
9	3	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga

Figura 8.7: Relação r_{23}

```
SELECT M.NOME AS MUSICA, C.GENERO
FROM MUSICAS AS M, CANTOR AS C
WHERE M.CANTOR = C.CANTOR
AND M.RITMO = 'movimentado'.
```

O atributo NOME da relação MUSICAS é renomeado para MUSICA, as tabelas MUSICAS e CANTORES são renomeadas para M e C, respectivamente. Depois de renomeados os campos e tabelas pode ser referenciados por seus novos nomes nas cláusulas *SELECT* e *WHERE*.

8.1.4 As Cláusulas UNION, EXCEPT, INTERSECT e CONCATENATION

Na linguagem *TPref-SQL* as operações de união, diferença, interseção e concatenação são realizadas com a utilização das cláusulas *UNION*, *EXCEPT*, *INTERSECT*, *CONCATENATION*, respectivamente. Estas operações são obtidas através de duas consultas intercaladas com uma das cláusulas. Assim como ocorre na álgebra *TPrefAR* as relações resultantes destas duas consultas devem ser compatíveis.

Por exemplo, para realizar uma operação de união entre as seqüências presentes nas relações NACIONAIS e INTERNACIONAIS pode ser feita a consulta:

```
SELECT *
FROM NACIONAIS
UNION
SELECT *
FROM INTERNACIONAIS.
```

Já uma operação de interseção entre as seqüências de tais relações pode ser feita com a consulta:

```
SELECT *
FROM NACIONAIS
INTERSECT
SELECT *
FROM INTERNACIONAIS.
```

Consultas para realizar operações de diferença e concatenação podem ser realizadas de maneira análoga.

8.1.5 A Cláusula SUBSEQUENCES

Uma consulta para obter as subseqüências de uma relação pode ser realizada com o acréscimo da cláusula *SUBSEQUENCES* após a cláusula *SELECT*. A posição inicial e a dimensão das subseqüências a serem obtidas são definidas pelas cláusulas *START* e *POSITIONS* que são inseridas a após a cláusula *WHERE*.

O Exemplo 8.2 exibe uma consulta que obtém subseqüências e as concatena simultaneamente.

Exemplo 8.2. Neste exemplo vamos considerar a consulta:

```
SELECT SUBSEQUENCES *
FROM LISTAS
WHERE (T = 2 AND DURACAO = 'media')
START 1 POSITIONS 1
CONCATENATION
SELECT SUBSEQUENCES *
FROM LISTAS
WHERE (T = 2 AND DURACAO = 'media')
START 3 POSITIONS 2
```

A consulta antes da cláusula *CONCATENATION* obtém as subseqüências da relação LISTAS contendo apenas a primeira posição para as seqüências com duração média na segunda posição. A consulta depois da cláusula *CONCATENATION* obtém as subseqüências da relação LISTAS contendo duas posições a partir da terceira posição (ou seja, a terceira e a quarta posição) para as seqüências com duração média na segunda posição. A palavra *CONCATENATION* realiza a concatenação das seqüências obtidas através destas duas consultas, como é mostrado na relação r_{24} exibida na Figura 8.8.

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
1	2	Crazy	Alanis Morissette	movimentado	breve	contemporânea
1	3	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
2	1	Panela Velha	Sérgio Reis	movimentado	breve	antiga
2	2	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga
3	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
3	2	Crazy	Alanis Morissette	movimentado	breve	contemporânea
3	3	Pinga em Mim	Sérgio Reis	movimentado	breve	antiga
4	1	My heart will go on	Celine Dion	tranqüilo	longa	antiga
4	2	Menino da Porteira	Sérgio Reis	tranqüilo	longa	antiga

Figura 8.8: Relação r_{24}

8.1.6 A Cláusula ACCORDING TO PREFERENCES

A cláusula *ACCORDING do PREFERENCES* é utilizada para que preferências temporais possam ser incorporadas as consultas. Isto é, quando uma consulta *TPref-SQL* possui regras de preferências especificadas através da cláusula *ACCORDING TO PREFERENCES*, os operadores de seleção de seqüências ótimas serão utilizados para resolver a consulta.

A escolha dos operadores é feita de acordo com as cláusulas *EXACT NO EMPTY*, posicionadas entre *ACCORDING TO* e *PREFERENCES*. Se for utilizada apenas a palavra *EXACT* será utilizado o operador **BestSeqs-E** para resolver a consulta. As palavras *EXACT NO EMPTY* implicam na utilização do operadores **BestSeqs-N** e se nenhuma palavra for especificada será utilizado o operador **BestSeqs-D**.

As regras de preferências possuem um formato semelhante às regras-pct definidas pelo formalismo lógico *TPref* obedecendo a algumas conversões. Os operadores temporais **First** e **Prev** permanecem sem conversão, os demais obedecem as seguintes conversões:

- ♦ é convertido para *SOMEPREV*;
- ■ é convertido para *ALWAYSPREV*;

Além disto, uma implicação $(A = a) \rightarrow (B = b) > (B = b')$ é convertida para *IF* A = a *THEN* (B = b) > (B = b'). As regras são concatenadas através do conectivo *AND*.

No Exemplo 8.3 é mostrada uma consulta que faz uma seleção de seqüências ótimas.

Exemplo 8.3. Neste exemplo vamos considerar a consulta:

```
SELECT *
FROM (

     SELECT GENERO, RITMO, DURACAO
     FROM LISTAS, CANTORES
     WHERE LISTAS.CANTOR = CANTORES.CANTOR

)
ACCORDING TO PREFERENCES
IF FIRST THEN
(RITMO = 'tranqüilo') > (RITMO = 'movimentado') AND
IF PREV RITMO = 'tranqüilo' THEN
(RITMO = 'movimentado') > (RITMO = 'tranqüilo') AND
IF PREV RITMO = 'movimentado' THEN
(RITMO = 'tranqüilo') > (RITMO = 'movimentado') AND
IF RITMO = 'movimentado' THEN
(DURACAO = 'breve') > (DURACAO = 'longa') AND
IF RITMO = 'tranqüilo' THEN
(DURACAO = 'longa') > (DURACAO = 'breve')
```

Podemos notar que é utilizada uma subconsulta na qual é realizada uma junção entre as relações LISTAS e CANTORES. Ainda na subconsulta é feita uma projeção obtendo os

campos GENERO, RITMO e DURACAO. Após a subconsulta é realizada uma operação de seleção de seqüências ótimas cujo resultado é a relação r_{25} exibida na Figura 8.9.

$\mathcal{ID}$	$\mathcal{P}$	GENERO	RITMO	DURACAO
1	1	sertanejo	tranqüilo	longa
1	2	sertanejo	movimentado	breve
1	3	country	tranqüilo	breve
2	1	rock	movimentado	breve
2	2	sertanejo	movimentado	longa
2	3	rock	movimentado	breve
3	1	pop	tranqüilo	longa
3	2	sertanejo	tranqüilo	longa
4	1	mpb	tranqüilo	media
4	2	sertanejo	tranqüilo	longa
5	1	pop	tranqüilo	longa
5	2	mpb	tranqüilo	media
5	3	sertanejo	tranqüilo	longa

Figura 8.9: Relação r_{25}

8.1.7 A Cláusula BUILDBESTSEQS

Uma consulta para construir as seqüências ótimas é feita com o uso da cláusula *BUILDBESTSEQS* inclusa após a cláusula *SELECT*. O número máximo de seqüências a serem construídas é definido através da cláusula *MAX* inclusa após a cláusula *WHERE*. Já o tamanho das seqüências a serem construídas é definido através das cláusula *POSITIONS* que também deve ser incluída após a cláusula *WHERE*. As regras de preferências são informadas com as cláusulas *ACCORDING TO PREFERENCES*.

Uma consulta para construir seqüências ótimas é exibida no Exemplo 8.4.

Exemplo 8.4. Neste exemplo vamos considerar a consulta:

```
SELECT BUILDBESTSEQS *
FROM MUSICAS
MAX 2 POSITIONS 3
ACCORDING TO EXACT NO EMPTY PREFERENCES
IF FIRST THEN
(RITMO = 'tranqüilo') > (RITMO = 'movimentado') AND
IF RITMO = 'tranqüilo' THEN
(DURACAO = 'longa') > (DURACAO = 'breve') AND
IF RITMO = 'movimentado' THEN
(DURACAO = 'breve') > (DURACAO = 'longa') AND
IF PREV RITMO = 'movimentado' THEN
(RITMO = 'tranqüilo') > (RITMO = 'movimentado') AND
IF PREV RITMO = 'tranqüilo' THEN
(RITMO = 'movimentado') > (RITMO = 'tranqüilo')
```

O resultado desta consulta é a relação R26 exibida na Figura 8.10.

$\mathcal{ID}$	$\mathcal{P}$	NOME	CANTOR	RITMO	DURACAO	EPOCA
1	1	Dias Melhores	Jota Quest	tranqüilo	longa	contemporânea
1	2	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
1	3	Dias Melhores	Jota Quest	tranqüilo	longa	contemporânea
2	1	Brasil	Antonio Carlos Jobim	tranqüilo	longa	antiga
2	2	Apaixonado por você	Gino e Geno	movimentado	longa	contemporânea
2	3	Brasil	Antonio Carlos Jobim	tranqüilo	longa	antiga

Figura 8.10: Relação R26

8.2 Processamento de Consultas na Linguagem TPref-SQL

O processamento de consultas da linguagem *TPref-SQL* apresenta diversas peculiaridades, isto porque as relações armazenam seqüências e não tuplas como no modelo relacional. Como este não é o objetivo deste trabalho não abordamos questões relacionadas a planos de execução de consultas *TPref-SQL*, mas destacamos este assunto como uma importante tarefa para trabalhos futuros.

8.3 Considerações finais

Neste capítulo foi especificada a linguagem de consulta para banco de dados *TPref-SQL* baseada na álgebra *TPrefAR*. Foi definida a sintaxe desta linguagem através de diversos exemplos e foi exibido o relacionamento entre consultas *TPref-SQL* e expressões da álgebra *TPrefAR*. E, também, foram apresentados diversos exemplos de consultas com esta linguagem.

Capítulo 9

Algoritmos e Implementação

Neste capítulo serão propostos algoritmos para resolver os operadores de seleção de seqüências ótimas e de construção de seqüências ótimas apresentados no capítulo 7. Além disto será apresentada uma implementação de um protótipo para um fragmento da linguagem de consulta *TPref-SQL*.

Na Seção 9.1 serão propostos os algoritmos para resolver os operadores de seleção de seqüências ótimas **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**. Em seguida, na Seção 9.2 serão propostos os algoritmos para resolver os operadores de construção de seqüências ótimas **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D**. Já a Seção 9.3 descreve como foi implementado o protótipo para um fragmento da linguagem *TPref-SQL*. E por fim, na Seção 9.4 são expostas as considerações finais acerca deste capítulo.

9.1 Algoritmos para Seleção de Seqüências Ótimas

Nas subseções a seguir serão apresentados os algoritmos **GetBestSeqs-E**, **GetBestSeqs-N** e **GetBestSeqs-D** responsáveis por resolver os operadores de seleção de seqüências ótimas **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**, respectivamente.

9.1.1 Algoritmo GetBestSeqs-E

O algoritmo **GetBestSeqs-E** foi desenvolvido para resolver o operador **BestSeqs-E**. Este algoritmo recebe como entrada uma relação r, uma teoria-pct Φ consistente e retorna as seqüências ótimas absolutas segundo Φ que estão presentes em r, como é mostrado no esquema da Figura 9.1.

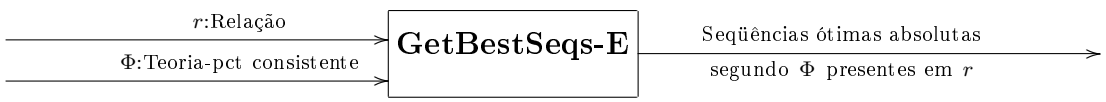


Figura 9.1: Esquema do algoritmo **GetBestSeqs-E**

O pseudo-código para o algoritmo **GetBestSeqs-E** é exibido na Figura 9.2.

GetBestSeqs-E(r, Φ) :

1: $S = \emptyset$
2: $m = |biggerSeq(r)|$ // Obtém o tamanho da maior seqüência de r
3: **for** $i = 1$ **to** m **do** // Laço para considerar as seqüências de tamanho 1 a m
4: $S_i =$ **BuildSeqs**$(\Phi, \langle\rangle, i)$ // Constrói as seqüências ótimas absolutas de tamanho i segundo Φ
5: $S_i = select(r, S_i)$ // Obtém as seqüências ótimas presentes em r
6: $S = S \cup S_i$
7: **end for**
8: **return** S

Figura 9.2: Algoritmo **GetBestSeqs-E**

O algoritmo **GetBestSeqs-E** utiliza a subrotina **BuildSeqs** para construir as seqüências ótimas absolutas e verifica quais das seqüências construídas estão presentes em r. A subrotina **BuildSeqs** é chamada várias vezes para que todos os tamanhos de seqüências possam ser considerados para a relação r.

A complexidade do algoritmo **GetBestSeqs-E** é de $O(kn^{3nk} + km)$., onde $|\Phi| = n$, $|r| = m$ e k é o tamanho da maior tupla em r. Isto acontece porque a complexidade da subrotina **BuildSeqs** é de $O(n^{3kn})$, a subrotina *select* possui o custo de m e estas subrotinas são executadas k vezes pelo algoritmo.

O Exemplo 9.1 exibe uma execução do algoritmo **GetBestSeqs-E**.

Exemplo 9.1. Neste exemplo vamos considerar a relação m_6 sobre o esquema $M_6(N, C, G)$ exibida na Figura 9.3, onde N, C e G são os atributos nome, cantor e gênero, respectivamente. Vamos considerar também as preferências musicais de um usuário expressas através da teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde:

- $\varphi_1 :$ **First** $\rightarrow (G = r) > (G = c)$;
- $\varphi_2 :$ **First** $\rightarrow (G = p) > (G = c)$;
- $\varphi_3 :$ **Prev**$(G = r) \rightarrow (G = p) > (G = r)$;
- $\varphi_4 :$ **Prev**$(G = p) \rightarrow (G = r) > (G = p)$.

As letras p, r e c representam os valores pop, rock e clássico, respectivamente. Deste modo, o algoritmo **GetBestSeqs-E** é executado da seguinte maneira:

- Obtém o tamanho da maior seqüência em m_6, $m = 3$;
- Constrói as seqüências ótimas absolutas de tamanho 1, $s_1 = \langle(r)\rangle$ e $s_2 = \langle(p)\rangle$;
- Nenhuma das seqüências ótimas de tamanho 1 está presente em m_6;
- Constrói as seqüências ótimas absolutas de tamanho 2, $s_{1,1} = \langle(r), (p)\rangle$ e $s_{2,1} = \langle(p), (r)\rangle$;

- A relação m_6 possui a seqüência ótima absoluta $s_{1,1}$;
- $S = \{\langle (m_5, c_5, r), (m_7, c_1, p)\rangle\}$
- Constrói as seqüências ótimas absolutas de tamanho 3, $s_{1,1,1} = \langle (r), (p), (r)\rangle$ e $s_{2,1,1} = \langle (p), (r), (p)\rangle$;
- A relação m_6 possui a seqüência ótima absoluta $s_{2,1,1}$;
- Retorna $S = \{\langle (m_5, c_5, r), (m_7, c_1, p)\rangle, \langle (m_8, c_7, p), (m_6, c_6, r), (m_9, c_7, p)\rangle\}$.

$$\left\{\begin{array}{l}\langle (m_1, c_2, i)\rangle, \\ \langle (m_2, c_3, c), (m_4, c_5, r), (m_7, c_1, p)\rangle, \\ \langle (m_5, c_5, r), (m_7, c_1, p)\rangle, \\ \langle (m_8, c_7, p), (m_6, c_6, r), (m_9, c_7, p)\rangle, \\ \langle (m_5, c_1, r), (m_3, c_4, c)\rangle, \\ \langle (m_3, c_4, c)\rangle, \\ \langle (m_2, c_3, c), (m_8, c_7, p)\rangle, \\ \langle (m_7, c_1, p), (m_9, c_7, p)\rangle\end{array}\right\}$$

Figura 9.3: Relação m_6 sobre o esquema $M_6(N, C, G)$

Subrotina BuildSeqs

A subrotina **BuildSeqs** é um algoritmo que foi proposto no trabalho de [de Amo e Giacometti 2007] para construir as seqüências ótimas absolutas de acordo com uma teoria-pct Φ consistente. Além de uma teoria-pct Φ consistente, a subrotina recebe uma seqüência base s e um inteiro positivo d. As seqüências ótimas absolutas são construídas a partir de s até o tamanho d.

A subrotina **BuildSeqs** faz a construção das seqüências de forma incremental, isto é, uma seqüência ótima de tamanho $k+1$ é construída a partir de uma seqüência ótima de tamanho k.

A cada posição é obtida uma teoria-pc considerando as regras da teoria-pct válidas para a posição atual. A obtenção da teoria-pc é realizada com a remoção das fórmulas básicas que possuem operadores temporais nas condições das regras. Em seguida, é utilizada a subrotina **BuildBest** apresentada no Capítulo 5 para construir os objetos ótimos para a posição corrente. O pseudo-código para a subrotina **BuildSeqs** é exibido na Figura 9.4.

A complexidade da subrotina **BuildSeqs** é de $O(n^{nd+3d})$, onde $n = |\Phi|$. Isto acontece porque esta subrotina **BuildSeqs** utiliza subrotina **BuildBest** que possui custo de $O(n^3)$. A subrotina **BuildBest** pode retornar até n objetos, sendo que para cada um destes objetos é feita uma chamada recursiva da subrotina **BuildSeqs**. Além disto, este processo se repete d vezes até que as seqüências de tamanho d sejam construídas. Desta maneira, podemos concluir que existe nd chamadas recursivas com custo de n^3, portanto n^{3nd}.

Uma possível execução do algoritmo **BuildSeqs** pode ser vista no Exemplo 9.2.

Exemplo 9.2. Neste exemplo vamos considerar as preferências musicais de um usuário expressas através da teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$, onde:

BuildSeqs(Φ, s, d) :

```
1:  S = ∅                                   // Conjunto de seqüências ótimas, inicialmente vazio
2:  if |s| < d then                         // Verifica se s já possui o tamanho d
3:      Γ_{|s|+1} = Γ_{|s|+1}(Φ, s)         // Obtém a teoria-pc para a posição |s| + 1
4:      for all o_i ∈ BuildBest(Γ_{|s|+1}) do   // Objetos ótimos a partir de Γ_{|s|+1}
5:          s' = add(s, o_i)                // Nova seqüência com as posições de s e o objeto o_i
6:          S = S ∪ BuildSeqs(Φ, s', d) :   // Chamada recursiva com s'
7:      end for
8:  else
9:      return {s}
10: end if
11: return S
```

Figura 9.4: Algoritmo **BuildSeqs**

- $\varphi_1 : (G = p) \rightarrow (D = l) > (D = b)$;
- $\varphi_2 : (G = r) \rightarrow (D = b) > (D = l)$;
- $\varphi_3 : \mathbf{First} \rightarrow (G = p) > (G = r)$;
- $\varphi_4 : \mathbf{Prev}(G = r) \rightarrow (G = p) > (G = r)$;
- $\varphi_5 : \mathbf{Prev}(G = p) \rightarrow (G = r) > (G = p)$.

O atributo gênero é representado por G e o atributo duração é representado por D, enquanto os valores pop, rock, longa, breve são representados por p, r, l e b, respectivamente. Considerando $d = 3$, o algoritmo **BuildSeqs** é executado da seguinte maneira:

- Chamada principal **BuildSeqs**$(\Phi, \langle\rangle, 3)$
- Obtém a teoria-pc para a posição 1, $\Gamma = \{\varphi_1, \varphi_2, \varphi_3'\}$ (as regras φ_1 e φ_2, são consideradas em todas as posições, pois não possuem operadores temporais e a regra φ_3' é a regra φ_3 sem a condição **First**);
- O algoritmo **BuildBest** retorna o objeto (p, l);
- Cria a nova seqüência $s' = \langle(p, l)\rangle$;
- Primeira chamada recursiva **BuildSeqs**$(\Phi, s', 3)$
  - Obtém a teoria-pc para a posição 2, $\Gamma = \{\varphi_1, \varphi_2, \varphi_5'\}$ (a regra φ_5' é a regra φ_5 sem a condição $\mathbf{Prev}(G = p)$);
  - O algoritmo **BuildBest** retorna o objeto (r, b);
  - Cria a nova seqüência $s'' = \langle(p, l), (r, b)\rangle$;
  - Segunda chamada recursiva **BuildSeqs**$(\Phi, s'', 3)$
    - Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi_1, \varphi_2, \varphi_5'\}$ (a regra φ_5' é a regra φ_5 sem a condição $\mathbf{Prev}(G = r)$);
    - O algoritmo **BuildBest** retorna o objeto (p, l);
    - É criada a nova seqüência $s''' = \langle(p, l), (r, b), (p, l)\rangle$;

* Terceira chamada recursiva **BuildSeqs**$(\Phi, s''', 3)$, esta chamada apenas retorna $S = \{s'''\}$, já que $|s'''| = d$;
* Retorna $S = \{\langle (p,l),(r,b),(p,l)\rangle\}$ para a primeira chamada recursiva;

– Retorna $S = \{\langle (p,l),(r,b),(p,l)\rangle\}$ para a chamada principal;

- O algoritmo retorna $S = \{\langle (p,l),(r,b),(p,l)\rangle\}$.

9.1.2 Algoritmo GetBestSeqs-N

O algoritmo **GetBestSeqs-N** foi desenvolvido para resolver o operador **BestSeqs-N**. Este algoritmo recebe como entrada uma relação r e uma teoria-pct Φ consistente. O algoritmo retorna as seqüências ótimas absolutas segundo Φ presentes em r ou todas as seqüências de r, como é apresentado no esquema do algoritmo exibido na Figura 9.5.

Figura 9.5: Esquema do algoritmo **GetBestSeqs-N**

O algoritmo **GetBestSeqs-N** utiliza o algoritmo **GetBestSeqs-E** para obter as seqüências ótimas absolutas presentes em r. Se r não possuir nenhuma das seqüências ótimas absolutas, então todas as seqüências de r são retornadas, como podemos ver através do pseudo-código para o algoritmo exibido na Figura 9.6.

GetBestSeqs-N(r, Φ) :

```
1: S = GetBestSeqs-E(r, Φ)          // Obtém as seqüências ótimas absolutas de r
2: if S = ∅ then                    // Verifica se r possui ao menos uma seqüência ótima absoluta
3:     S = r                        // Retorna todas as seqüências de r
4: end if
5: return S
```

Figura 9.6: Algoritmo **GetBestSeqs-N**

Como o algoritmo **GetBestSeqs-N** utiliza o algoritmo **GetBestSeqs-E** como subrotina, a complexidade do algoritmo é a mesma do algoritmo **GetBestSeqs-E**, ou seja, é de $O(kn^{3nk} + km)$.

9.1.3 Algoritmo GetBestSeqs-D

O algoritmo **GetBestSeqs-D** foi desenvolvido para resolver o operador **BestSeqs-D**. Este algoritmo recebe como entrada uma relação r, uma teoria-pct Φ consistente e retorna

Figura 9.7: Esquema do algoritmo **GetBestSeqs-D**

as seqüências dominantes de r de acordo com Φ, como é apresentado no esquema exibido na Figura 9.7.

O algoritmo **GetBestSeqs-D** baseia-se no algoritmo **GetBest-D** apresentado no Capítulo 5, entretanto no algoritmo **GetBest-D** são realizadas comparações entre tuplas e no algoritmo **GetBestSeqs-D** são realizadas comparações entre seqüências. A comparação entre seqüências é feita através da subrotina **CompareSeqs**, como é mostrado no pseudo-código exibido na Figura 9.8.

GetBestSeqs-D(r, Φ) :

```
1:  TMP = ∅
2:  for all s ∈ r do                                  // Laço para cada seqüência da relação
3:      if TMP = ∅ then
4:          TMP = {s}
5:      else
6:          ignore = false
7:          for all s' ∈ TMP do                       // Laço para cada seqüência de TMP
8:              if ¬ignore then
9:                  s'' = CompareSeqs(r, Φ, s, s')    // Compara as seqüências s e s'
10:                 if s'' = s' then                  // Verifica se s' domina s
11:                     ignore = true
12:                 else if s'' = s then              // Verifica se s domina s'
13:                     TMP = TMP − {s'}              // Remove a seqüência s' da solução
14:                 end if
15:             end if
16:         end for
17:         if ¬ignore then
18:             TMP = TMP ∪ {s}                       // Acrescenta a seqüência s a solução
19:         end if
20:     end if
21: end for
22: return TMP
```

Figura 9.8: Algoritmo **GetBestSeqs-D**

A complexidade do algoritmo **GetBestSeqs-D** é de $O(knm^4 + km9)$, onde k é o tamanho da maior seqüência de r, $n = |\Phi|$ e $m = |r|$. Isto acontece porque a complexidade da subrotina **CompareSeqs** é de $O(knm^2 + km^7)$ e esta subrotina é invocada m^2 vezes.

No Exemplo 9.3 podemos ver uma execução do algoritmo **GetBestSeqs-D**.

Exemplo 9.3. Neste exemplo vamos considerar a relação m_7 sobre o esquema $M_7(D, G)$ exibida na Figura 9.9, onde D e G são os atributos duração e gênero, respectivamente.

Sendo que $\mathbf{Dom}(D) = \{b, l\}$ e $\mathbf{Dom}(G) = \{c, p, r\}$, onde b, l, p, r e c representam os valores breve, longa, pop, rock e clássico, respectivamente. Vamos considerar também a teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4\}$, onde:

- $\varphi_1 : \mathbf{First} \rightarrow (G = r) > (G = c)$;
- $\varphi_2 : \mathbf{First} \rightarrow (G = p) > (G = c)$;
- $\varphi_3 : \mathbf{Prev}(G = r) \rightarrow (G = p) > (G = r)$;
- $\varphi_4 : \mathbf{Prev}(G = p) \rightarrow (G = r) > (G = p)$.

Deste modo, o algoritmo **GetBestSeqs-D** recebendo como entrada m_7 e Φ é executado da seguinte maneira:

- Para a primeira seqüência de m_7 $TMP = \emptyset$, portanto s_1 é adicionada a TMP, $TMP = \{s_1\}$
- A seqüência s_2 é adicionada a TMP pois s_2 é incomparável a s_1, $TMP = \{s_1, s_2\}$
- A seqüência s_3 domina s_1 e s_1 é removida de TMP;
- A seqüência s_3 domina s_2 e s_2 é removida de TMP;
- A seqüência s_3 é adicionada a TMP, $TMP = \{s_3\}$
- A seqüência s_4 é adicionada a TMP pois s_4 é incomparável a s_3, $TMP = \{s_3, s_4\}$
- A seqüência s_5 é dominada por s_3 e portanto é ignorada;
- A seqüência s_6 é dominada por s_3 e portanto é ignorada;
- A seqüência s_7 é adicionada a TMP pois s_7 é incomparável a s_3 e a s_4, $TMP = \{s_3, s_4, s_7\}$;
- A seqüência s_8 domina s_4 e s_4 é removida de TMP;
- A seqüência s_8 é adicionada a TMP, $TMP = \{s_3, s_7, s_8\}$;
- O algoritmo retorna as seqüências s_3, s_7 e s_8.

$$\left\{\begin{array}{l} s_1 = \langle (l,c) \rangle, \\ s_2 = \langle (b,c),(b,r),(l,p) \rangle, \\ s_3 = \langle (l,r),(l,c) \rangle, \\ s_4 = \langle (b,p),(l,p),(l,p) \rangle, \\ s_5 = \langle (l,c),(b,r) \rangle, \\ s_6 = \langle (b,c) \rangle, \\ s_7 = \langle (l,p),(b,p) \rangle, \\ s_8 = \langle (b,p),(l,r),(b,r) \rangle \end{array}\right\}$$

Figura 9.9: Relação m_7 sobre o esquema $M_7(N, C, G)$

Subrotina CompareSeqs

A subrotina **CompareSeqs** compara duas seqüências segundo uma teoria-pct utilizando as propriedades do formalismo lógico *TPref* e o conceito de listas de escopos proposto no Capítulo 3, como foi demonstrado no Capítulo 6.

O algoritmo **CompareSeqs** recebe uma relação r, uma teoria-pct Φ consistente, duas seqüências s e s' e retorna a seqüência dominante (s ou s') ou uma seqüência vazia ($\langle\rangle$) caso s e s' sejam incomparáveis.

O pseudo-código para a subrotina **CompareSeqs** é exibido na Figura 9.10.

CompareSeqs(r, Φ, s, s') :

```
1:  min = smaller(|s|, |s'|)                    // Obtém o tamanho da menor seqüência
2:  for j = 1 to min do
3:      o = pos(s, j)                            // Obtém o objeto da posição j de s
4:      o' = pos(s', j)
5:      if o ≠ o' then                           // Verifica se os objetos o e o' são diferentes
6:          Γ_j = currentPreferences(Φ, s, j)    // Obtém as preferências para a posição j de s
7:          L = GetScopeLists(r, Γ_j)
8:          l = interval(o, L)                   // Obtém os escopos associados a o
9:          l' = interval(o', L)
10:         if subsume(l, l') then               // Verifica se o domina o'
11:             return s
12:         else if subsume(l', l) then
13:             return s'
14:         else
15:             return ⟨⟩
16:         end if
17:     end if
18: end for
19: return ⟨⟩
```

Figura 9.10: Algoritmo **CompareSeqs**

A subrotina **CompareSeqs** é tem complexidade de $O(knm^2 + km^7)$, isto porque a subrotina **GetScopeLists** com complexidade de $O(nm^2 + m^7)$ é invocada k vezes.

O Exemplo 9.4 exibe uma execução para a subrotina **CompareSeqs**.

Exemplo 9.4. Neste exemplo vamos considerar a relação m sobre o esquema $M(G, D)$, onde G é o atributo Gênero e D é o atributo Duração. Sendo que $\mathbf{Dom}(D) = \{b, l\}$ e $\mathbf{Dom}(G) = \{c, p, r\}$, onde b, l, p, r e c representam os valores breve, longa, pop, rock e clássico, respectivamente. Vamos considerar a teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$, onde :

- $\varphi_1 : \mathbf{First} \rightarrow (G = s) > (G = r)$;
- $\varphi_2 : \mathbf{First} \rightarrow (G = r) > (G = p)$;
- $\varphi_3 : \mathbf{Prev}(G = s) \rightarrow (G = r) > (G = s)$;

- $\varphi_4 : \mathbf{Prev}(G = r) \rightarrow (G = s) > (G = r)$;
- $\varphi_5 : (G = r) \rightarrow (D = b) > (D = l)$;
- $\varphi_6 : (G = s) \rightarrow (D = l) > (D = b)$.

Vamos considerar também as seguintes seqüências armazenadas na relação m:

- $s_1 = \langle (s, m), (r, b), (p, l), (s, m) \rangle$;
- $s_2 = \langle (s, m), (r, b), (r, b) \rangle$;

A execução **CompareSeqs**(m, Φ, s_1, s_2) é feita da seguinte maneira:

- A seqüência s_2 possui o menor tamanho, $m = 3$;
- As seqüências s_1 e s_2 diferem na terceira posição;
- É obtida a teoria-pc referente às preferências desta posição

$$\Gamma_3 = \Gamma_3(\Phi, s_1) = \Gamma_3(\Phi, s_2) = \left\{ \begin{array}{l} \varphi'_4 : (G = s) > (G = r), \\ \varphi'_5 : (G = r) \rightarrow (D = b) > (D = l), \\ \varphi'_6 : (G = s) \rightarrow (D = l) > (D = b) \end{array} \right\};$$

- O objeto (p, l) da terceira posição de s_1 possui a lista de escopos $[9, 9]$
- O objeto (r, b) da terceira posição de s_2 possui a lista de escopos $[1, 1]$, **[3,4]**
- Os objetos (p, l) e (r, b) são incomparáveis;
- O algoritmo retorna $\langle\rangle$.

9.2 Algoritmos para Construção de Seqüências Ótimas

Nas subseções a seguir serão apresentados os algoritmos **BuildSeqs-E**, **BuildSeqs-N** e **BuildSeqs-D** responsáveis por resolver os operadores de construção de seqüências ótimas **BuildBestSeqs-E**, **BuildBestSeqs-N** e **BuildBestSeqs-D**, respectivamente.

9.2.1 Algoritmo BuildSeqs-E

O algoritmo **BuildSeqs-E** foi desenvolvido para resolver o operador **BuildBestSeqs-E**. Este algoritmo recebe uma relação r, uma teoria-pct Φ consistente, uma seqüência base s, um tamanho desejado d e um número máximo de seqüências a serem construídas n. Se $n = 0$ então o algoritmo constrói todas as seqüências ótimas possíveis. Um esquema para a entrada e saída do algoritmo **BuildSeqs-E** é mostrado na Figura 9.11.

O algoritmo **BuildSeqs-E** é semelhante a subrotina **BuildSeqs**, porém o algoritmo **BuildSeqs-E** utiliza o algoritmo **GetBest-E** para a construção de cada posição das seqüências a serem construídas, como pode ser visto em seu pseudo-código exibido na Figura 9.12.

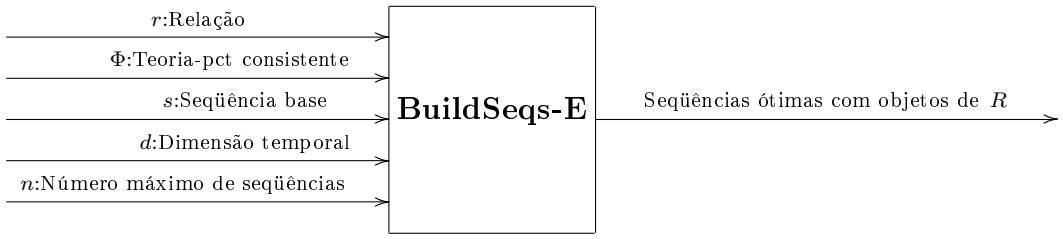


Figura 9.11: Esquema do algoritmo **BuildSeqs-E**

BuildSeqs-E(r, Φ, s, d, n) :

1: $S = \emptyset$
2: **if** $|s| < d$ **then**
3: $\quad \Gamma =$ **currentPreferences**(Φ, seq)
4: $\quad$ **for all** $o_i \in$ **GetBest-E**(r, Γ) **do** // Objetos em r que atendem Γ
5: $\quad\quad s' = add(s, o_i)$
6: $\quad\quad S = S \cup$ **BuildSeqs-E**(Φ, d, n, r, s') :
7: $\quad\quad$ **if** $n > 0$ **and** $|S| \geq n$ **then** // Verifica se o número máximo de seqüências a ser retornado foi atingido
8: $\quad\quad\quad$ **return** $Subset(S, n)$ // Retorna as n primeiras seqüências de S
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end if**
12: **return** S

Figura 9.12: Algoritmo **BuildSeqs-E**

O algoritmo **BuildSeqs-E** possui complexidade de $O((n^3+nm)^{nd})$. Tal complexidade se deve a utilização do algoritmo **GetBest-E** que é feita nd vezes.

Uma execução do algoritmo **BuildSeqs-E** pode ser acompanhada no Exemplo 9.5.

Exemplo 9.5. Neste exemplo vamos a relação m_8 sobre o esquema $M_8(G, D)$ exibida na Figura 9.13, onde G é o atributo Gênero e D é o atributo Duração. Sendo que $\mathbf{Dom}(D) = \{b, l\}$ e $\mathbf{Dom}(G) = \{s, r\}$, onde b, l, s e r representam os valores breve, longa, sertanejo e rock, respectivamente. Vamos considerar também considerar as preferências musicais de um usuário expressas através da teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$, onde:

- $\varphi_1 : \mathbf{First} \rightarrow (G = s) > (G = r)$;
- $\varphi_2 : \mathbf{Prev}(G = r) \rightarrow (G = s) > (G = r)$;
- $\varphi_3 : \mathbf{Prev}(G = s) \rightarrow (G = r) > (G = s)$.

A execução **BuildSeqs-E**$(m_8, \Phi, \langle\rangle, 3, 4)$ é feita de acordo com os seguintes passos:

- Obtém a teoria-pc para a posição 1, $\Gamma = \{\varphi_1'\}$ (onde φ_1' é a regra φ_1 sem a condição **First**);
- O algoritmo **GetBest-E** retorna as tuplas t_1, t_5 e t_6;

- Considerando a tupla t_1, $s_1 = \langle t_1 \rangle$;
- Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_1, 3, 4)$;
  - Obtém a teoria-pc para a posição 2, $\Gamma = \{\varphi'_3\}$ (a regra φ'_3 é a regra φ_3 sem a condição **Prev**$(G = s)$);
  - O algoritmo **GetBest-E** retorna as tuplas t_2 e t_4;
  - Considerando a tupla t_2, $s_{1,1} = \langle t_1, t_2 \rangle$;
  - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,1}, 3, 4)$;
    - Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi'_2\}$ (a regra φ'_2 é a regra φ_2 sem a condição **Prev**$(G = r)$);
    - O algoritmo **GetBest-E** retorna as tuplas t_1, t_5 e t_6;
    - Considerando a tupla t_1, $s_{1,1,1} = \langle t_1, t_2, t_1 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,1,1}, 3, 4)$;
    - $S = \{s_{1,1,1}\}$
    - Considerando a tupla t_5, $s_{1,1,2} = \langle t_1, t_2, t_5 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,1,2}, 3, 4)$;
    - $S = \{s_{1,1,1}, s_{1,1,2}\}$
    - Considerando a tupla t_6, $s_{1,1,3} = \langle t_1, t_2, t_6 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,1,3}, 3, 4)$;
    - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$
    - A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$;
  - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$
  - Considerando a tupla t_4, $s_{1,2} = \langle t_1, t_4 \rangle$;
  - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,2}, 3, 4)$;
    - Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi'_2\}$;
    - O algoritmo **GetBest-E** retorna as tuplas t_1, t_5 e t_6;
    - Considerando a tupla t_1, $s_{1,2,1} = \langle t_1, t_4, t_1 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,2,1}, 3, 4)$;
    - $S = \{s_{1,2,1}\}$
    - Considerando a tupla t_5, $s_{1,2,2} = \langle t_1, t_4, t_5 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,2,2}, 3, 4)$;
    - $S = \{s_{1,2,1}, s_{1,2,2}\}$
    - Considerando a tupla t_6, $s_{1,2,3} = \langle t_1, t_4, t_6 \rangle$;
    - Chamada recursiva **BuildSeqs-E**$(m_8, \Phi, s_{1,2,3}, 3, 4)$;
    - $S = \{s_{1,2,1}, s_{1,2,2}, s_{1,2,3}\}$;
    - A chamada recursiva retorna $\{s_{1,2,1}, s_{1,2,2}, s_{1,2,3}\}$;
  - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}, s_{1,2,2}, s_{1,2,3}\}$;
  - A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}\}$;
- $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}, s_{1,2,2}, s_{1,2,3}\}$;

- $|S| \geq 4$;
- O algoritmo retorna as seqüências $\left\{ \begin{array}{l} \langle (s,l),(r,b),(s,l) \rangle, \\ \langle (s,l),(r,b),(s,m) \rangle, \\ \langle (s,l),(r,b),(s,b) \rangle, \\ \langle (s,l),(r,l),(s,l) \rangle \end{array} \right\}$.

	G	D
t_1	s	l
t_2	r	b
t_3	p	b
t_4	r	l
t_5	s	m
t_6	s	b

Figura 9.13: Relação m_8 sobre o esquema $M_8(G, D)$

9.2.2 Algoritmo BuildSeqs-N

O algoritmo **BuildSeqs-N** foi desenvolvido para resolver o operador **BuildBestSeqs-N**. Este algoritmo **BuildSeqs-N** recebe uma relação r, uma teoria-pct Φ consistente, uma seqüência base s, um tamanho desejado d e um número máximo de seqüências a serem construídas n. Se $n = 0$ então o algoritmo constrói todas as seqüências possíveis. O esquema para a entrada e saída do algoritmo **BuildSeqs-N** é mostrado na Figura 9.14.

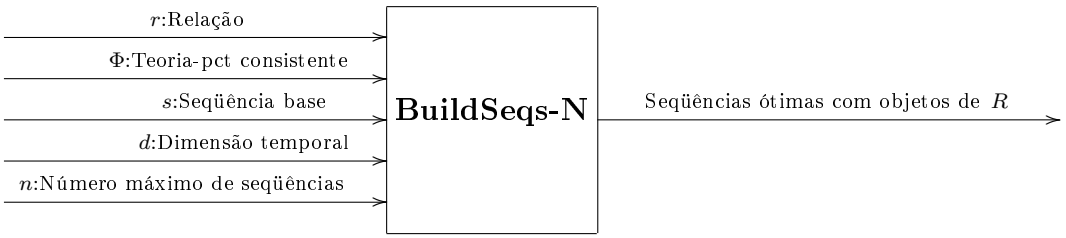


Figura 9.14: Esquema do algoritmo **BuildSeqs-N**

O algoritmo **BuildSeqs-N** é semelhante ao algoritmo **BuildSeqs-E**, porém o algoritmo **BuildSeqs-N** utiliza o algoritmo **GetBest-N** para a construção de cada posição das seqüências a serem construídas, como pode ser visto em seu pseudo-código exibido na Figura 9.15.

Assim como acontece com o algoritmo **BuildSeqs-E**, o algoritmo **BuildSeqs-N** possui complexidade de $O((n^3 + nm)^{nd})$. Tal complexidade se deve a utilização do algoritmo **GetBest-N** que é feita nd vezes.

Uma execução do algoritmo **BuildSeqs-N** pode ser acompanhada no Exemplo 9.6.

Exemplo 9.6. Neste exemplo vamos a relação m_9 sobre o esquema $M_9(G, D)$ exibida na Figura 9.16, onde G é o atributo Gênero e D é o atributo Duração. Sendo que $\mathbf{Dom}(D) = \{b, l\}$ e $\mathbf{Dom}(G) = \{s, r\}$, onde b, l, s e r representam os valores breve, longa, sertanejo e rock, respectivamente. Vamos considerar também considerar as preferências musicais de um usuário expressas através da teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$, onde:

BuildSeqs-N(r, Φ, s, d, n) :

```
1: S = ∅
2: if |s| < d then
3:     Γ = currentPreferences(Φ, seq)
4:     for all o_i ∈ GetBest-N(r, Γ) do                    // Objetos em r que atendem Γ
5:         s' = add(s, o_i)
6:         S = S ∪ BuildSeqs-N(Φ, d, n, r, s') :
7:         if n > 0 and |S| ≥ n then    // Verifica se o número máximo de seqüências a ser retornado foi atingido
8:             return Subset(S, n)                          // Retorna as n primeiras seqüências de S
9:         end if
10:    end for
11: end if
12: return S
```

Figura 9.15: Algoritmo **BuildSeqs-N**

	G	D
t_1	s	l
t_2	r	l
t_3	r	m

Figura 9.16: Relação m_9 sobre o esquema $M_9(G, D)$

- φ_1 : **First** $\rightarrow (G = s) > (G = r)$;
- φ_2 : **Prev**$(G = r) \rightarrow (G = s) > (G = r)$;
- φ_3 : **Prev**$(G = s) \rightarrow (G = r) > (G = s)$;
- $\varphi_4 : (G = p) \rightarrow (D = l) > (D = b)$;
- $\varphi_5 : (G = r) \rightarrow (D = b) > (D = l)$.

A execução **BuildSeqs-N**$(m_9, \Phi, \langle\rangle, 3, 0)$ é feita de acordo com os seguintes passos:

- Obtém a teoria-pc para a posição 1, $\Gamma = \{\varphi'_1, \varphi_4, \varphi_5\}$ (as regras φ_4 e φ_5, são consideradas em todas as posições, pois não possuem operadores temporais e a regra φ'_1 é a regra φ_1 sem a condição **First**);
- O algoritmo **GetBest-N** retorna a tupla t_1;
- Considerando a tupla t_1, $s_1 = \langle t_1 \rangle$;
- Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s_1, 3, 0)$
  - Obtém a teoria-pc para a posição 2, $\Gamma = \{\varphi'_3, \varphi_4, \varphi_5\}$ (a regra φ'_3 é a regra φ_3 sem a condição **Prev**$(G = s)$);
  - O algoritmo **GetBest-N** retorna as tuplas t_1, t_2 e t_3;
  - Considerando a tupla t_1, $s_{1,1} = \langle t_1, t_1 \rangle$;
  - Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 1, 3, 0)$;
    - Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi'_2, \varphi_4, \varphi_5\}$ (a regra φ'_2 é a regra φ_2 sem a condição **Prev**$(G = r)$);

        * O algoritmo **GetBest-N** retorna as tuplas t_1, t_2 e t_3;
        * Considerando a tupla t_1, $s_{1,1,1} = \langle t_1, t_1, t_1 \rangle$;
        * Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 1, 1, 3, 0)$;
        * $S = \{s_{1,1,1}\}$
        * Considerando a tupla t_2, $s_{1,1,2} = \langle t_1, t_1, t_2 \rangle$;
        * Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 1, 2, 3, 0)$;
        * $S = \{s_{1,1,1}, s_{1,1,2}\}$
        * Considerando a tupla t_3, $s_{1,1,3} = \langle t_1, t_1, t_3 \rangle$;
        * Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 1, 3, 3, 0)$;
        * $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$
        * A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$;
    - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}\}$;
    - Considerando a tupla t_2, $s_{1,2} = \langle t_1, t_2 \rangle$;
    - Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 2, 3, 0)$;
        * Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi_2', \varphi_4, \varphi_5\}$;
        * O algoritmo **GetBest-N** retorna a tupla t_1;
        * Considerando a tupla t_1, $s_{1,2,1} = \langle t_1, t_2, t_1 \rangle$;
        * Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 2, 1, 3, 0)$;
        * $S = \{s_{1,2,1}\}$;
        * A chamada recursiva retorna $\{s_{1,2,1}\}$;
    - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}\}$;
    - Considerando a tupla t_3, $s_{1,3} = \langle t_1, t_3 \rangle$;
    - Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 3, 3, 0)$;
        * Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi_2', \varphi_4, \varphi_5\}$;
        * O algoritmo **GetBest-N** retorna a tupla t_1;
        * Considerando a tupla t_1, $s_{1,3,1} = \langle t_1, t_3, t_1 \rangle$;
        * Chamada recursiva **BuildSeqs-N**$(m_9, \Phi, s1, 3, 1, 3, 0)$;
        * $S = \{s_{1,3,1}\}$;
        * A chamada recursiva retorna $\{s_{1,3,1}\}$;
    - $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}, s_{1,3,1}\}$;
    - A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}, s_{1,3,1}\}$;
- $S = \{s_{1,1,1}, s_{1,1,2}, s_{1,1,3}, s_{1,2,1}, s_{1,3,1}\}$;
- O algoritmo retorna as seqüências $\left\{ \begin{array}{l} \langle (s,l),(s,l),(s,l) \rangle, \\ \langle (s,l),(s,l),(r,l) \rangle, \\ \langle (s,l),(s,l),(r,m) \rangle, \\ \langle (s,l),(r,l),(s,l) \rangle, \\ \langle (s,l),(r,m),(s,l) \rangle \end{array} \right\}$.

9.2.3 Algoritmo BuildSeqs-D

O algoritmo **BuildSeqs-D** foi desenvolvido para resolver o operador **BuildBestSeqs-D**. Este algoritmo recebe uma relação r, uma teoria-pct Φ consistente, uma seqüência base s, um tamanho desejado d e um número máximo de seqüências a serem construídas n. Se $n = 0$ então o algoritmo constrói todas as seqüências possíveis. Um esquema para a entrada e saída do algoritmo **BuildSeqs-D** é mostrado na Figura 9.17.

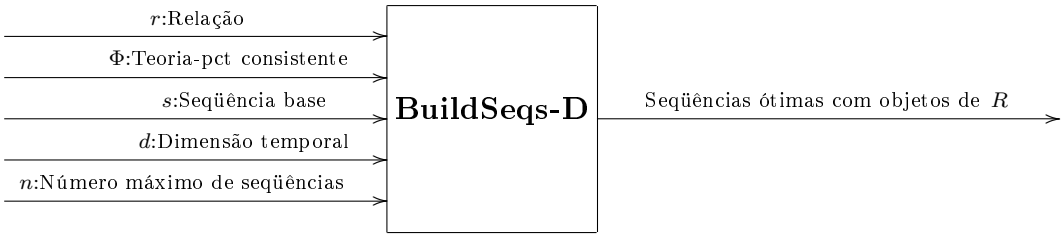


Figura 9.17: Esquema do algoritmo **BuildSeqs-D**

O algoritmo **BuildSeqs-N** é semelhante aos algoritmos **BuildSeqs-E** e **BuildSeqs-N**, porém o algoritmo **BuildSeqs-N** utiliza o algoritmo **GetBest-N** para a construção de cada posição das seqüências a serem construídas, como pode ser visto em seu pseudo-código exibido na Figura 9.15.

BuildSeqs-D(r, Φ, s, d, n) :

```
1:  S = ∅
2:  if |s| < d then
3:      Γ = currentPreferences(Φ, seq)
4:      for all o_i ∈ GetBest-D(r, Γ) do                    // Objetos em r que atendem Γ
5:          s' = add(s, o_i)
6:          S = S ∪ BuildSeqs-D(Φ, d, n, r, s') :
7:          if n > 0 and |S| ≥ n then     // Verifica se o número máximo de seqüências a ser
    retornado foi atingido
8:              return Subset(S, n)                  // Retorna as n primeiras seqüências de S
9:          end if
10:     end for
11: end if
12: return S
```

Figura 9.18: Algoritmo **BuildSeqs-D**

O algoritmo **BuildBest-D** possui complexidade de $O((nm^2 + m^7)^{nd})$. Isto porque o algoritmo **GetBest-D** é utilizado como uma subrotina, tal subrotina é invocada nd vezes e possui complexidade de $O(nm^2 + m^7)$.

Uma execução do algoritmo **BuildSeqs-D** pode ser acompanhada no Exemplo 9.7.

Exemplo 9.7. Neste exemplo vamos a relação m_{10} sobre o esquema $M_{10}(G, D)$ exibida na Figura 9.19, onde G é o atributo Gênero e D é o atributo Duração. Sendo que

$\mathbf{Dom}(D) = \{b, l\}$ e $\mathbf{Dom}(G) = \{s, r\}$, onde b, l, s e r representam os valores breve, longa, sertanejo e rock, respectivamente. Vamos considerar também considerar as preferências musicais de um usuário expressas através da teoria-pct $\Phi = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5\}$, onde:

- $\varphi_1 : \mathbf{First} \rightarrow (G = s) > (G = r)$;
- $\varphi_2 : \mathbf{Prev}(G = r) \rightarrow (G = s) > (G = r)$;
- $\varphi_3 : \mathbf{Prev}(G = s) \rightarrow (G = r) > (G = s)$;
- $\varphi_4 : (G = p) \rightarrow (D = l) > (D = b)$;
- $\varphi_5 : (G = r) \rightarrow (D = b) > (D = l)$.

A execução **BuildSeqs-D**$(m_{10}, \Phi, \langle\rangle, 3, 0)$ é feita de acordo com os seguintes passos:

- Obtém a teoria-pc para a posição 1, $\Gamma = \{\varphi'_1, \varphi_4, \varphi_5\}$ (as regras φ_4 e φ_5, são consideradas em todas as posições, pois não possuem operadores temporais e a regra φ'_1 é a regra φ_1 sem a condição **First**);
- O algoritmo **GetBest-D** retorna as tuplas t_1 e t_2;
- Considerando a tupla t_1, $s_1 = \langle t_1 \rangle$;
- Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_1, 3, 0)$;
  - Obtém a teoria-pc para a posição 2, $\Gamma = \{\varphi'_3, \varphi_4, \varphi_5\}$ (a regra φ'_3 é a regra φ_3 sem a condição $\mathbf{Prev}(G = s)$);
  - O algoritmo **GetBest-D** retorna a tupla t_3;
  - Considerando a tupla t_3, $s_{1,1} = \langle t_1, t_3 \rangle$;
  - Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{1,1}, 3, 0)$;
    - Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi'_2, \varphi_4, \varphi_5\}$;
    - O algoritmo **GetBest-D** retornas as tuplas t_1 e t_2;
    - Considerando a tupla t_1, $s_{1,1,1} = \langle t_1, t_3, t_1 \rangle$;
    - Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{1,1,1}, 3, 0)$;
    - $S = \{s_{1,1,1}\}$;
    - Considerando a tupla t_2, $s_{1,1,2} = \langle t_1, t_3, t_2 \rangle$;
    - Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{1,1,2}, 3, 0)$;
    - $S = \{s_{1,1,1}, s_{1,1,2}\}$;
    - A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}\}$;
  - $S = \{s_{1,1,1}, s_{1,1,2}\}$;
  - A chamada recursiva retorna $\{s_{1,1,1}, s_{1,1,2}\}$;
- $S = \{s_{1,1,1}, s_{1,1,2}\}$;
- Considerando a tupla t_2, $s_2 = \langle t_2 \rangle$;
- Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_2, 3, 0)$;
  - Obtém a teoria-pc para a posição 2, $\Gamma = \{\varphi'_3, \varphi_4, \varphi_5\}$;

  - O algoritmo **GetBest-D** retorna a tupla t_3;
  - Considerando a tupla t_3, $s_{2,1} = \langle t_2, t_3 \rangle$;
  - Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{2,1}, 3, 0)$;
    * Obtém a teoria-pc para a posição 3, $\Gamma = \{\varphi'_2, \varphi_4, \varphi_5\}$;
    * O algoritmo **GetBest-D** retornas as tuplas t_1 e t_2;
    * Considerando a tupla t_1, $s_{2,1,1} = \langle t_2, t_3, t_1 \rangle$;
    * Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{2,1,1}, 3, 0)$;
    * $S = \{s_{2,1,1}\}$;
    * Considerando a tupla t_2, $s_{2,1,2} = \langle t_2, t_3, t_2 \rangle$;
    * Chamada recursiva **BuildSeqs-D**$(m_{10}, \Phi, s_{2,1,2}, 3, 0)$;
    * $S = \{s_{2,1,1}, s_{2,1,2}\}$;
    * A chamada recursiva retorna $\{s_{2,1,1}, s_{2,1,2}\}$;
  - $S = \{s_{2,1,1}, s_{2,1,2}\}$;
  - A chamada recursiva retorna $\{s_{2,1,1}, s_{2,1,2}\}$;
- $S = \{s_{1,1,1}, s_{1,1,2}, s_{2,1,1}, s_{2,1,2}\}$;
- O algoritmo retorna as seqüências $\left\{ \begin{array}{l} \langle (s,l),(r,l),(s,l) \rangle, \\ \langle (s,l),(r,l),(s,m) \rangle, \\ \langle (s,m),(r,l),(s,l) \rangle, \\ \langle (s,m),(r,l),(s,m) \rangle \end{array} \right\}$.

	G	D
t_1	s	l
t_2	s	m
t_3	r	l
t_4	s	b

Figura 9.19: Relação m_{10} sobre o esquema $M_{10}(G, D)$

9.3 Implementação do Protótipo

No decorrer deste trabalho foi implementado um protótipo para um fragmento da linguagem *TPref-SQL*, este protótipo apresenta um funcionamento análogo ao protótipo da linguagem *CPref-SQL* apresentado no Capítulo 5.

O protótipo para a linguagem *TPref-SQL* foi implementado de forma integrada ao SGBD PostgreSQL e também utiliza a biblioteca dinâmica denominada **preflib**.

Desenvolvemos um interpretador que efetua uma conversão sobre uma consulta *TPref-SQL* fazendo com que o SGBD utilize a biblioteca **preflib** para resolver operações envolvendo preferências. O resultado destas operações é devolvido ao SGBD que, por sua vez, retorna o resultado da consulta ao interpretador. Este processo de execução de consultas *TPref-SQL* é exibido na Figura 9.20.

Atualmente os algoritmos **BestSeqs-E**, **BestSeqs-N**, **BuildSeqs-E** e **BuildSeqs-N** encontram-se implementados dentro da biblioteca **preflib**.

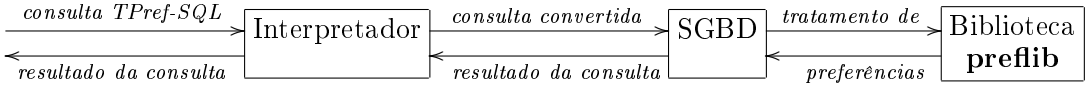


Figura 9.20: Processo de execução de consultas *TPref-SQL*

Alguns operadores da álgebra *TPrefAR* podem ser implementados dentro do interpretador através de uma conversão na consulta *TPref-SQL* para uma consulta SQL padrão. Como exemplo, vamos considerar o operador de seleção.

A conversão de uma consulta *TPref-SQL* contendo restrições na cláusula *WHERE* pode ser feita transformando cada condição básica em uma consulta SQL e transformando os conectivos das condições básicas em operações de interseção (para o conectivo *AND*) ou união (para o conectivo *OR*). Como exemplo, vamos considerar a consulta *TPref-SQL*:

```
SELECT *
FROM R
WHERE (T=1 AND A=a)
AND (T=2 AND B=b)
OR NOT(T=3 AND C=C)
```

que pode ser convertida na consulta SQL:

```
SELECT *
FROM R AS R0
WHERE EXISTS (

     SELECT *
     FROM R AS R2
     WHERE R0.ID = R1.ID
     AND (T=1 AND A=a)

)
INTERSECT
SELECT *
FROM R AS R0
WHERE EXISTS (

     SELECT *
     FROM R AS R2
     WHERE R0.ID = R1.ID
     AND (T=2 AND B=b)

)
UNION
SELECT *
FROM R AS R0
WHERE NOT EXISTS (

     SELECT *
     FROM R AS R2
     WHERE R0.ID = R1.ID
     AND (T=3 AND C=C)

).
```

Ainda é preciso um estudo mais detalhado quanto a conversão de consultas *TPref-SQL* para consultas SQL padrão principalmente no que diz respeito a desempenho. Além disto, alguns operadores podem necessitar de algoritmos específicos pelo fato de trabalharem com seqüências de tuplas.

9.4 Considerações Finais

Neste capítulo foram propostos os algoritmos **GetBestSeqs-E**, **GetBestSeqs-N** e **GetBestSeqs-D** que foram desenvolvidos para resolver os operadores de seleção de seqüências ótimas **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**, respectivamente.

Os algoritmos propostos neste capítulo possuem uma complexidade elevada, portanto há uma grande necessidade de pesquisas no sentido de otimizar tais algoritmos.

Além dos algoritmos especificados, descrevemos como foi implementado um protótipo para um fragmento da linguagem *TPref-SQL*.

Capítulo 10

Conclusão e Trabalhos Futuros

Recentemente, diversas pesquisas têm sido realizadas sobre preferências e sobre como aplicar preferências no contexto de Banco de Dados no sentido de proporcionar respostas cada vez mais próximos ao desejo de um usuário. Nesta dissertação foram especificadas as linguagens de consulta para banco de dados com suporte a preferências *CPref-SQL* e *TPref-SQL*.

A linguagem *CPref-SQL* consiste de uma extensão da linguagem SQL padrão realizada por meio dos operadores de seleção de seqüências ótimas **Best-E**, **Best-N** e **Best-D**. Apesar destes operadores possuírem expressões equivalentes na álgebra relacional padrão, foram desenvolvidos algoritmos específicos para cada operador objetivando um melhor desempenho.

Os algoritmos **GetBest-E** e **GetBest-N** desenvolvidos para resolver os operadores **Best-E** e **Best-N**, respectivamente, verificam a existência das tuplas ótimas absolutas de acordo com as preferências em uma relação e não realizam qualquer comparação entre tuplas. No caso do algoritmo **GetBest-D** responsável pelo operador **Best-D** existe a necessidade de comparar as tuplas de uma relação considerando um conjunto de preferências. Para que estas comparações possam ser realizadas diretamente foi proposto o conceito de listas de escopos.

Desenvolvemos um protótipo utilizando o SGBD PostgreSQL, onde os algoritmos **GetBest-E** e **GetBest-N** encontram-se implementados.

Foi proposto um modelo relacional onde as relações de banco de dados armazenam seqüências de tuplas proporcionando uma noção cronológica para os dados armazenados. Foi proposta a álgebra *TPrefAR* com operadores específicos para trabalhar com este tipo de relação, em especial os operadores de seleção de seqüências ótimas e de construção de seqüências ótimas.

Foram propostos os operadores **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D** para seleção de seqüências ótimas considerando preferências condicionais temporais. Os operadores **BestSeqs-E** e **BestSeqs-N** verificam a existência de seqüências ótimas absolutas (considerando somente as preferências) em uma relação. Já o operador **BestSeqs-D** retorna

as seqüências dominantes de uma relação considerando as preferências.

Os operadores de construção de seqüências ótimas são capazes de construir seqüências a partir de tuplas considerando um conjunto de regras de preferências condicionais temporais.

A partir da álgebra *TPrefAR* foi especificada a linguagem *TPref-SQL* através da qual podemos elaborar consultas para um banco de dados de seqüências.

Foram propostos os algoritmos **GetBestSeqs-E**, **GetBestSeqs-N** e **GetBestSeqs-D** para resolver os operadores de seleção de seqüências ótimas **BestSeqs-E**, **BestSeqs-N** e **BestSeqs-D**, respectivamente. Implementamos um protótipo para linguagem *TPref-SQL* contendo implementações dos algoritmos **BuildSeqs-E** e **BuildSeqs-N**. Tal protótipo foi implementado de forma semelhante ao protótipo para a linguagem *CPref-SQL*.

Com a especificação das linguagens *CPref-SQL* e *TPref-SQL* atingimos o principal objetivo deste trabalho, ou seja, a especificação de linguagens de consulta para banco de dados capazes de considerar preferências condicionais ao resolver consultas. Estas linguagens prestam uma importante contribuição para a área de Banco de Dados permitindo que que consultas contendo preferências possam ser especificadas de uma maneira mais intuitiva. Isto acontece principalmente porque as preferências são especificadas através de regras que apresentam um componente condicional.

Trabalhos Futuros

Com a realização desta dissertação vislumbramos uma grande diversidade de otimizações e extensões que podem proporcionar importantes pesquisas futuras. Estas otimizações e extensões são discutidas a seguir.

Na linguagem de preferências condicionais, assim como no formalismo lógico **TPref**, tanto nas condições das regras quanto nas próprias preferências, consideramos apenas valores discretos, ou seja, temos apenas atribuições do tipo $A = a$. Uma importante melhoria pode ser realizada no sentido de permitir que as regras possuam trabalhar com atributos contínuos, ou seja, desigualdades do tipo $A > a$ ou $A < a$, tanto nas condições quanto nas próprias preferências. Entretanto devem ser realizadas pesquisas para verificar se este tipo de melhoria mantém as propriedades da linguagem de preferências condicionais e do formalismo lógico **TPref**.

No Capítulo 3 foi proposta uma otimização para o algoritmo **GetBest-D** utilizando o conceito de clones de subgrafos BTG. Entretanto, outras otimizações podem ser realizadas sobre este algoritmo visando reduzir o número de nós do grafo BTG. Um tipo de otimização pode utilizar o conceito de dependência preferencial entre os atributos de uma teoria-pc.

Nesta dissertação mostramos que as consultas *CPref-SQL* possuem planos de execução semelhantes a linguagem SQL padrão, mas não abordamos a otimização destes planos

de execução. Pesquisas futuras podem tratar a otimização dos planos de execução de consultas *CPref-SQL* para que o tempo de respostas destas consultas seja mínimo.

A subrotina **BuildBest** utilizada pelos algoritmos **GetBest-E** e **GetBest-N** para tuplas ótimas absolutas não suporta teorias-pc sem regras-pc com condição vazia (deve existir pelo menos uma). Esta rotina pode ser estendida para trabalhar com teorias-pc deste tipo.

Atualmente os algoritmos **GetBest-E** e **GetBest-N** estão implementados através de um protótipo. Para que a linguagem *CPref-SQL* esteja totalmente implementada o algoritmo **GetBest-D** responsável pelo operador **Best-D** deve ser implementado também. A implementação deste algoritmo poderá usar diversas rotinas já implementadas no protótipo como o processo de captura das regras de preferências contidas em uma consulta *CPref-SQL*.

No formalismo lógico *TPref* proposto por [de Amo e Giacometti 2007] a consistência de teorias-pct é garantida apenas para teorias-pct **TPref***. Desta maneira, podem ser desenvolvidos métodos para garantir a consistência de teorias-pct que não estão presentes em **TPref***, ou seja, teorias-pct que possuem regras-pct cujas condições apresentam fórmulas *LTS* de futuro. Com isto, podem ser desenvolvidos novos algoritmos para seleção e construção de seqüências ótimas considerando uma variedade maior de teorias-pct. Além disto podem ser propostos outros operadores para a álgebra *TPrefAR* baseados nos novos algoritmos e, conseqüentemente, a linguagem *TPref-SQL* pode ser estendida com estes operadores.

O operador de seleção da álgebra *TPrefAR* aceita apenas condições básicas do tipo $(\mathcal{P}\ op\ p) \wedge (A_i\ op\ v)$, como foi visto no Capítulo 7. Este operador pode ser modificado para aceitar condições de seleção mais gerais expressas por meio de fórmulas da lógica temporal.

Neste trabalho implementamos os operadores **BuildBestSeqs-E** e **BuildBestSeqs-N** da álgebra *TPrefAR*. Para que a linguagem *TPref-SQL* se torne mais funcional será necessário a implementação dos demais operadores da álgebra *TPrefAR*. Além disto muitos destes operadores podem ser implementados por mais de um método. Desta forma, para determinados operadores existe a necessidade de considerar vários métodos e compará-los para selecionar aquele que for mais adequado. A implementação de todos os operadores da álgebra *TPrefAR* permitirá que um protótipo mais completo seja construído para a linguagem *TPref-SQL*

Referências Bibliográficas

[Abiteboul et al. 1995] Abiteboul, S., Hull, R., e Vianu, V. (1995). *Foundations of Databases: The Logical Level*. Addison-Wesley, Boston, Massachusetts, USA.

[Agrawal et al. 1989] Agrawal, R., Borgida, A., e Jagadish, H. V. (1989). Efficient management of transitive relationships in large data and knowledge bases. In *ACM SIGMOD International Conference on Management of Data*, pp. 253–262, Portland, Oregon, USA.

[Agrawal e Wimmers 2000] Agrawal, R. e Wimmers, E. L. (2000). A Framework for Expressing and Combining Preferences. In *ACM SIGMOD International Conference on Management of Data*, pp. 297–306, Dallas, Texas, USA.

[Andréka et al. 2002] Andréka, H., Ryan, M., e Schobbens, P.-Y. (2002). Operators and Laws for Combining Preference Relations. *Journal of Logic and Computation*, 12(1):13–53.

[Bäckström e Nebel 1995] Bäckström, C. e Nebel, B. (1995). Complexity Results for SAS+ Planning. *Computational Intelligence*, 11(4):625–655.

[Börzsönyi et al. 2001] Börzsönyi, S., Kossmann, D., e Stocker, K. (2001). The Skyline Operator. In *17th International Conference on Data Engineering (ICDE)*, pp. 421–430, Heidelberg, Germany.

[Boutilier et al. 2004a] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., e Poole, D. (2004a). CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. *Journal of Artificial Intelligence Research (JAIR)*, 21:135–191.

[Boutilier et al. 2004b] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., e Poole, D. (2004b). Preference-based Constrained Optimization with CP-nets. *Computational Intelligence*, 20(2):137–157.

[Boutilier et al. 1999] Boutilier, C., Brafman, R. I., Hoos, H. H., e Poole, D. (1999). Reasoning with Conditional Ceteris Paribus Preference Statements. In *15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 71–80, Stockholm, Sweden.

[Brafman e Domshlak 2002] Brafman, R. I. e Domshlak, C. (2002). Introducing Variable Importance Tradeoffs into CP-Nets. In *18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 69–76, Edmonton, Alberta, Canada.

[Brafman et al. 2004] Brafman, R. I., Domshlak, C., e Shimony, S. E. (2004). Qualitative Decision Making in Adaptive Presentation of Structured Information. *ACM Transaction on Information Systems (TOIS)*, 22(4):503–539.

[Brafman et al. 2006a] Brafman, R. I., Domshlak, C., e Shimony, S. E. (2006a). On Graphical Modeling of Preference and Importance. *Journal of Artificial Intelligence Research (JAIR)*, 25:389–424.

[Brafman et al. 2006b] Brafman, R. I., Domshlak, C., Shimony, S. E., e Silver, Y. (2006b). Preferences over Sets. In *21st National Conference on Artificial Intelligence (AAAI)*, Menlo Park, California, USA.

[Brafman e Friedman 2006] Brafman, R. I. e Friedman, D. (2006). Adaptive Rich Media Presentations via Preference-Based Constrained Optimization. In *Preferences: Specification, Inference, Applications*, número 04271 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany.

[Burke 2000] Burke, R. (2000). Knowledge-based recommender systems. In Kent, A. (editor), *Encyclopedia of Library and Information Systems*, volume 69, pp. 180–200. Marcel Dekker, New York, New York, USA.

[Bylander 1994] Bylander, T. (1994). The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2):165–204.

[Castañeda 1958] Castañeda, H. N. (1958). Review of Halldén "On the Logic of 'Better' ". *Philosophy and Phenomenological Research*, 19(2):266.

[Chajewska et al. 1998] Chajewska, U., Getoor, L., Norman, J., e Shahar, Y. (1998). Utility Elicitation as a Classification Problem. In *14th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 79–88, Madison, Wisconsin, USA.

[Chan et al. 2005] Chan, C.-Y., Eng, P.-K., e Tan, K.-L. (2005). Stratified Computation of Skylines with Partially-Ordered Domains. In *ACM SIGMOD International Conference on Management of Data*, pp. 203–214, Baltimore, Maryland, USA.

[Chomicki 2003] Chomicki, J. (2003). Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466.

[de Amo e Giacometti 2007] de Amo, S. e Giacometti, A. (2007). Temporal Conditional Preferences over Sequences of Objects. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 2, pp. 246–253, Pratas, Greece.

[de Amo e Giacometti 2008] de Amo, S. e Giacometti, A. (2008). *Preferences on Objects, Sets and Sequences*. I-Tech Publications, Vienna, Austria.

[de Amo e Ribeiro 2009] de Amo, S. e Ribeiro, M. R. (2009). CPref-SQL: A Query Language Supporting Conditional Preferences. In *24th Annual ACM Symposium on Applied Computing (ACM SAC 2009)*, Honolulu, Hawaii, USA. (A ser publicado).

[des Jardins e Wagstaff 2005] des Jardins, M. e Wagstaff, K. (2005). DD-PREF: A Language for Expressing Preferences over Sets. In *20th National Conference on Artificial Intelligence (AAAI)*, pp. 620–626, Pittsburgh, Pennsylvania, USA.

[Domshlak e Brafman 2002a] Domshlak, C. e Brafman, R. I. (2002a). CP-nets: Reasoning and Consistency Testing. In *8th International Conference on Principles and Knowledge Representation and Reasoning (KRR)*, pp. 121–132, Toulouse, France.

[Domshlak e Brafman 2002b] Domshlak, C. e Brafman, R. I. (2002b). Structure and Complexity in Planning with Unary Operators. In *6th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 34–43, Toulouse, France.

[Domshlak et al. 2001] Domshlak, C., Brafman, R. I., e Shimony, S. E. (2001). Preference-Based Configuration of Web Page Content. In *17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1451–1456, Seattle, Washington, USA.

[Doyle et al. 1991] Doyle, J., Shoham, Y., e Wellman, M. P. (1991). A Logic of Relative Desire (Preliminary Report). In *6th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, volume 542 de *Lecture Notes in Computer Science*, pp. 16–31, Charlotte, North Carolina, USA.

[Doyle e Wellman 1994] Doyle, J. e Wellman, M. P. (1994). Representing Preferences as Ceteris Paribus Comparatives. In *AAAI Spring Symposium on Decision-Theoretic Planning*, pp. 69–75, Stanford, California, USA.

[Endres e Kießling 2006] Endres, M. e Kießling, W. (2006). Transformation of TCP-Net Queries into Preference Database Queries. In *2nd Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*, pp. 23–30, Riva del Garda, Italy.

[Endres e Kießling 2008] Endres, M. e Kießling, W. (2008). Optimization of Preference Queries with Multiple Constraints. In *International Workshop on Personalized Access, Profile Management, and Context Awareness: Databases (PersDB)*, Auckland, New Zealand.

[Fishburn 1999] Fishburn, P. C. (1999). Preference Structures and Their Numerical Representations. *Theoretical Computer Science*, 217(2):359–383.

[Gabbay 1987] Gabbay, D. M. (1987). The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In *Temporal Logic in Specification*, volume 398 de *Lecture Notes in Computer Science*, pp. 409–448, Altrincham, UK.

[Halldén 1957] Halldén, S. (1957). *On the Logic of 'Better'*. CWK Gleerup, Lund, Copenhagen, Sweden.

[Hansson 1996] Hansson, S. O. (1996). What is ceteris paribus preference? *Journal of Philosophical Logic*, 25(3):307–332.

[Hodkinson e Reynolds 2005] Hodkinson, I. M. e Reynolds, M. (2005). Separation - Past, Present, and Future. In *We Will Show Them! (Essays in honour of Dov Gabbay on his 60th birthday)*, volume 2, pp. 117–142, London, UK.

[Hristidis et al. 2001] Hristidis, V., Koudas, N., e Papakonstantinou, Y. (2001). PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *ACM SIGMOD International Conference on Management of Data*, pp. 259–270, Santa Barbara, California, United States.

[Ioannidis e Koutrika 2005] Ioannidis, Y. E. e Koutrika, G. (2005). Personalized Systems: Models and Methods from an IR and DB Perspective. In *31st International Conference International Conference on Very Large Data Bases (VLDB)*, p. 1365, Trondheim, Norway.

[Kießling 2002] Kießling, W. (2002). Foundations of Preferences in Database Systems. In *28th International Conference International Conference on Very Large Data Bases (VLDB)*, pp. 311–322, Hong Kong, China.

[Kießling et al. 2004] Kießling, W., Fischer, S., e Döring, S. (2004). COSIMA B2B–Sales Automation for E-Procurement. In *6th IEEE Conference on E-Commerce Technology (CEC)*, pp. 59–68, San Diego, California, USA.

[Kießling e Köstler 2002] Kießling, W. e Köstler, G. (2002). Preference SQL - Design, Implementation, Experiences. In *28th International Conference International Conference on Very Large Data Bases (VLDB)*, pp. 990–1001, Hong Kong, China.

[Koutrika et al. 2006] Koutrika, G., Simitsis, A., e Ioannidis, Y. E. (2006). Précis: The Essence of a Query Answer. In *22nd International Conference on Data Engineering (ICDE)*, pp. 69–78, Atlanta, Georgia, USA.

[Kron e Milovanović 1975] Kron, A. e Milovanović, V. (1975). Preference and choice. *Theory and Decision*, 6(2):185–196.

[Nguyen e Haddaway 1999] Nguyen, H. e Haddaway, P. (1999). The Decision-Theoretic Interactive Video Advisor. In *15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 494–501, Stockholm, Sweden.

[Öztürk et al. 2006] Öztürk, M., Tsoukiàs, A., e Vincke, P. (2006). Preference Modelling. In *Preferences: Specification, Inference, Applications*, número 04271 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany.

[Preisinger e Kießling 2007] Preisinger, T. e Kießling, W. (2007). The Hexagon Algorithm for Pareto Preference Queries. In *3rd Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*, Vienna, Austria.

[Preisinger et al. 2006] Preisinger, T., Kießling, W., e Endres, M. (2006). The BNL++ Algorithm for Evaluating Pareto Preference Queries. In *2nd Multidisciplinary Workshop on Advances in Preference Handling (MPREF)*, pp. 114–121, Riva del Garda, Italy.

[Prior 1967] Prior, A. N. (1967). *Past, Present and Future.* Oxford University Press, Oxford, New York, USA.

[Resnick e Varian 1997] Resnick, P. e Varian, H. R. (1997). Recommender Systems. *Communications of the ACM*, 40(3):56–58.

[Shimony e Domshlak 2003] Shimony, S. E. e Domshlak, C. (2003). Complexity of probabilistic reasoning in directed-path singly-connected Bayes networks. *Artificial Intelligence*, 151(1-2):213–225.

[Stefanidis e Pitoura 2008] Stefanidis, K. e Pitoura, E. (2008). Fast contextual preference scoring of database tuples. In *11th International Conference on Extending Database Technology (EDBT)*, volume 261 de *ACM International Conference Proceeding Series*, pp. 344–355, Nantes, France.

[Trapp 1985] Trapp, R. W. (1985). Utility theory and preference logic. *Erkenntnis*, 22(1-3):301–339.

[von Wright 1963] von Wright, G. H. (1963). *The Logic of Preference: An Essay*. Edinburgh University Press, Edinburgh, Scotland.

[von Wright 1972] von Wright, G. H. (1972). The logic of preference reconsidered. *Theory and Decision*, 3(2):140–169.

[Wellman e Doyle 1991] Wellman, M. P. e Doyle, J. (1991). Preferential Semantics for Goals. In *9th National Conference on Artificial Intelligence (AAAI)*, pp. 698–703, Anaheim, California, USA.

[Wilson 2004a] Wilson, N. (2004a). Consistency and Constrained Optimisation for Conditional Preferences. In *16th Eureopean Conference on Artificial Intelligence (ECAI)*, pp. 888–894, Valencia, Spain.

[Wilson 2004b] Wilson, N. (2004b). Extending CP-Nets with Stronger Conditional Preference Statements. In *19th National Conference on Artificial Intelligence (AAAI)*, pp. 735–741, San Jose, California, USA.

[Wilson 2006] Wilson, N. (2006). An Efficient Upper Approximation for Conditional Preference. In *17th European Conference on Artificial Intelligence (ECAI)*, pp. 472–476, Riva del Garda, Italy.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo