Programação de TV

Programação de TV utilizando PLR

Mestrado Integrado em Engenharia Informática e Computação
Programação em Lógica

Turma 3MIEIC05, Grupo Programação de TV_4:
Mário Gustavo Gomes Rosas de Azevedo Fernandes - up201201705@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de Dezembro de 2016

Resumo

O trabalho descrito neste artigo consiste em receber um horario com alguns dias da semana e horas (conjunto de slots) e preencher cada slot com uma série fictícia que se enquadrava nas restrições do problema.

Foi utilizado programação em lógica por restrições e como principal resultado foi obtido um programa bastante eficiente para amostras pequenas mas quanto mais series e slots se adicionam mais lento fica.

Dito isto, foi possível concluir após a realização que a programação em lógica por restrições é um método bastante mais eficiente que outro lecionado até hoje quando o objetivo é procurar soluções viáveis para variáveis.

1 Introdução

Como segundo trabalho da cadeira de Programação em Lógica foram propostos vários trabalhos que se dividiam em puzzles e problemas de optimização.

Visto que no primeiro trabalho já tinha sido feito um jogo, achei mais interessante tentar resolver algo diferente de um puzzle, escolhendo assim o problema 11: Programação TV.

O objetivo deste trabalho foi preencher slots de uma semana previamente estabelecidos com séries também previamente estabelecidas e suas restrições.

Comparativamente a trabalhos já executados durante o curso, incluindo o primeiro da cadeira de Programação em Lógica, o tipo de pensamento para resolver o problema é bastante diferente, devido ao método das restrições utilizado. Enquanto que outros métodos de programação para resolver este projeto iriam fazer algo como escolher uma série para uma slot, verificar todas as restrições e no final aceitar ou escolher outra série (aplicando outra vez todas as regras), PLR consegue definir as ditas regras para variáveis não definidas e apenas no final de "memorizar"todas as regras, procura a melhor solução.

Neste artigo será descrito em pormenor toda a estrutura de representação das varáveis, todas os métodos utilizados, os resultados obtidos e ainda estatísticas do tempo de funcionamento do programa.

2 Descrição do Problema

O problema consiste em programar um horário válido para uma estação de televisão com séries previamente definidas. A ideia seria que, a estação de televisão desse uma lista de slots, com duração de 30 minutos cada, ao longo de uma semana, e uma lista de séries com as suas restrições, e, com auxílio do programa feito, iria se obter a solução mais lucrativa que tivesse em conta as audiências e o custo de comprar/emitir a série.

Neste problema existiam 5 pontos importantes a referir:

• Apenas poderia ser emitida determinada série uma vez por semana.

• A duração das série teria de ser respeitada,por exemplo uma série que durasse 1 hora apenas poderia ser introduzida se houvessem duas slots consecutivas.

• Séries poderiam estar restritas de passar no mesmo dia que outra série.

• Séries poderiam estar restritas de passar num determinado horario.

• O programa deveria estar apto para fazer a melhor escolha que minimizasse o custo e maximizasse as audiências.

3 Abordagem

3.1 Variáveis de Decisão

As variáveis de decisão do programa são apenas os títulos a preencher de cada slot e o domínio (em ID's de séries) vai do 1 até ao número máximo de series na base de dados.

Todas as variáveis têm de ser representadas sob a forma correta para que o programa seja válido e estas são:

• As séries (series), seguindo a forma de listas que contêm o id, o título, o custo e a duração.

• As votações (votacao), seguindo a forma de listas que contêm o ID da série a que se refere, o ID da possível slot e a resptiva pontuação.

• As restrições de séries por dia (rest_series), seguindo a forma de listas que contêm dois ID's referentes a duas séries que não podem ser emitidas no mesmo dia.

• As restrições de séries por hora (rest_series_hours), seguindo a forma de listas que contêm o ID da série em causa, o identificador (se for 1 significa que nao pode passar antes de X horas e vice-versa se for 2), e as horas de referência.

• As slots (slots), seguindo a forma de listas que contêm o ID da slot, o dia a que se referenciam, o tempo referente a quando irão comecar (sendo que o tempo está em minutos desde o dia 1, isto é, por exemplo, as 10 horas do dia 2 da semana iriam corresponder a 2040) e por fim, a variável de decisão, o título ainda nao atribuído.

3.2 Restrições

Como já referido na descrição do problema existem alguns pontos principais a tratar. Os relativos a quando podem ser emitidas as séries, são restrições rígidas e as relativas à escolha da melhor opção são flexiveis.

Numa fase inicial é utilizado o predicado domain para estabelecer o domínio das variáveis.

3.2.1 Séries não repetidas

Para resolver esta questão decidi utilizar o predicado all_distinct para que todas as variáveis assumissem números diferentes. A restrição funciona com sucesso se todas as restrições seguintes assumirem que uma slot corresponde a uma série.

predicados: all_distinct

3.2.2 Duração das séries respeitadas

Para resolver este ponto foi tentado utilizar restrições baseadas em tasks e máquinas em que cada task correspondesse a 1 slot e cada máquina uma série.

A ideia aqui seria dizer que cada máquina teria X recursos, que correspodia à duração da série, e que cada task utilizasse 30 minutos de recursos de uma máquina, podendo assim existir, por exemplo, duas tasks que fossem alocadas para uma máquina com recurso igual a 60 minutos.

Os problemas aqui encontrados foram garantir que apenas estivessem numa máquina tasks consecutivas, algo que não foi possível realizar, e ainda restringir que cada série só pudesse passar uma vez por semana. Neste último ponto não poderia ser utilizado o all_distinct pois por exemplo, as séries de 1h estariam representadas em duas slots.

Numa outra abordagem pensei em transformar slots consecutivas numa única slot com uma maior duração, algo que iria dar para colocar, por exemplo, séries de 1h em tasks de 1h, e podendo assim utilizar o all_distinct, mas desta forma não era possível alocar séries de duração menor nestas tasks.

Por fim decidi instituir o sistema de tasks que funciona se apenas existir séries de 30 minutos na base de dados.

predicados: makeLT que transforma as slots em tasks, makeMachines que transforma as series em machines e cumulatives que "ativa"a restrição.

3.2.3 Restrições por dia

Para resolver esta restrição foram feitos predicados que por cada restrição vai percorrer a matriz de tasks e restringir as séries que podem estar diariamente.

A maneira como os predicados funcionam é, se for atribuido uma certa série num dia, e se estiver presente uma restricao diária com outra série, os predicados irão assegurar que essa outra série não consiga passar no dia da série em conta.

Para isto é especialmente importante a utilização da variável Bo e Exists: se for encontrada alguma série dentro de uma restrição, Exists fica 1; se Exists se verificar, então implica que Bo se terá de verificar. Esta última variável Bo é a variável de retorno do predicado restrictDay3 que aplica a restricao propriamente dita.

predicados: restrictDay, restrictDay2, restrictDay3.

3.2.4 Restrições por horas

Para resolver este tópico são utilizados os predicados restrictHours e restrictHours2 que seguem uma política parecida à restrição anterior: percorre as restrições e para cada restrição percorre a lista de tasks (sendo que esta lista não necessita de ser uma matriz por dias), e para cada task obriga a que a variável do titulo corresponda à restrição.

Uma coisa a referir destes predicados é a necessidade de converter o StartTime das tasks para minutos do dia (por exemplo, converter 2040 para 10 horas), tendo este valor de ser independente dos dias.

predicados: restrictHours, restrictHours2

3.3 Função de Avaliação

Para avaliar a melhor opção foi feito um predicado chamado getValor. Este predicado percorre a lista de votações e de tasks e para cada uma, consoante o título dessa slot irá buscar tanto o custo como a pontuação referente a essa série nessa slot. Com estas duas variáveis é possível determinar uma variável que mais tarde (no labeling) irá ser maximizada.

Tudo isto é feito com base na fórmula sum(Votacao/Cost) em que Votacao é a pontuacao da serie na slot (que muda consoante a slot) e Cost é o custo da série (que é constante).

predicados: getValor, getVotacao, getCost

3.4 Estratégia de Pesquisa

Relativamente ao uso de labeling, a única opção utilizada foi o maximize(Valor), sendo Valor a variável retornada pelo getValor correspondendo à formula acima indicada. Este parâmetro é o que permite ao programa selecionar a melhor opção consoante a fórmula descrita.

Poderia se ter utilizado outras opções (como por exemplo bisect) mas de acordo com o programa desenvolvido e sua estrutura não fazia muito sentido, podendo tornar o programa até mais lento.

predicados: labeling

4 Visualização da Solução

Após o labeling é necessário representar a informação obtida sob forma de texto, para que seja possível ao utilizador ler a informação.

A tradução de informação é feita no predicado displayVar que recebe os parâmetros Slots (lista de slots, já com os títulos definidos) e Series (lista de séries). A partir deste momento apenas é necessário fazer "write"das informações necessárias, tendo em conta que o último parâmetro das Slots é agora o ID da Série correspondente.

predicados: displayVar

5 Resultados

Após alguns testes com diferentes parâmetros pode-se concluir que o factor que mais influencia a eficiencia do programa é a quantidade de slots para preencher (que segue um aumento de tempo exponencial sempre que se adiciona uma slot).

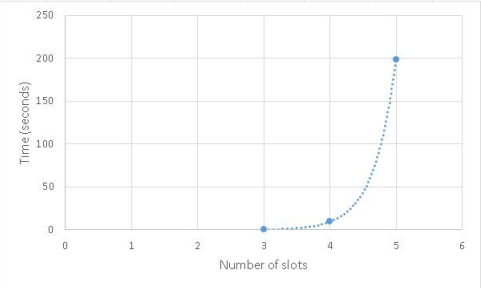

Figura 1: Descrição do tempo em função do número de slots a analisar

Com base na imagem pode-se concluir que a partir das 5 slots o programa irá demorar muito tempo para resolver.

De seguida são apresentados alguns gráficos com alguns diferentes testes feitos:

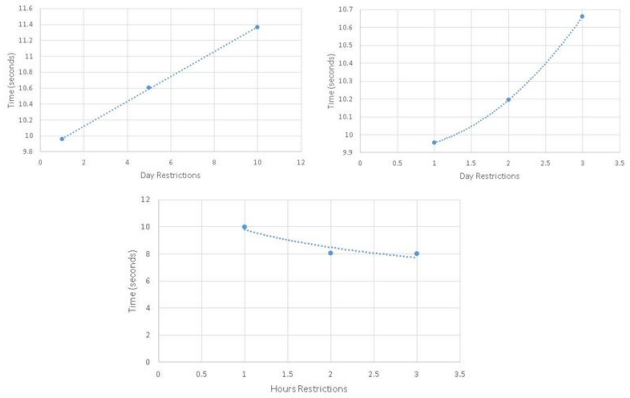

Figura 2: 3 gráficos: à esquerda, o tempo em função da quantidade de restrições de séries por dias (que não influencia o resultado), à direita, o tempo em função da quantidade de restrições de séries por dias (que influencia o resultado) e em baixo, o tempo em função da quantidade de restrições de séries por horas (que influencia o resultado).

Analisando, bem os gráficos podemos verificar que o tempo praticamente não se altera com a adição de restrições por dia (quer sejam utilizadas ou não) sendo apenas uma diferença de milisegundos.

Observando o gráfico das restrições por horas, quantas mais se adicionarem, mais o programa se torna mais rápido. Isto acontece pois se está a restringir diretamente o posicionamento das slots, isto é, não depende do posicionamento das outras séries nas slots, mas apenas de uma única. Ao restringir desta maneira, as escolhas de séries para 1 slot diminuem em 1 valor. Tendo isto em conta o gráfico segue uma reta logarítmica.

É importante referir ainda que se se retirar o método de optimização (maximize(Valor)), o processo de labeling ocorre instantaneamente (0 2ms), visto que admite a primeira seleção encontrada como válida.

Por fim podemos verificar o resultado final (com os exemplos do ficheiro example.pl).

```
| ?- solver.
Execution took 10885 ms.
[8,12,15,14]
Telenovela vai ser emitido Domingo das 10.0 as 10.5
Porto vai ser emitido Segunda-Feira das 10.0 as 10.5
Avatar vai ser emitido Quinta-Feira das 20.0 as 20.5
House MD vai ser emitido Quinta-Feira das 20.5 as 21.0
yes
| ?-
```

Figura 3: Resultado final

6 Conclusões e Trabalho Futuro

Com o final do trabalho pode-se concluir que PLR é uma ferramente bastante poderosa na resolução de problemas que envolvem variáveis a definir e restrições sobre as mesmas.

Programação em Lógica no geral é uma área extremamente interessante que usa abordagens e métodos de resolver problemas bastante diferentes dos

habituais.

No geral achei o trabalho difícil de fazer, não pela quantidade de trabalho, mas pela matéria em si. O facto de ter trabalhado sozinho também dificultou um pouco.

Referências

[1] SICStus Prolog, `https://sicstus.sics.se/`