Ano Escolar

Resolução de Problemas de Otimização utilizando Programação em Lógica com Restrições

João Barbosa and José Martins

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

Resumo Para este trabalho foi proposta a resolução de um problema de otimização referente a calendarização de testes e trabalhos de casa numa escola, utilizando programação em lógica com restrições em **SICStus Prolog**, permitindo variar o numero de turmas, disciplinas, trabalhos de casa por dia e por disciplina, assim como os horários de cada turma. Foi implementado um conjunto de predicados de modo a aplicar todas as restrições pertinentes para uma boa resolução do problema em questão.

Keywords: computational efficiency, logic programming, Prolog, restraints

1 Introdução

Este projeto esta a ser realizado no âmbito da unidade curricular Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação, da Faculdade de Engenharia da Universidade do Porto (FEUP). O objetivo do projeto prende-se com a resolução de um problema de otimização, utilizando programação em lógica com restrições. O problema consiste na calendarização de testes e trabalhos de casa de um periodo letivo, com vários parametros váriaveis como o número de turmas, disciplinas, trabalhos de casa (por dia e por disciplina), assim como a existência de diferentes horários para cada turma. Existem também diversas restrições que devem ser respeitadas. Ao longo deste artigo será descrito o respetivo problema com um nivel de detalhe mais elevado, a abordagem levada a cabo pelo grupo para a implementação de uma possivel solução, e as conclusões retiradas acerca da solução implementada.

2 Descrição do Problema

O Problema "Ano Escolar" é referente à maracação de testes e TPCs, ao longo do periodo, sendo que as seguintes restrições devem ser tidas em conta na resolução apresentada:

- Cada disciplina tem 2 testes por período de aulas, que decorrem num conjunto de semanas específico (mais ou menos a meio e no fim do período).
- Os alunos não podem ter mais do que 2 testes na mesma semana de aulas, nem testes em dias consecutivos.
- Os testes realizados pelas diferentes turmas a uma mesma disciplina devem ser o mais próximos possível.
- Em cada dia, não pode haver TPC a mais do que 2 disciplinas.
- Em pelo menos um dia por semana (que deve ser sempre o mesmo ao longo do período), não pode haver TPC.
- Em cada disciplina, só pode haver TPC, no máximo, em metade das aulas.

O resultado deve incluir as datas dos testes de cada turma/disciplina, bem como os dias em que o professor de cada disciplina pode mandar trabalho para casa.

Deve ser possível resolver problemas desta classe com diferentes parâmetros, como por exemplo variando o número de turmas e disciplinas, horários, número máximo de TPC por disciplina e por dia, entre outros.

3 Abordagem

A abordagem para o problema em questão consistiu, inicialmente, na avaliação das variáveis de decisão necessárias para a resolução do problema, e, por fim, na elaboração de uma estrutura que permitisse a imposição de restrições de um modo simples e eficiente.

Seguidamente, decidiu-se separar conceitos e elaborar predicados que resolvem o problema apenas para uma turma, que depois são utilizados para resolver o problema geral.

3.1 Variáveis de Decisão

A estrutura adoptada pelo grupo para resolver o problema consiste numa lista para cada turma com tamanho igual ao numero de disciplinas da turma, sendo que cada elemento dessa mesma lista é outra lista com 3 variaveis:

- Identificador da disciplina - Variável já instanciada (conhecida pelo problema);
- Lista de testes - Esta lista contém dois valores, sendo que cada um deles se refere à data do primeiro e segundo teste, respetivamente. Os domínios destas variáveis dependem do número de dias do período.
- Lista de trabalhos de casa - Lista com o tamanho igual ao numero de dias úteis do período. Cada elemento da lista se trata de uma variavel de decisão com dominio de 0 a 1, simbolizando a existencia (1) ou inexistência (0) de tabalhos de casa para a disciplina em questão.

3.2 Restrições

1. **Garantir dois testes por período para cada disciplina, mais ou menos a meio e no fim do período.**

   Para garantir o que o numero de testes por periodo para cada disciplina fosse igual a 2 e que cada teste estivesse numa época diferente (meio do periodo ou fim do periodo), foi utilizado o predicado *twoTestsPerPeriod(+Class)*, que recebe uma turma com a estrutura especificada no ponto **3.1** e, para cada disciplina dessa turma utiliza, a lista de variáveis de decisão que contém a informação dos testes. Foi arbitrado que o período seria divido em 4 partes: o primeiro *1/6* corresponde a um intervalo sem testes, e os seguintes *2/6* à época dos primeiros testes. O mesmo método foi aplicado para a onda de segundos testes (começando onde acaba o período para os primeiros).
   Os dois testes são garantidos, por só existirem duas variáveis em questão, para cada disciplina.

2. **Máximo de 2 testes na mesma semana de aulas, e não permitir testes em dias consecutivos.**

   Para solucionar esta restrição foi utilizado o predicado *testPlacementRestrictions(+Days,+Class)*, que uma turma, e o número de dias do período. Este predicado, auxiliando-se do predicado global *global_cardinality*, obtém uma lista de tamanho igual ao número de dias, e garante que:
   - De 5 em 5 dias, a soma de valores não pode ser maior que 2.
   - De 2 em 2 dias, a soma de valores não pode ser maior que 1.

   Desta forma, a restrição pedida é garantida. Note-se, ainda, que foram tidos em consideração os fins-de-semana, e, portanto, o predicado tem cuidado na transição de sexta para segunda, deixando que ambos os dias tenham testes (ainda que na representação interna sejam dias consecutivos).

3. **Os testes realizados pelas diferentes turmas a uma mesma disciplina devem ser o mais próximos possível.**

   De modo a permitir que os testes da mesma disciplina fossem o mais próximos possível em turmas diferentes foi utilizado o predicado *testsCloseBetweenClasses(+Classes, +DisciplineIds, -Sum1, -Sum2)* , que recebe todas as turmas e os identificadores de todas as disciplinas leccionadas nas turmas em questão, retornando o somatório da diferença entre os testes da mesma disciplina em turmas diferentes para a primeira época de testes (*Sum1*) e para a segunda (*Sum2*). Depois, estes valores, no processo de etiquetagem, serão minimzados, assegurando, desta forma, a condição apresentada.

4. **Em cada dia, não pode haver TPC em mais do que N disciplinas.**

   A fim de permitir ao utilizador especificar o número máximo de trabalhos de casa por dia, foi implementado o predicado *maxNumberTpcPerDay(+Class,+Days,+N)*, que obtém uma lista com o número total de TPCs em cada dia do periodo, para a turma em questão (tendo em consideração todas as disciplinas), sucedendo-se, de seguida, para cada elemento desta lista, uma comparação, de modo a garantir que o número de TPCs será inferior a N, uma vez instanciadas as variaveis de decisão.

5. **Em pelo menos um dia por semana (que deve ser sempre o mesmo ao longo do período), não pode haver TPC.**

   No sentido da implementação desta restrição foi criado o predicado *clearTpcDay(Class,NoTpcDay)* que recebe o dia em que não existem trabalhos de casa e depois percorre a lista de disciplinas da turma, de forma a que, na lista com as variáveis de decisão que dizem respeito aos TPCs, sejam instanciadas (com o valor 0) e, assim, indicar a não existencia de TPCs para todas as variaveis de decisão que dizem respeito ao dia da semana especificado nos parâmetros do predicado. Este valor deve variar entre 1 e 5 (em que cada

número corresponde ao dia da semana).

6. **Em cada disciplina, só pode haver TPC numa percentagem das aulas.**

   Em relação a esta restrição foi concebido o predicado *limitNumberOfTpcPerPeriod(+Class,+Ratio,+Schedule,+Days)* que recebe a razão especificada pelo utilizador (ex: 2 - metade, 3 - um terço, 4 - um quarto), e numa primeira fase é obtido o numero total de aulas de uma disciplina ao longo do período, sendo que depois a soma das variaveis de decisão referentes aos trabalhos de casa é limitada de modo a ser inferior ou igual a *1/Ratio*. Este processo é de seguida repetido para todas as disciplinas da turma.

3.3 Função de Avaliação

Na solução encontrada para o problema, um dos factores a ter em conta é ser necessário que os testes entre turmas, para uma dada disciplina, sejam o mais próximos entre si o quanto possível.

Para garantir esta condição, são calculadas as diferenças das datas de teste entre cada par de turmas, sendo estes valores também somados no final. Como há varios testes por disciplina, estas somas também terão de ser somadas - estamos, portanto, perante uma soma de somas de somas. Este valor deve ser minimizado, quando a solução está a ser procurada.

Para testar a eficiência e os resultados do programa, foi criado um predicado *run(+Days, +Schedules, -Classes)*, que calcula o tempo de execução do programa.

3.4 Estratégia de Pesquisa

A estratégia de etiquetagem passa por utilizar o predicado de *labeling* com a seguinte estrutura:

labeling([ffc, down, minimize(Sum), time_out([90000, _])], R)

Em que

- Sum - Soma das somas das somas entre a diferença de dias de teste para cada disciplina, entre cada turma.
- R - lista de variáveis de domínio, a serem instanciadas.

A opção ***ffc*** melhora a eficiência global do programa, utilizando a estratégia *First-Fail*, e, adicionalmente, instancia em primeiro lugar as variáveis com maior influência nos domínios restantes. Nos testes corridos, houve um grande aumento de eficiência utilizando esta opção, relativamente ao uso de ***ff*** ou nenhuma opção.

Utilizando ***down***, a instanciação das variáveis ocorre por ordem descendente do seu intervalo de domínio. Esta opção permite, de forma simples e eficiente,

garantir que são colocados o número máximo possível de TPCs para as turmas, que obedeçam às restrições dadas. Note-se que outra solução passaria por maximizar a soma de todos os TPCs, mas essa estratégia seria mais lenta, oferecendo os mesmos resultados.

O ***minimize***, para a variável *Sum*, garante que os testes, entre turmas, estão o mais próximos entre si, o quanto possível.

Finalmente, o ***time_out*** funciona como uma salvaguarda, no sentido em que para o programa, quando 1 minuto e 30 segundos estiverem decorridos, desde o início da execução. Caso decorra o tempo estipulado, então será retornada a melhor solução encontrada até então.

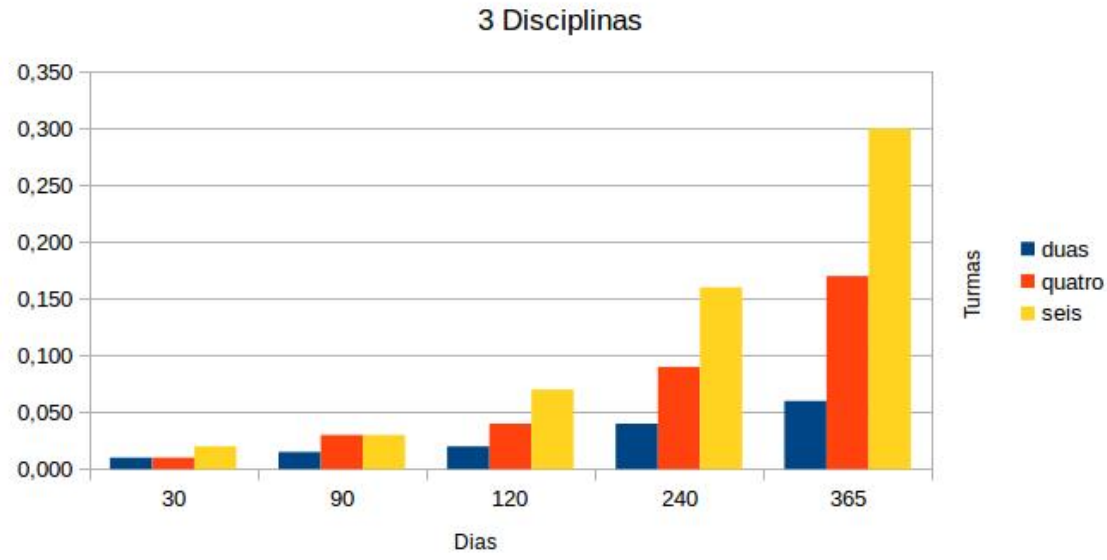
4 Visualização da Solução

Explicar os predicados que permitem visualizar a solução em modo de texto

5 Resultados

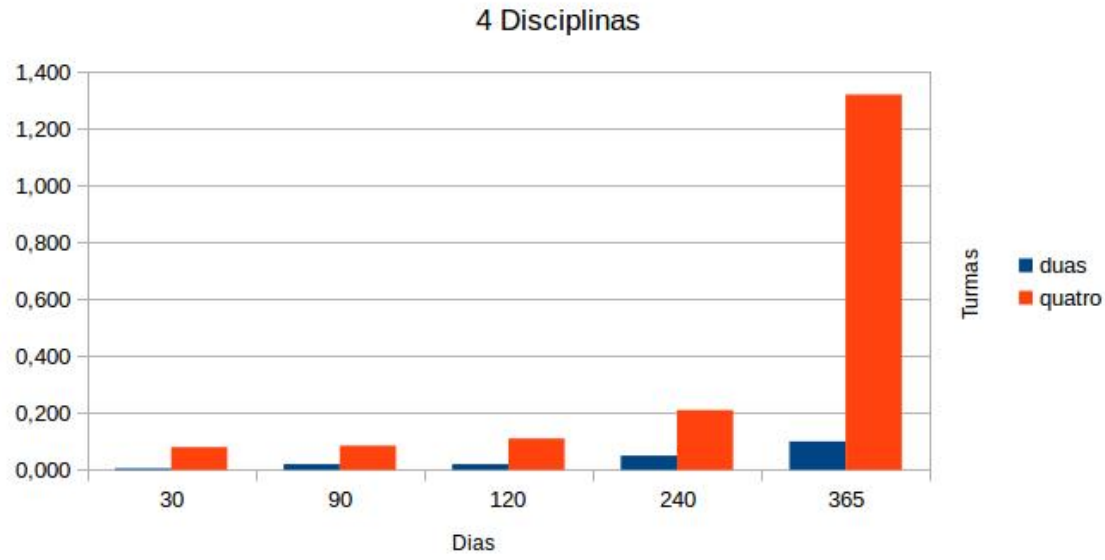
Nesta secção serão mostrados os resultados da execução do programa. As variáveis a considerar são: o número de disciplinas, o número de turmas, e o número de dias.

No primeiro gráfico são visíveis os resultados para 3 discplinas, variando o número de dias (entre 30 e 365) e o número de turmas (entre duas e seis):



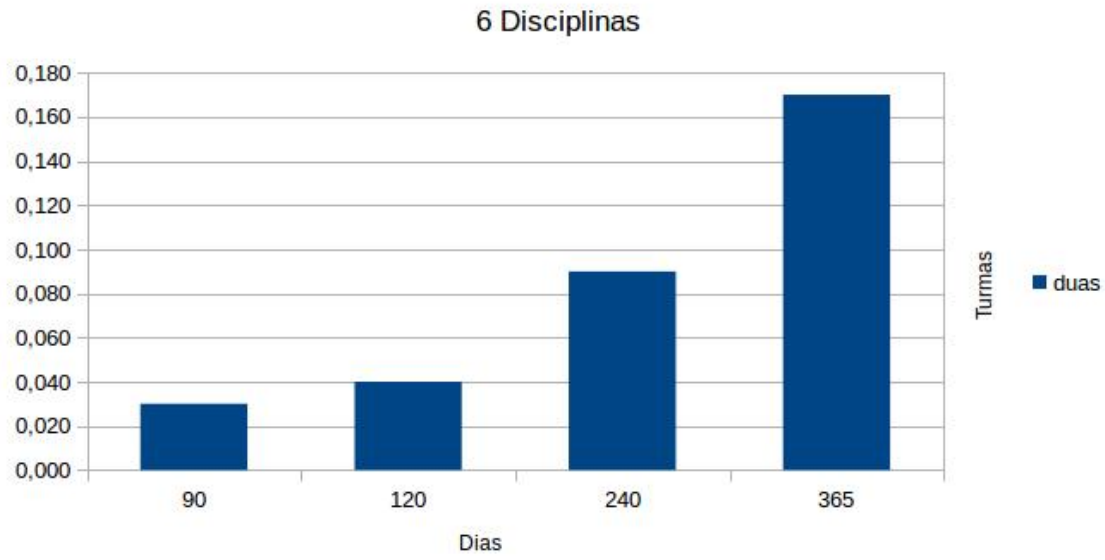
Os resultados mostram-se bons, sendo o tempo máximo de execução obtido *0.3 segundos*.

Seguidamente, o programa foi corrido num cenário de 4 disciplinas:



Note-se que não são mostrados os valores para seis turmas, pois o programa não conseguiu encontrar uma solução dentro do tempo estabelecido de 1 minuto e 30 segundos.
Os tempos de execução continuam a ser rápidos, excetuando para 4 turmas e 365 dias - que correu em *1.32 segundos*, mostrando-se irregular, comparativamente aos outros resultados.

Finalmente, as seguintes amostras foram obtidas com 6 disciplinas:



Os resultados não são mostrados para 4 e 6 turmas, pois, tal como aconteceu para 4 disciplinas, a solução ideal não foi encontrada no tempo estabelecido.

É importante referir que os tempos de execução dependem em grande escala dos *inputs* dados ao programa. Os valores apresentados podem variar bastante, tendo em conta os horários das turmas.

6 Conclusões e Trabalho Futuro

A construção deste projeto permitiu aprofundar os conhecimentos em *Programação em Lógica com Restrições*, assim como pensar de forma lógica (e não funcional). Foi possível, também, conhecer o poder do **Prolog** e, em específico, do predicado *labeling*, verificando como tornam possível a resolução de problemas complexos em poucas linhas de código.

Tendo em conta os resultados obtidos, é possível afirmar que, de uma forma geral, o programa tem tempos de execução rápidos. É, no entanto, importante notar que o aumento do número de disciplinas/turmas provoca uma diminuição na rapidez do programa mais acentuada, relativamente ao aumento do número de dias - estes últimos, apesar de dilatarem o tempo de execução, nunca foram a causa do programa não conseguir executar no tempo estabelecido.
Assim sendo, pode-se afirmar que o número de dias não se tratam do *bottleneck* do programa, podendo este estar no número de disciplinas ou o número de turmas.

A principal vantagem do programa, de uma forma geral, passa pela eficiência e baixo tempo de execução. Além disso, é totalmente flexível quanto ao número de disciplinas (que devem estar na base de dados), número de turmas e número de dias, e também quanto aos horários, o número máximo de TPCs por dia, percentagem de TPCs por período e ao dia que não deve haver TPCs.
No entanto, a flexibilidade nestes *inputs* torna o programa pouco previsível, podendo, por vezes, haver discrepâncias nos tempos de execução. Além disso, apesar de não ter sido pedido, a aplicação poderia ser ainda mais flexibilizada - podendo receber, por exemplo, o número de testes por período.

A melhoria do projeto passaria, então, por flexibilizar ainda mais os *inputs*, assim como identificar e melhorar a eficiência do *bottleneck* da aplicação.

Referências

1. SWI-Prolog, http://www.swi-prolog.org
2. SICStus-Prolog, https://sicstus.sics.se

Anexo

Código fonte

O código fonte encontra-se no ficheiro ***code.zip***.