Resolução de Problema de Decisão/Otimização usando Programação em Lógica com Restrições: Grupos de Trabalho

Ana Rita Norinho Pinto and Joana Maria Cerqueira da Silva
Grupos de Trabalho_4

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

Resumo. O projeto proposto, no âmbito da unidade curricular de Programação em Lógica, foi a criação de um programa no Sistema de Desenvolvimento SICStus Prolog implementando Restrições que seja capaz de resolver um problema de decisão/otimização. O problema escolhido foi o problema de otimização de grupos de trabalho, que consiste na formação de grupos de trabalho para dois projetos de uma unidade curricular. Este trabalho foi realizado com sucesso sendo que neste artigo será explicado minuciosamente todo o processo de resolução do problema proposto.

Palavras-chave: Grupos de Trabalho, Prolog, SICStus, Restrições, Variáveis de Domínio, FEUP.

1 Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Programação em Lógica, unidade curricular do 3º ano do Mestrado Integrado em Engenharia Informática e de Computação da Faculdade de Engenharia da Universidade do Porto. Para o mesmo foi necessário desenvolver uma possível solução para um problema de decisão ou de otimização na linguagem Prolog, implementando restrições.

O problema escolhido para ser resolvido foi um problema de otimização com o nome de Grupos de Trabalho. Este problema consiste na alocação de estudantes de uma unidade curricular em grupos para dois projetos, sendo que existem vários fatores a ser considerados para a distribuição dos mesmos.

O artigo apresenta a seguinte estrutura:

- **Descrição do Problema:** descrição em detalhe do problema de otimização/decisão em análise.
- **Abordagem:** descrição da modelação do problema como um PSR, de acordo com as seguintes subsecções:
  - **Variáveis de Decisão:** descrição das variáveis de decisão e os seus domínios.
  - **Restrições:** descrição das restrições rígidas e flexíveis do problema e a sua implementação usando o SICStus Prolog.

  - **Fundação de Avaliação:** descrição da forma de avaliar a solução obtida problema e a sua implementação usando o SICStus Prolog.
  - **Estratégia de pesquisa:** descrição da estratégia de etiquetagem utilizada ou implementada.
- **Visualização da Solução:** explicação dos predicados que permitem visualizar a solução em modo de texto.
- **Resultados:** descrição dos resultados obtidos para diferentes dimensões do problema.
- **Conclusões e Trabalho Futuro:** conclusões retiradas da realização do projeto e possíveis melhorias.
- **Bibliografia:** referências a livros, artigos e páginas Web usadas no desenvolvimento do projeto.
- **Anexo:** código fonte, ficheiros de dados e resultados, entre outros…

2 Descrição do problema

O problema de Grupos de Trabalho é um problema de otimização. É pretendido que se formem grupos de trabalho para dois projetos de uma dada unidade curricular, de forma a que certos requisitos sejam cumpridos e otimizados.

É necessário evitar colocar no mesmo grupo pessoas que já tenham trabalhado juntas no passado, embora tal seja possível acontecer. O número de temas disponíveis é fixado à partida, tendo em conta o número de estudantes inscritos na UC, e os grupos terão de apresentar entre três e quatro elementos, ter médias equilibradas e não ter repetições entre o primeiro e o segundo trabalho.

Ou seja, as médias dos membros de cada grupo não podem ser muito desniveladas e não poderá haver dois estudantes que trabalharam juntos no primeiro trabalho a trabalhar juntos no segundo.

3 Abordagem

Na resolução deste problema foi usado um predicado **class(ListClass,NumberStudents)** que possui diversas instanciações para diferentes números de alunos. Cada posição da lista (**ListClass**) corresponde ao grupo em que o estudante desse índice trabalhou nessa unidade curricular.

```
class([1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,5,5],20).
```

Como podemos observar na imagem acima, por exemplo, os alunos com o índice 0, 2, 7 e 8 trabalharam juntos no passado, uma vez que se encontravam no mesmo grupo, que, neste caso, é o grupo número 1.

Para a geração das médias dos alunos, foi também criado um predicado **grades(ListGrades,NumberStudents)** em que cada elemento da lista é correspondente à média do aluno do respetivo índice. A estrutura é idêntica ao predicado **class**.

3.1 Variáveis de Decisão

A solução do problema vem na forma de 2 listas que representam a distribuição dos alunos em grupos para dois projetos diferentes. Os tamanhos das listas são iguais ao número de alunos passado como parâmetro.

3.2 Restrições

Restrições Flexíveis

- **É pretendido evitar colocar no mesmo grupo pessoas que já tenham trabalhado juntas no passado em diferentes UCs, no entanto tal pode acontecer.**

Para que isso aconteça, é usado o seguinte predicado que verifica se os grupos atuais de dois pares são iguais e se na unidade curricular anterior também eram. Se isso acontecer, é incrementado um acumulador. Posteriormente, nas opções do labeling este valor é minimizado. O predicado **count_repetitions(Groups,Acc,LastUcGroups)** utiliza o predicado **repetitions(Groups,LastUcGroups,Acc)** que compara o primeiro elemento de cada lista com os restantes da mesma e em *Bin* é guardado o valor da veracidade da restrição: se dois elementos trabalharam juntos na última unidade curricular e agora isso repete-se então o valor de Bin é 1, caso contrário é 0. Este valor é adicionado ao acumulador corrente.

```
count_repetitions([_],0,[_]).

count_repetitions([H|T],Acc,[X1|X]):- count_repetitions(T,Acc2,X),
                                      repetitions([H|T],[X1|X],Acc1),
                                      Acc #= Acc2+Acc1.

repetitions([A,B],[X1,X2],Acc):- (X1 #= X2 #/\ A #=B)#<=>Bin, Acc #= Bin.
repetitions([A,B,C|T],[X1,X2,X3|X],Acc):-
                                        repetitions([A,C|T],[X1,X3|X],Acc1),
                                        (X1 #= X2 #/\ A #=B)#<=>Bin,
                                        Acc #= Acc1 + Bin.
```

- **Os grupos devem de ser o mais equilibrados possíveis, não havendo muito desnível nas médias dos elementos de cada grupo.**

Para tal, é calculada a soma de notas de cada grupo através do predicado **sum_grades(Group,Grades,CurrentGroup,SumGrades)** em que Group é a lista de grupos final, Grades é a lista de notas da turma, CurrentGroup é o grupo do qual a média está a ser calculada e SumGrades o acumulador.

Este predicado é invocado através do **calculate_average_grades(List,Grades,GroupElements,CurrentGroup,NumberGroups,FinalAverageGrades)**. Este recebe a lista de alunos, a lista de notas, a lista com o número de elementos de cada grupo, o grupo atual, o número de grupos e a lista de médias de cada grupo. Por fim, é usado o predicado general_distance_between_grades(GroupGrades,CurrentDistanceGrades,DistanceGrades) que calcula a diferença entre cada uma das médias dos grupos. No solver, é calculado o valor máximo desta lista e nas opções do labeling este valor é minimizado.

```
calculate_average_grades(List,Grades,GroupElements,CurrentGroup,NumberGroups,[]).
calculate_average_grades(List,Grades,[G1|G],CurrentGroup,NumberGroups,[H|T]):-
                                                    sum_grades(List,Grades,CurrentGroup,Sum),
                                                    H #= Sum / G1,
                                                    CurrentGroup1 #= CurrentGroup+1,
                                                    calculate_average_grades(List,Grades,G,CurrentGroup1,NumberGroups,T).

sum_grades([A],[G1],Group,SumGrades):- (A #= Group) #<=> Bin,
                                       SumGrades #= G1 * Bin.

sum_grades([A,B|T],[G1,G2|G],Group,SumGrades):-       sum_grades([B|T],[G2|G],Group,Sum),
                                                      (A #= Group) #<=> Bin,
                                                      SumGrades #= Sum + G1 * Bin.

general_distance_between_grades([_],FinalList,FinalList).

general_distance_between_grades([H|T],CurrList,FinalList):- distance_between_grades([H|T],[],FinalList1),
                                                            append(CurrList,FinalList1,CurrList1),
                                                            general_distance_between_grades(T,CurrList1,FinalList).

distance_between_grades([A,B],CurrList,FinalList):- Diff #= abs(B-A),
                                                    append(CurrList,[Diff],CurrList1),
                                                    FinalList = CurrList1.
distance_between_grades([A,B,C|T],CurrList,FinalList):-
                                                      Diff #= abs(B-A),
                                                      append(CurrList,[Diff],CurrList1),
                                                      distance_between_grades([A,C|T],CurrList1,FinalList).
```

Restrições Rígidas

- **Não existem repetições de grupos do primeiro trabalho para o segundo.**

Para tal, é usado o predicado **count_repetitions(Groups,Acc,Groups1)** acima exibido em que o acumulador tem o valor 0 uma vez que não podem existir repetições de pares entre Groups e Groups1.

- **Grupo deve de ter entre 3 e 4 elementos**.

Os grupos formados devem ter entre 3 e 4 elementos e o número total de grupo varia consoante o número de temas estabelecidos inicialmente. Para calcular o número de grupos é utilizado predicado **calculate_number_groups** que recebe o número de estudantes e o número de temas e calcula o número de grupos de 4 elementos e o número de grupos de 3 elementos de modo a respeitar as restrições estabelecidas.

```
calculate_number_groups(Students,NumberGroups4,NumberGroups3,NumberThemes):-
                        Values=[NumberGroups4,NumberGroups3],
                        NumberGroups3 in 0..Students,
                        NumberGroups4 in 0..Students,
                        Students #= NumberGroups3 * 3 + NumberGroups4 * 4,
                        NumberThemes #= NumberGroups3+NumberGroups4,
                        labeling([],Values).
```

Após este cálculo, é construída uma lista consoante o número de elementos de cada grupo. Por outras palavras, o número de vezes que o identificativo de um grupo se repete na lista de grupos. O predicado em causa denomina-se **list_global.**

```
list_global(CurrPosition,NumberGroups,NumberElements,FinalList,FinalList,ListGroups,ListGroups):- CurrPosition >= NumberGroups.
list_global(CurrPosition,NumberGroups,NumberElements,CurrList,FinalList,CurrentGroups,FinalGroups):-
                        CurrPosition < NumberGroups,
                        CurrPosition1 is CurrPosition+1,
                        append(CurrList,[CurrPosition1-NumberElements],CurrList1),
                        append(CurrentGroups,[NumberElements],CurrentGroups1),
                        list_global(CurrPosition1,NumberGroups,NumberElements,CurrList1,FinalList,CurrentGroups1,FinalGroups).
```

3.3 Função de Avaliação

Sendo este um problema de otimização, o objetivo não é simplesmente encontrar uma solução, mas sim encontrar a melhor solução e essa é aquela que respeita os seguintes tópicos:

- O menor número de repetições de pares nos grupos da unidade curricular corrente e da unidade curricular passada.
- As médias serem o mais próximo possível.

Para que isso seja respeitável, são usados os predicados **count_repetitions**, que devolve o valor do número de repetições e **generate_distance_between_grades** que calcula a diferença entre cada uma das médias do grupo. De seguida, é calculada a máxima diferença através do predicado **maximum**.

```
general_distance_between_grades(AverageGrades,[],DiffGrades),
maximum(Max,DiffGrades),
```

Assim, para gerar a melhor solução é efetuada uma combinação dos dois valores para minimizar. Neste caso, foi considerado que é mais relevante as médias serem mais próximas do que haver pares repetidos de outras unidades curriculares. Portanto,

Metric #= 3 * Max / 4 + Acc /4

Por fim, este processo é repetido duas vezes uma vez que se pretende formar grupos para dois trabalhos diferentes e no fim são somadas as duas combinações a minimizar da seguinte forma:

FinalMetric #= Metric1 + Metric2

3.4 Estratégia de pesquisa

Para a resolução deste problema foram testadas variadas opções de pesquisa utilizando um *timeout* de 20 segundos. Através dos múltiplos testes pudemos concluir que a melhor estratégia de pesquisa é o uso das opções ***step first_fail*** no labeling, sendo a segunda melhor estratégia o uso das opções ***median first_fail***. Também foi possível concluir que a pior estratégia são as opções ***bissect anti_first_fail,*** uma vez que não se atingiu nenhuma solução nos 20 segundos dados para a resolução. Os valores concretos testados podem ser consultados na **tabela 1** que se encontra em Anexo.

4 Visualização da Solução

Após ser encontrada uma solução para o problema, esta é exibida de forma a uma fácil perceção daquilo que foi encontrado.
Para instanciar o problema deve ser feita uma chamada na consola a groups com os atributos número de estudantes e número de temas. É necessário ter em atenção quais as dimensões dos predicados que instanciam a lista de notas e a lista da turma uma vez que o comprimento destas deve estar de acordo com o número de estudantes introduzido e ser possível formar grupos de 3 e 4 elementos a partir do número de alunos consoante o número de temas.

```
groups(NumberStudents,NumberThemes):- write('Processing...'),nl,
                    class(X,NumberStudents),
                    grades(ListGrades,NumberStudents),
                    statistics(walltime, [Start,_]),
                    length(Groups1,NumberStudents),
                    calculate_number_groups(NumberStudents,Groups4,Groups3,NumberThemes),
                    NumberGroups #= Groups3+Groups4,
                    domain(Groups1,1,NumberGroups),
                    list_global(0,Groups3,3,[],List,[],CurrentGroups),
                    list_global(Groups3,NumberGroups,4,List,FinalList,CurrentGroups,FinalGroups),
                    global_cardinality(Groups1,FinalList),
                    count_repetitions(Groups1,Acc,X),length(AverageGrades,NumberGroups),
                    calculate_average_grades(Groups1,ListGrades,FinalGroups,1,NumberGroups,AverageGrades),
                    general_distance_between_grades(AverageGrades,[],DiffGrades),
                    maximum(Max,DiffGrades),
                    Metric #= 3*Max / 4 + Acc /4,
                    length(Groups2,NumberStudents),
                    domain(Groups2,1,NumberGroups),
                    global_cardinality(Groups2,FinalList),
                    count_repetitions(Groups2,0,Groups1),length(AverageGrades1,NumberGroups),
                    count_repetitions(Groups2,Acc1,X),
                    calculate_average_grades(Groups2,ListGrades,FinalGroups,1,NumberGroups,AverageGrades1)
                    general_distance_between_grades(AverageGrades1,[],DiffGrades1),
                    maximum(Max1,DiffGrades1),
                    Metric1 #= 3* Max1 / 4 + Acc1 / 4,
                    TotalMetrics #= Metric1+Metric,
                    append(Groups1,Groups2,TotalGroups),
                    labeling([minimize(TotalMetrics),time_out(70000,_)],TotalGroups),
                    statistics(walltime, [End,_]),
                    Time is End - Start,
                    fd_statistics,
                    output(ListGrades,X,Groups1,AverageGrades,Groups2,AverageGrades1,Acc,Acc1,Time).
```

Diversos predicados para distintas dimensões.

```
class([1,1,1,2,2,2,3,3,3],9).
class([1,1,2,2,4,4,5,5,1,2,3,4],12).
class([1,1,1,3,2,2,4,2,3,4,3,4,3,4],14).
class([1,3,4,3,4,1,2,2,1,1,2,2,3,3,4],15).
class([1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4],16).
class([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18],18).
class([1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,5,5],20).
class([1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,6,5,5,6,6,6],24).

grades([15,11,13,13,14,13,14,15,16],9). % 9 alunos
grades([12,16,17,13,12,13,18,16,12,18,12,19],12).%12 alunos
grades([14,11,10,12,17,13,12,18,12,15,18,19,18,11],14). % 14 alunos
grades([13,12,18,18,11,14,11,10,12,15,12,19,18,19,17],15). %15 alunos
grades([12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12],16). % 16 alunos
grades([15,17,17,19,15,10,12,17,11,15,11,19,15,16,18,13,14,10],18). %18 alunos
grades([13,11,13,19,15,12,15,13,14,11,18,14,11,17,14,18,11,18,19,20],20). %20 alunos
grades([10,13,16,15,10,12,18,13,19,14,14,10,17,14,12,19,11,19,19,12,13,13,14,15],24). %24alunos
```

Após ser resolvido o problema, é mostrado na consola a lista das notas da turma, a lista dos grupos de uma unidade curricular anterior, a lista de grupos para o primeiro trabalho, as médias dos grupos do primeiro trabalho, a lista de grupos para o segundo trabalho, as médias dos grupos do segundo trabalho, o número de repetições de pares de uma unidade curricular anterior e do primeiro grupo, o número de repetições de uma unidade curricular anterior e o segundo grupo e ainda o tempo despendido.

```
output(ListGrades,X,Groups1,AverageGrades,Groups2,AverageGrades1,Time):-
                                write('Grades > ') , write(ListGrades),nl,
                                write('Other class groups > ') , write(X),nl,
                                write('First work groups > '),write(Groups1),nl,
                                write('First work groups average grades > '), write(AverageGrades),nl,
                                write('Second work groups > ' ), write(Groups2),nl,
                                write('Second work groups average grades > '), write(AverageGrades1),nl,
                                format(' > Duration: ~3d s~n', [Time]),nl.
```

Por exemplo, fazendo a seguinte chamada : groups (16,4). Os predicados para instanciar os grupos de uma unidade curricular anterior e as notas de cada aluno são os seguintes:

```
class(X,NumberStudents),
grades(ListGrades,NumberStudents),
```

O output gerado é o seguinte:

```
| ?- groups(16,4).
Processing...
Grades > [12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12]
Other class groups > [1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4]
First work groups > [1,1,2,3,1,2,3,4,1,2,2,3,3,4,4,4]
Repetitions between first groups and other class groups > 2
First work groups average grades > [13,13,13,14]
Second work groups > [1,2,1,3,3,3,2,4,4,4,2,4,1,2,3,1]
Second work groups average grades > [13,13,14,13]
Repetitions between second groups and other class groups > 3
 > Duration: 11.229 s
```

5 Resultados

De forma a poder retirar conclusões dos resultados obtidos com a solução desenvolvida foram realizadas várias medições, tais como, a medição do tempo de obtenção da melhor solução até ao momento, o número de retrocessos exigidos na solução aplicada e, também, o número de restrições aplicadas. As condições de teste e os resultados obtidos encontram-se de seguida:

1. **Fazer variar o número de pessoas existentes na turma (Tabela 2, Figuras 1, 2 e 3 em Anexo)**

A partir da tabela podemos inferir que tanto o tempo de execução da solução, o número de retrocessos e o número de restrições aplicadas variam, tendencialmente, de uma forma linear com o número de estudantes existentes na turma. Podemos então concluir através da análise do gráficos e da tabela que quanto maior for o número de estudantes, maior vai ser o tempo de resolução do problema, mais retrocessos vamos ter e mais restrições vão ser aplicadas na solução.

2. **Fazer variar o número de grupos que devem de existir na turma (Tabela 3 e 4, Figuras 4, 5, 6, 7, 8, 9 em Anexo)**

Na tabela 3 podemos observar o registo da variação de tempo, retrocessos e restrições aplicadas, em função do número de grupos de uma turma de 16 alunos.

Para o caso do tempo e do número de retrocessos (figuras 4 e 5), pode-se concluir que têm uma tendência para diminuir quando o número de grupos na turma aumenta, havendo uma variação linear negativa. No entanto, para o caso do número de restrições (figura 6), há uma correlação linear positiva em função do número de grupos, havendo mais restrições se o número de grupos for maior.

Na tabela 4 temos presentes as mesmas condições existentes na tabela 3, mudando apenas o número de estudantes da turma de 16 para 24.

A partir dos gráficos das figuras 7, 8 e 9, podemos constatar que quanto menor o número de elementos na turma e quanto menor for o número de grupos existentes, mais rápida será a obtenção da melhor solução, menos retrocessos haverão e será menor o número de restrições aplicadas. Foi possível deduzir estes factos através do aumento linear de todas estas condições em função do aumento do número de alunos e de grupos.

6 Conclusões e Trabalho Futuro

O projeto foi uma excelente forma de consolidar e aplicar o conhecimento adquirido ao longo do semestre nas aulas práticas e teóricas. Notou-se também que a implementação de restrições na linguagem de Prolog é uma ferramenta muito útil e eficaz de se aplicar na resolução de problemas tanto de decisão como de otimização.

Ao longo da realização do projeto foram encontradas algumas dificuldades, maioritariamente no início ao tentar perceber qual seria a melhor forma de implementar as restrições exigidas, no entanto, as dúvidas foram sendo resolvidas após uma melhor compreensão da biblioteca *clpfd* e da comunicação tanto entre os membros do grupo como com o professor.

Quanto a aspetos que podiam ser trabalhados num futuro trabalho, pensamos que seria o facto de podermos ter mais do que uma lista a representar os trabalhos das unidades anteriores, de forma a se ter mais do que uma opção de pessoas já terem trabalhado juntas antes. De resto, pensamos que o problema foi resolvido da melhor forma que possível, apresentando todos os requisitos pedidos.

Concluindo, o projeto proposto foi desenvolvido com sucesso, não só tendo sido arranjada uma solução para o problema escolhido, mas tendo também a sua realização contribuído bastante para uma melhor aprendizagem e compreensão do uso de restrições em Prolog.

Bibliografia

1. Carlsson, M.: SICStus Prolog User's Manual 4.4.1. RISE SICS AB, Sweden (2018).
2. SICStus Homepage, https://sicstus.sics.se/documentation.html, last accessed 2018/12/19.

Anexo

Tabelas e Gráficos

Tabela 1. Testes de Estratégia de Pesquisa

	leftmost	min	max	first_fail	anti_first_fail	occurrence	most_constrained	max_regret
step	11,403	20,053	20,044	10,197	20,055	20,051	20,044	10,455
enum	12,068	20,032	20,035	10,507	20,036	20,042	20,043	11,148
bisect	11,010	20,051	20,024	10,587	20,048[1]	20,039	20,046	10,531
middle	11,324	20,145	20,034	10,434	20,043	20,046	20,046	11,350
median	11,326	20,020	20,050	10,449	20,017	20,051	20,046	11,343

Tabela 2. Variação do tempo, retrocessos e restrições criadas em função do número de estudantes

Número de Grupos	4				
Número de Pessoas	**Tempo(s)**	**Retrocessos**	**Restrições Criadas**	**Class**	**Médias**
12	0,045	137	1252	[1,1,2,2,4,4,5,5,1,2,3,4]	[12,16,17,13,12.13.18,16,12,18,12,19]
14	2,451	20145	1919	[1,1,1,3,2,2,4,2,3,4,3,4,3,4]	[14,11,10,12,17,13,12,18,12,15,18,19,18,11]
15	7,520	58783	2459	[1,3,4,3,4,1,2,2,1,1,2,2,3,3,4]	[13,12,18,18,11,14,11,10,12,15,12,19,18,19,17]
16	11,317	70559	3095	[1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4]	[12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12]

[1] Passados os 20 dados segundos para a resolução do problema (timeout) ainda não se tinha atingido nenhuma solução para as opções bisect anti_first_fail.

Tabela 3. Variação do tempo, retrocessos e restrições criadas em função do número de grupos

Número de Pessoas	16				
Número de Grupos	**Tempo(s)**	**Retrocessos**	**Restrições Criadas**	**Class**	**Médias**
4	11,317	70559	3095	[1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4]	[12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12]
5	5,668	39435	3209	[1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4]	[12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12]

Tabela 4. Variação do tempo, retrocessos e restrições criadas em função do número de grupos

Número de Pessoas	24				
Número de Grupos	**Tempo(s)**	**Retrocessos**	**Restrições Criadas**	**Class**	**Médias**
6	56.462	343126	8591	[1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,6,5,5,6,6,6]	[10,13,16,15,10,12,18,13,19,14,14,10, 7,14,12,19,11,19,19,12,13,13,14,15]
7	72.710	362199	8761	[1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,6,5,5,6,6,6]	[10,13,16,15,10,12,18,13,19,14,14,10, 17,14,12,19,11,19,19,12,13,13,14,15]
8	77.541	402229	8935	[1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,6,5,5,6,6,6]	[10,13,16,15,10,12,18,13,19,14,14,10, 17,14,12,19,11,19,19,12,13,13,14,15]

12

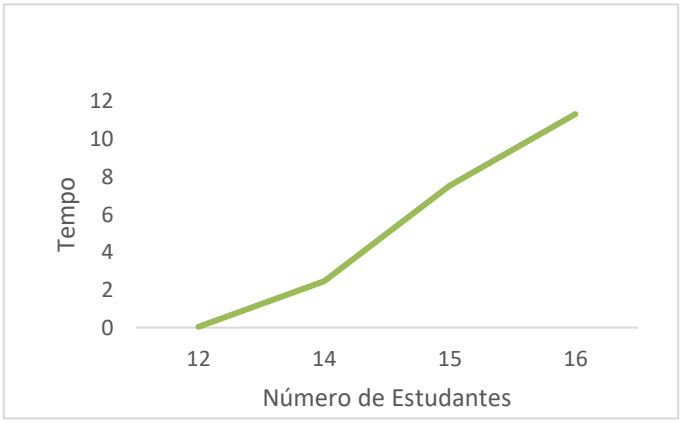

Fig. 1. Variação do tempo de resolução em função do número de estudantes na turma

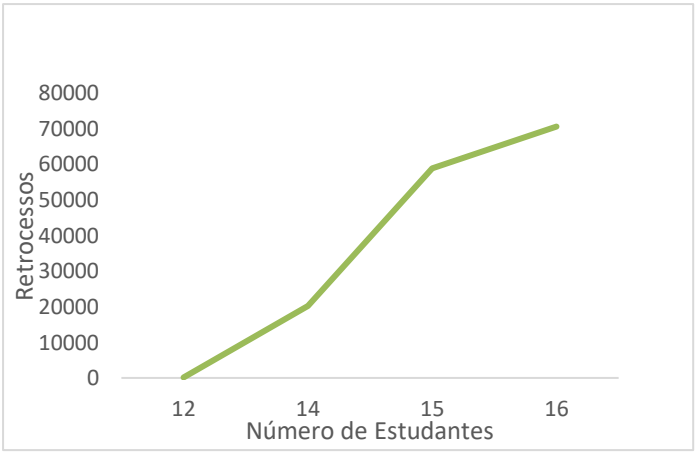

Fig. 2. Variação do número de retrocessos em função do número de estudantes na turma

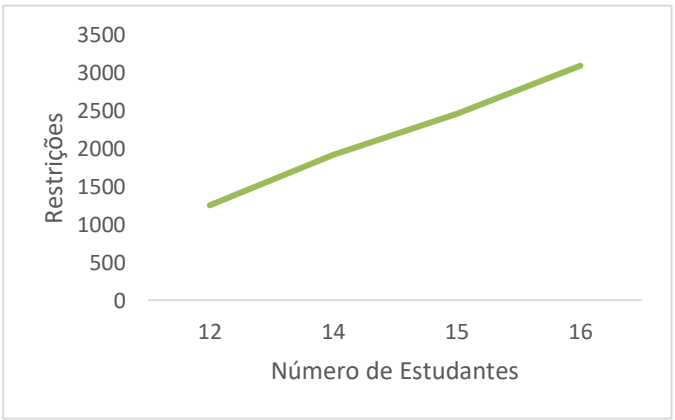

Fig. 3. Variação do número de restrições em função do número de estudantes na turma

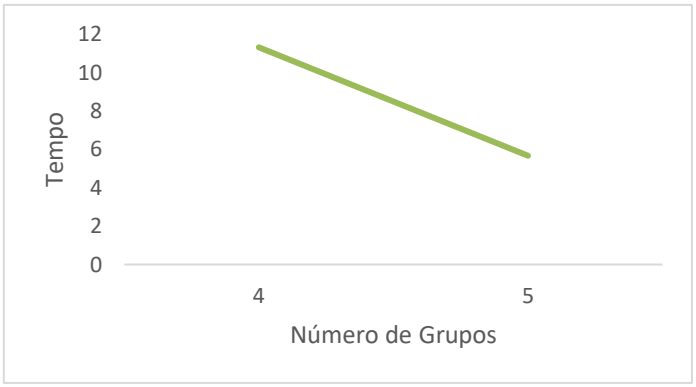

Fig. 4. Variação do tempo de resolução em função do número de grupos numa turma com 16 alunos

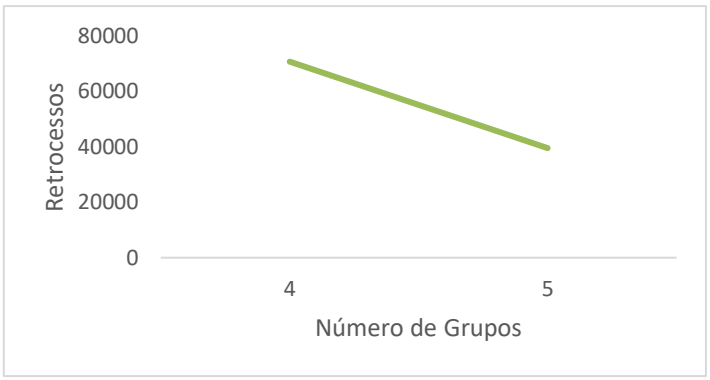

Fig. 5. Variação do número de retrocessos em função do número de grupos numa turma com 16 alunos

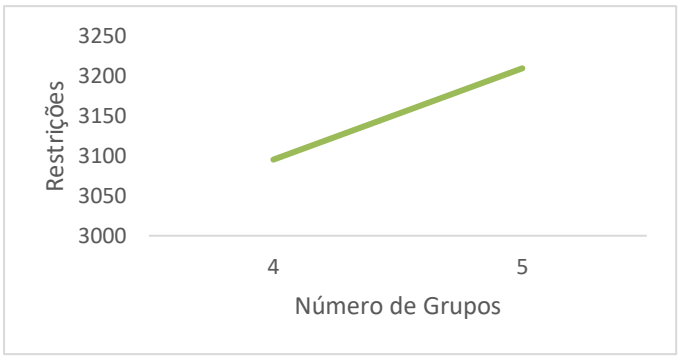

Fig. 6. Variação do número de restrições em função do número de grupos numa turma com 16 alunos

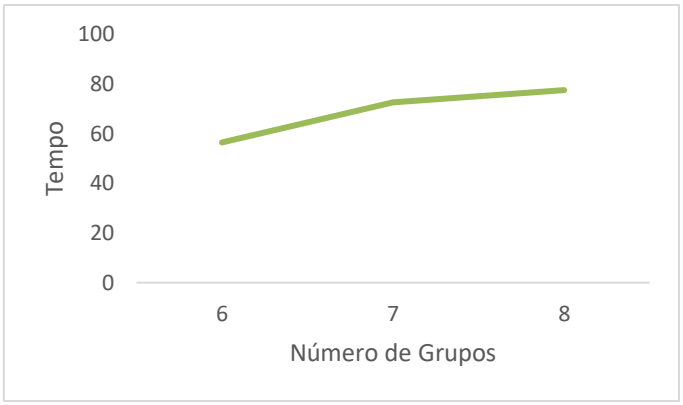

Fig. 7. Variação do tempo de resolução em função do número de grupos numa turma com 24 alunos

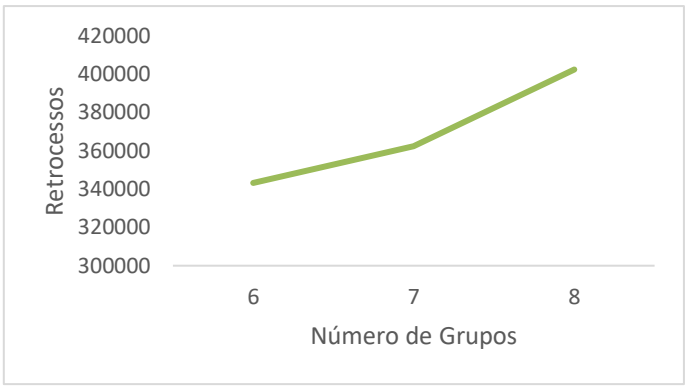

Fig. 8. Variação do número de retrocessos em função do número de grupos numa turma com 24 alunos

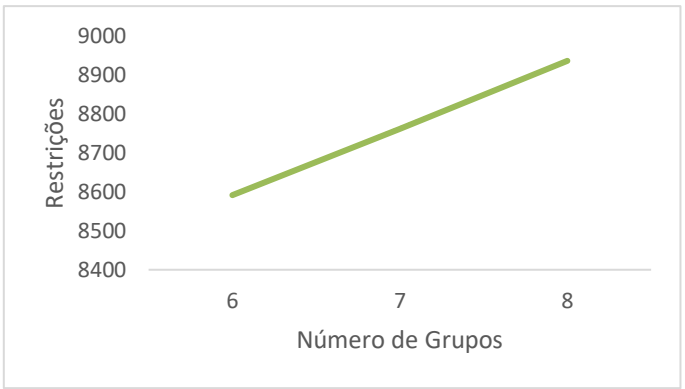

Fig. 9. Variação do número de restrições em função do número de grupos numa turma com 24 alunos

Código Fonte

```
:-use_module(library(clpfd)).
:-use_module(library(random)).
:-use_module(library(lists)).


% diferentes turmas de ucs de anos anteriores
% indice corresponde ao aluno e o elemento é o grupo em qual o aluno trabalhou
class([1,1,1,2,2,2,3,3,3],9).
class([1,1,2,2,4,4,5,5,1,2,3,4],12).
class([1,1,1,3,2,2,4,2,3,4,3,4,3,4],14).
class([1,3,4,3,4,1,2,2,1,1,2,2,3,3,4],15).
class([1,1,1,1,2,2,2,2,3,4,3,4,3,4,3,4],16).
class([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18],18).
class([1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,5,5],20).
class([1,2,1,2,2,2,3,1,1,3,3,4,4,4,3,4,5,5,6,5,5,6,6,6],24).


% medias dos estudantes nas turmas das ucs de anos anteriores
grades([15,11,13,13,14,13,14,15,16],9). % 9 alunos
grades([12,16,17,13,12,13,18,16,12,18,12,19],12).%12 alunos
grades([14,11,10,12,17,13,12,18,12,15,18,19,18,11],14). % 14 alunos
grades([13,12,18,18,11,14,11,10,12,15,12,19,18,19,17],15). %15 alunos
grades([12,11,10,12,17,11,12,18,12,15,18,10,18,11,17,12],16). % 16 alunos
grades([15,17,17,19,15,10,12,17,11,15,11,19,15,16,18,13,14,10],18). %18 alunos
grades([13,11,13,19,15,12,15,13,14,11,18,14,11,17,14,18,11,18,19,20],20). %20 alunos
grades([10,13,16,15,10,12,18,13,19,14,14,10,17,14,12,19,11,19,19,12,13,13,14,15],24). %24alunos

generate_grades(0,FinalList,FinalList):- write(FinalList).
generate_grades(Index,CurrList,FinalList):-
                    Index > 0,
                    Index1 is Index-1,
                    random(10,20,H),
                    append(CurrList,[H],CurrList1),
                    generate_grades(Index1,CurrList1,FinalList),!.

output(ListGrades,X,Groups1,AverageGrades,Groups2,AverageGrades1,Time):-
                                        write('Grades > ') , write(ListGrades),nl,
                                        write('Other class groups > ') , write(X),nl,
                                        write('First work groups > '),write(Groups1),nl,
                                        write('First work groups average grades > '), write(AverageGrades),nl,
                                        write('Second work groups > ' ), write(Groups2),nl,
                                        write('Second work groups average grades > '), write(AverageGrades1),nl,
                                        format(' > Duration: ~3d s~n', [Time]),nl.

```

```
45
46  groups(NumberStudents,NumberThemes):- write('Processing...'),nl,
47                          class(X,NumberStudents),
48                          grades(ListGrades,NumberStudents),
49                          statistics(walltime, [Start,_]),
50                          length(Groups1,NumberStudents),
51                          calculate_number_groups(NumberStudents,Groups4,Groups3,NumberThemes),
52                          NumberGroups #= Groups3+Groups4,
53                          domain(Groups1,1,NumberGroups),
54                          list_global(0,Groups3,3,[],List,[],CurrentGroups),
55                          list_global(Groups3,NumberGroups,4,List,FinalList,CurrentGroups,FinalGroups),
56                          global_cardinality(Groups1,FinalList),
57                          count_repetitions(Groups1,Acc,X),length(AverageGrades,NumberGroups),
58                          calculate_average_grades(Groups1,ListGrades,FinalGroups,1,NumberGroups,AverageGrades),
59                          general_distance_between_grades(AverageGrades,[],DiffGrades),
60                          maximum(Max,DiffGrades),
61                          Metric #= 3*Max / 4 + Acc /4,
62                          length(Groups2,NumberStudents),
63                          domain(Groups2,1,NumberGroups),
64                          global_cardinality(Groups2,FinalList),
65                          count_repetitions(Groups2,0,Groups1),length(AverageGrades1,NumberGroups),
66                          count_repetitions(Groups2,Acc1,X),
67                          calculate_average_grades(Groups2,ListGrades,FinalGroups,1,NumberGroups,AverageGrades1),
68                          general_distance_between_grades(AverageGrades1,[],DiffGrades1),
69                          maximum(Max1,DiffGrades1),
70                          Metric1 #= 3* Max1 / 4 + Acc1 / 4,
71                          TotalMetrics #= Metric1+Metric,
72                          append(Groups1,Groups2,TotalGroups),
73                          labeling([minimize(TotalMetrics),time_out(70000,_)],TotalGroups),
74                          write('Repetions > '),
75                          write(Acc1),nl,write('Repetions > '),
76                          write(Acc),nl,
77                          statistics(walltime, [End,_]),
78                            Time is End - Start,
79                          fd_statistics,
80                          output(ListGrades,X,Groups1,AverageGrades,Groups2,AverageGrades1,Time).
81
82  distance_between_grades([A,B],CurrList,FinalList):- Diff #= abs(B-A),
83                                                      append(CurrList,[Diff],CurrList1),
84                                                      FinalList = CurrList1.
85  distance_between_grades([A,B,C|T],CurrList,FinalList):-
86                                                          Diff #= abs(B-A),
87                                                          append(CurrList,[Diff],CurrList1),
88                                                          distance_between_grades([A,C|T],CurrList1,FinalList).
89
90  general_distance_between_grades([_],FinalList,FinalList).
91
92  general_distance_between_grades([H|T],CurrList,FinalList):- distance_between_grades([H|T],[],FinalList1),
93                                                              append(CurrList,FinalList1,CurrList1),
94                                                              general_distance_between_grades(T,CurrList1,FinalList).
```

```
96
97  list_global(CurrPosition,NumberGroups,NumberElements,FinalList,FinalList,ListGroups,ListGroups):- CurrPosition >= NumberGroups.
98  list_global(CurrPosition,NumberGroups,NumberElements,CurrList,FinalList,CurrentGroups,FinalGroups):-
99                                                  CurrPosition < NumberGroups,
100                                                 CurrPosition1 is CurrPosition+1,
101                                                 append(CurrList,[CurrPosition1-NumberElements],CurrList1),
102                                                 append(CurrentGroups,[NumberElements],CurrentGroups1),
103                                                 list_global(CurrPosition1,NumberGroups,NumberElements,CurrList1,FinalList,CurrentGroups1,FinalGroups).
104
105
106
107 calculate_average_grades(List,Grades,GroupElements,CurrentGroup,NumberGroups,[]).
108 calculate_average_grades(List,Grades,[G1|G],CurrentGroup,NumberGroups,[H|T]):-
109                                                                 sum_grades(List,Grades,CurrentGroup,Sum),
110                                                                 H #= Sum / G1,
111                                                                 CurrentGroup1 #= CurrentGroup+1,
112                                                                 calculate_average_grades(List,Grades,G,CurrentGroup1,NumberGroups,T).
113
114 sum_grades([A],[G1],Group,SumGrades):- (A #= Group) #<=> Bin,
115                                        SumGrades #= G1 * Bin.
116
117 sum_grades([A,B|T],[G1,G2|G],Group,SumGrades):-     sum_grades([B|T],[G2|G],Group,Sum),
118                                                     (A #= Group) #<=> Bin,
119                                                     SumGrades #= Sum + G1 * Bin.
120
121
122 count_repetitions([_],0,[_]).
123
124 count_repetitions([H|T],Acc,[X1|X]):- count_repetitions(T,Acc2,X),
125                                       repetitions([H|T],[X1|X],Acc1),
126                                       Acc #= Acc2+Acc1.
127
128 repetitions([A,B],[X1,X2],Acc):- (X1 #= X2 #/\ A #=B)#<=>Bin, Acc #= Bin.
129 repetitions([A,B,C|T],[X1,X2,X3|X],Acc):-
130                                         repetitions([A,C|T],[X1,X3|X],Acc1),
131                                         (X1 #= X2 #/\ A #=B)#<=>Bin,
132                                         Acc #= Acc1 + Bin.
133
134
135
136 calculate_number_groups(Students,NumberGroups4,NumberGroups3,NumberThemes):-
137                 Values=[NumberGroups4,NumberGroups3],
138                 NumberGroups3 in 0..Students,
139                 NumberGroups4 in 0..Students,
140                 Students #= NumberGroups3 * 3 + NumberGroups4 * 4,
141                 NumberThemes #= NumberGroups3+NumberGroups4,
142                 labeling([],Values).
```