Resolução de Problema de Otimização - Grupos de Trabalho

Francisco Filipe[1] up201604601@fe.up.pt e Pedro Fernandes[1] up201603846@fe.up.pt

FEUP-PLOG, Turma 3MIEIC1, Grupo Grupos de Trabalho_2

Resumo Este artigo foi desenvolvido no âmbito da unidade curricular de Programação em Lógica e tem como objetivo principal a resolução de um problema de otimização através de restrições. O problema de otimização presente é o tema Grupos de Trabalho e tem como finalidade principal gerar os grupos de trabalho para dois projetos distintos de uma unidade curricular. Para este processo é necessário ter em consideração a proximidade de médias dos seus elementos, o facto de estes não terem trabalhado juntos anteriormente e, do primeiro para o segundo projeto, não existirem pares de estudantes no mesmo grupo.

Keywords: PLR · SICStus · Grupos de Trabalho.

1 Introdução

Este artigo foi desenvolvido no âmbito da unidade curricular de Programação em Lógica e tem como objetivo descrever um problema de decisão ou otimização em Prolog utilizando restrições. Como tema de trabalho optámos por um problema de otimização, denominado Grupos de Trabalho.

O tema escolhido tem como objetivo gerar grupos de trabalho para 2 projetos de uma unidade curricular tendo em consideração que: a diferença de médias dos grupos deve ser o mais próxima possível; alunos que tenham trabalhado juntos antes não **devem** ficar no mesmo grupo; alunos que tenham trabalhado juntos no primeiro projeto não **podem** trabalhar juntos no segundo.

Este artigo segue a seguinte estrutura:

- Descrição do Problema: descrição detalhada do problema de otimização em questão.
- Abordagem: descrição da modelação do problema como um PLR, seguindo a seguinte estrutura:
  - Variáveis de Decisão: descrição das variáveis de decisão e respetivos domínios.
  - Restrições: descrição das restrições rígidas e flexíveis do problema e a sua implementação.

    - Função de Avaliação: descrição da forma de avaliar a solução obtida e a sua implementação utilizando o SICStus Prolog.
    - Estratégia de Pesquisa: descrição da estratégia de *labeling* implementada, nomeadamente no que diz respeito à ordenação de variáveis e valores.
- Visualização da Solução: explicação dos predicados que permitem visualizar a solução em modo de texto.
- Resultados: análise dos resultados e eficiência do programa para diferentes tipos de *input*.
- Conclusões e Trabalho Futuro: conclusões retiradas deste projeto, resultados obtidos, vantagens e limitações da solução proposta, aspetos a melhorar.
- Anexo: código fonte, ficheiros de dados e resultados.

2 Descrição do Problema

Tal como está descrito no enunciado, o tema Grupos de Trabalho consiste no seguinte:

> O professor X pretende formar grupos de trabalho para os dois projetos da sua unidade curricular, de acordo com alguns objetivos pessoais. Ele pretende evitar colocar no mesmo grupo pessoas que já tenham trabalhado juntas no passado, embora isso possa acontecer eventualmente (ele tem informação sobre quem trabalhou com quem em outras UCs passadas).
> O professor também quer garantir que não existem nunca repetições de grupos entre o primeiro e segundo projeto da sua UC (ou seja, dois estudantes que trabalham juntos no primeiro projeto não podem trabalhar juntos no segundo).
> Por outro lado, ele pretende equilibrar os grupos, fazendo com que as médias dos membros de cada grupo sejam o mais próximas possível.
> Os grupos devem ter entre 3 e 4 elementos, sendo o número de temas disponíveis (correspondente ao número de grupos) fixado à partida, de acordo com o número de estudantes da UC.

Pela análise do enunciado acima presente podem ser identificadas duas componentes principais:

- **Objetivo:** Gerar grupos de trabalho para dois projetos de uma unidade curricular.
- **Restrições:** Pela análise do enunciado podem-se identificar as seguintes restrições:
    - **Evitar** que estudantes que já tenham trabalhado juntos anteriormente fiquem juntos no mesmo grupo.
    - **Não permitir** que estudantes que trabalhem juntos no primeiro projeto o possam fazer no segundo projeto.
    - Para pertencerem ao mesmo grupo, os estudantes devem ter médias o mais próximas possível.

3 Abordagem

3.1 Variáveis de Decisão

As variáveis de decisão estão agrupadas em duas listas, uma para cada projeto da unidade curricular, com tamanho igual ao da lista de estudantes fornecida. Desta forma, cada variável é relacionada com o estudante de índice igual, indicando o grupo a que ele pertence.

O domínio de cada variável deverá ser um intervalo entre 1, e o número máximo de grupos possível, que é o número de estudantes dividido pelo tamanho máximo de cada grupo.

3.2 Restrições

O programa começa por aplicar a restrição `constrain_size/2` , que é responsável por fazer o agrupamento básico dos estudantes. Como um dos argumentos do programa é um intervalo representando o tamanho mínimo e máximo dos grupos, então é preciso garantir que cada variável que conste na lista ocorre um número de vezes que pertença a esse intervalo. Para tal, é usado o predicado `count/4`.

De seguida, é chamado o predicado `constrain_GPA/5` que, para cada grupo, procura as médias dos seus elementos, encontra o máximo e o mínimo, e guarda a diferença entre estes dois valores numa lista, aproveitando-se dos predicados `maximum/2` e `minimum/2`. A lista resultante é fornecida ao predicado `sum/3` para obter a soma.

O próximo passo é chamar `constrain_worked_before/4`, que 'retorna' uma lista que, quando somada, indica o número de pares de estudantes que estão no mesmo grupo e já trabalharam antes. A estratégia utilizada é percorrer a lista que contém os pares de estudantes que já trabalharam antes, e pesquisá-los na lista de variáveis do projeto. Se essas variáveis forem iguais, condição que é verificada através de `nvalue/2` e de uma restrição materializada, então é acrescentado o valor 1 à lista de retorno, ou então 0. Tal como no caso anterior, é obtida a soma dos elementos desta lista, cujo objetivo é explicado nas secções seguintes.

Finalmente, é aplicada a restrição `constrain_worked_first_project\5`, encarregue de não permitir que existam grupos repetidos entre o primeiro e o segundo projeto. Assim, deve encontrar os elementos do mesmo grupo na primeira lista de variáveis, e recorrendo a uma restrição materializável cria uma lista de igual tamanho, contendo 0, ou as variáveis correspondentes a um dado grupo na lista de variáveis do segundo projeto. Assim, deve-se assegurar com `nvalue/2` que o número de valores distintos nessa lista é igual ao tamanho do grupo, mais um. Exemplificando: Seja, após *labeling*, Proj1Vars = [1,2,2] e Proj2Vars = [1,2,3]. Esta distribuição está correta pois, para GroupID = 1, então a lista auxiliar será [1, 0, 0], onde o tamanho do grupo é 1. Por isso, o numero de valores distintos é 2. Para GroupID = 2, a lista auxiliar é [0,2,3], o tamanho do grupo é 2, e o número de valores distintos é 3. A nosso ver, esta forma de implementar a restrição está

correta, mas infelizmente não funciona para todos os casos. Por falta de tempo, não foi possível encontrar o problema.

3.3 Função de Avaliação

O problema engloba dois casos de otimização que são tratados numa função de avaliação. Para cada trabalho deve-se tentar minimizar o número de pares de estudantes que já trabalharam antes e ficam novamente no mesmo grupo, assim como a diferença de médias dentro de cada grupo. Para tal, são aproveitadas as somas referidas anteriormente, e são todas elas somadas numa única variável de decisão, denominada **Min**.

3.4 Estratégia de Pesquisa

A estratégia de etiquetagem consiste em minimizar a variável **Min**, de forma a encontrar a solução que melhor representa as otimizações pretendidas.

4 Visualização da Solução

O programa permite duas opções para a entrada e saída de dados. O utilizador pode escolher entre fornecer ficheiros de texto formatados devidamente, caso em que recebe noutros dois ficheiros os grupos respetivos, ou então fornecer as listas através de argumentos do programa, e visualizar as soluções no terminal do SICStus.

Ficheiros Visto que o predicado `read/2` suporta leitura de átomos através de ficheiros, pode ser usado para ler os dados. Assim, o predicado `read_files/9` recebe os ficheiros, e 'retorna' em listas os dados que lá estão contidos. De notar que os nomes dos temas não devem conter espaços.

Para escrever o resultado nos ficheiros, é chamado o predicado `write_files/4`, que coloca em dois ficheiros os grupos de cada projeto. Previamente, é chamado o predicado `get_groups/6` que é responsável por extrair os grupos das listas de variáveis retornadas pelo predicado principal.

Para facilitar a leitura dos ficheiros, o primeiro argumento é o diretório pai dos ficheiros de entrada, e os restantes são apenas os nomes.

Exemplo:

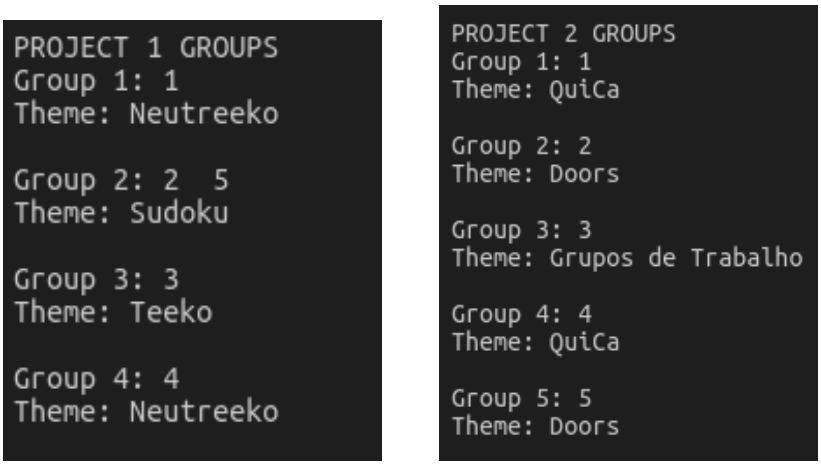
groups_files(' /Documentos/PLOG1819/T2/src/', 'students.txt', 'previousUCsInfo.txt', 'proj1Themes.txt', 'proj2Themes.txt', [1,2]).

Terminal Neste caso os dados são passados através de argumentos, e os resultados são mostrados no ecrã com recurso ao predicado `write_ter/4`, cuja funcionalidade é semelhante à sua contra-parte.

Exemplo:

groups_ter([1,2,3,4,5], [20,12,16,11,12], [[1,2]], [1,3], ['Neutreeko', 'Sudoku', 'Teeko'], ['QuiCa', 'Doors', 'Grupos de Trabalho']).

O resultado visualizável no terminal é o seguinte:



(a) Grupos do projeto 1 (b) Grupos do projeto 2

Figura 1: *Output* no terminal após chamada groups_ter/6

5 Resultados

Para se poderem tirar conclusões dos resultados obtidos foram medidos o tempo de resolução, o número de retrocessos e o número de restrições criadas. Como já foi dito acima, a implementação feita para gerar os grupos de dois trabalhos nem sempre é bem sucedida e, por esse motivo, as seguintes medições terão como base as restrições necessárias para gerar os grupos de um só trabalho. Seguem-se as condições de teste e as respetivas conclusões:

Tabela 1: Variação da duração, retrocessos e restrições criadas em função do número de estudantes.

Tamanho dos Grupos: [1,2]			
Número de Estudantes	Tempo(s)	Retrocessos	Restrições Criadas
3	0.0	9	172
4	0.0	51	282
5	0.01	255	385
6	0.21	4071	586
7	5.53	101573	780
8	114.85	1862304	1002

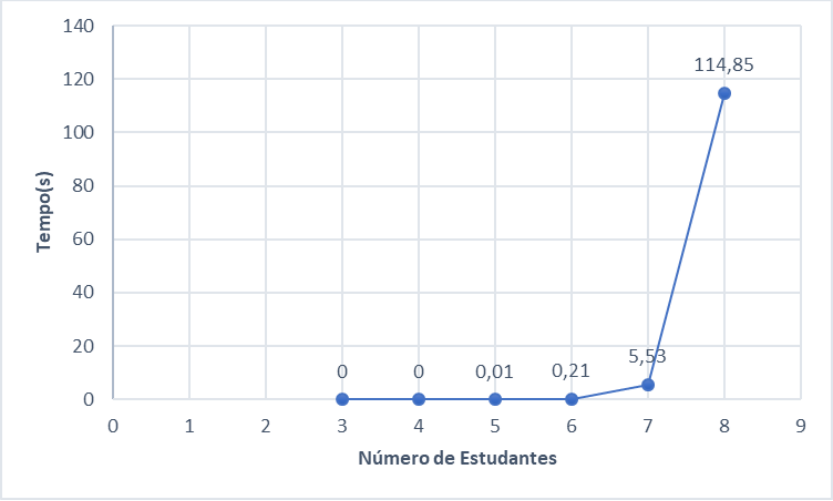


Figura 2: Variação da duração, retrocessos e restrições criadas em função do número de pessoas.

Tabela 2: Variação da duração, retrocessos e restrições criadas em função do tamanho dos grupos.

Número de Estudantes: 8				
Tamanho dos Grupos				
Min	Max	Tempo(s)	Retrocessos	Restrições Criadas
1	2	114.85	1862304	1002
1	3	220.76	3132248	1002
2	3	0.07	1289	506
2	4	0.09	1664	506
3	4	0.0	7	258
3	5	0.0	8	258

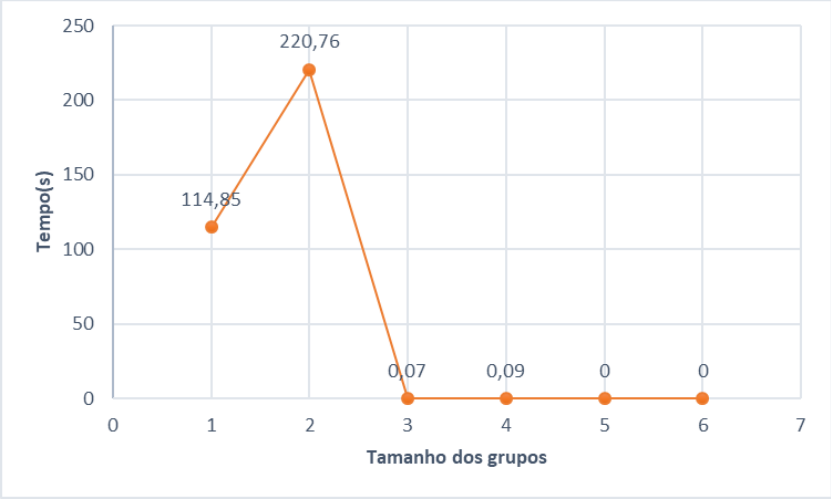


Figura 3: Variação da duração, retrocessos e restrições criadas em função do tamanho dos grupos.

Como é possível verificar pelos resultados anteriores o número de estudantes afeta diretamente a performance do programa. Quanto maior for o número de estudantes, maior é o tempo necessário para gerar os grupos de trabalho.

Quanto ao tamanho dos grupos, é possível observar que quantos mais elementos os grupos tiverem, melhor será a performance do programa. Isto justifica-se pelo facto de que o intervalo do número de grupos de trabalho (calculado durante a execução do programa) diminui e, por conseguinte, diminui também o número de possíveis soluções a testar.

6 Conclusões e Trabalho Futuro

Admitidamente, tivemos bastantes dificuldades ao longo do desenvolvimento do projeto. Deliniámos um raciocínio para resolver as restrições/otimizações pedidas, mas não conseguimos imediatamente colocá-lo em prática. Além disso, foram surgindo outros problemas que nos atrasaram ainda mais. Ainda assim, consideramos que conseguimos cumprir os principais objetivos propostos.

De facto, o programa poderia ser melhorado em várias vertentes, como por exemplo a eficiência. A biblioteca *clpfd* é bastante poderosa, e poderíamos ter usado predicados mais avançados para resolver as restrições.

7 Anexos

7.1 Código Fonte

groups.pl

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

:- consult('io.pl').


reset_timer :- statistics(walltime,_).
print_time :-
        statistics(walltime,[_,T]),
        TS is ((T//10)*10)/1000,
        nl, write('Time: '), write(TS), write('s'), nl, nl.


if_then_else(C, I, _):- C, !, I.
if_then_else(_, _, E):- E.


groups_ter(Students, GPAs, PreviousUCsInfo, GroupSize, Proj1Themes, Proj2Themes):-
    solve(Students, GPAs, PreviousUCsInfo, GroupSize, Proj1Vars, Proj2Vars),
    max_member(NumGroups1, Proj1Vars),
    max_member(NumGroups2, Proj2Vars),
    get_groups(Students, Proj1Vars, [], Proj1Groups, 1, NumGroups1),
    get_groups(Students, Proj2Vars, [], Proj2Groups, 1, NumGroups2),
    write('\nPROJECT 1 GROUPS\n'),
    length(Proj1Themes, Proj1ThemesLen),
    length(Proj2Themes, Proj2ThemesLen),
    write_ter(Proj1Groups, Proj1Themes, Proj1ThemesLen, 0),
```

```
    write('PROJECT 2 GROUPS\n'),
    write_ter(Proj2Groups, Proj2Themes, Proj2ThemesLen, 0), !.

groups_files(CWD, StudentsFile, PreviousUCsInfoFile, Proj1ThemesFile,
Proj2ThemesFile, GroupSize):-
    current_directory(_, CWD),
    read_files(StudentsFile, PreviousUCsInfoFile, Proj1ThemesFile, Proj2ThemesFile,
    Students, GPAs, PreviousUCsInfo, Proj1Themes, Proj2Themes),
    solve(Students, GPAs, PreviousUCsInfo, GroupSize, Proj1Vars, Proj2Vars),
    max_member(NumGroups1, Proj1Vars),
    max_member(NumGroups2, Proj2Vars),
    get_groups(Students, Proj1Vars, [], Proj1Groups, 1, NumGroups1),
    get_groups(Students, Proj2Vars, [], Proj2Groups, 1, NumGroups2),
    write_files(Proj1Groups, Proj2Groups, Proj1Themes, Proj2Themes), !.

get_groups_aux(_, [], Group, Group):-!.
get_groups_aux(Students, [H | T], CurrGroup, Group):-
    nth1(H, Students, Student),
    append(CurrGroup, [Student], NextGroup),
    get_groups_aux(Students, T, NextGroup, Group).

get_groups(_, _, Groups, Groups, Num, Max):- Num > Max, !.
get_groups(Students, ProjVars, CurrProjGroups, ProjGroups, Num, Max):-
    findall(X, nth1(X, ProjVars, Num), List),
    get_groups_aux(Students, List, [], Group),
    append(CurrProjGroups, [Group], NextProjGroups),
    NextNum is Num + 1,
    get_groups(Students, ProjVars, NextProjGroups, ProjGroups, NextNum, Max).

solve(Students, GPAs, PreviousUCsInfo, [MinSize, MaxSize], Proj1Vars, Proj2Vars):-

    %create list of Vars with the same length of students
    length(Students, NumStudents),
    length(Proj1Vars, NumStudents),
    length(Proj2Vars, NumStudents),

    %calculate maximum and minimum number of groups
    MaxNumGroups is NumStudents div MinSize,
    MinNumGroupsMod is NumStudents mod MaxSize,
    if_then_else(
                    (MinNumGroupsMod = 0),
                        (MinNumGroups is NumStudents div MaxSize),
                        (MinNumGroups is (NumStudents div MaxSize) + 1)
                ),
```

```
    %domain variables' domain
    domain(Proj1Vars, 1, MaxNumGroups),
    domain(Proj2Vars, 1, MaxNumGroups),

    %constrain group size
    get_elems(Proj1Vars, [MinSize, MaxSize], MaxNumGroups, 1,Proj1GroupElems),
    get_elems(Proj2Vars, [MinSize, MaxSize], MaxNumGroups, 1,Proj2GroupElems),
    constrain_size(Proj1GroupElems,[MinSize,MaxSize]),
    constrain_size(Proj2GroupElems,[MinSize,MaxSize]),
    count(0,Proj1GroupElems,#=, Zeros1),
    count(0,Proj2GroupElems,#=, Zeros2),
    maximum(Max1, Proj1Vars),
    maximum(Max2, Proj2Vars),
    Max1 #= MaxNumGroups - Zeros1,
    Max2 #= MaxNumGroups - Zeros2,

    %constrain GPA
    constrain_GPA(GPAs, Proj1Vars, MaxNumGroups, 1, GPADiffs1),
    constrain_GPA(GPAs, Proj2Vars, MaxNumGroups, 1, GPADiffs2),
    sum(GPADiffs1, #=, SumGPADiffs1),
    sum(GPADiffs2, #=, SumGPADiffs2),

    %constrain Worked Before
    constrain_worked_before(Students, PreviousUCsInfo, Proj1Vars, WorkedBefore1),
    sum(WorkedBefore1, #=, SumWorkedBefore1),
    constrain_worked_before(Students, PreviousUCsInfo, Proj2Vars, WorkedBefore2),
    sum(WorkedBefore2, #=, SumWorkedBefore2),

    %different from first project
    constrain_worked_first_project(Proj1Vars, Proj2Vars, MaxNumGroups, 1),

    %minimize variable
    Min #= SumGPADiffs1 + SumWorkedBefore1 + SumGPADiffs2 + SumWorkedBefore2,

    %labeling

    append(Proj1Vars, Proj2Vars, AllVars),
    reset_timer,
    labeling([minimize(Min)], AllVars),

    print_time,
    fd_statistics.


solve_only_first(Students, GPAs, PreviousUCsInfo, [MinSize, MaxSize], Vars):-
```

```
    %create list of Vars with the same length of students
    length(Students, NumStudents),
    length(Vars, NumStudents),

    %calculate the maximum and minimum number of groups of the given input
    MaxNumGroups is NumStudents div MinSize,
    MinNumGroupsMod is NumStudents mod MaxSize,
    if_then_else(
                    (MinNumGroupsMod = 0),
                        (MinNumGroups is NumStudents div MaxSize),
                        (MinNumGroups is (NumStudents div MaxSize) + 1)
                ),

    %domain variables' domain
    domain(Vars, 1, MaxNumGroups),

    %constrain group size
    get_elems(Vars, [MinSize, MaxSize], MaxNumGroups, 1, GroupElems),
    constrain_size(GroupElems, [MinSize,MaxSize]),
    count(0,GroupElems,#=, Zeros),
    maximum(Max, Vars),
    Max #= MaxNumGroups - Zeros,

    %constrain GPA
    constrain_GPA(GPAs, Vars, MaxNumGroups, 1, GPADiffs),
    sum(GPADiffs, #=, SumGPADiffs),

    %constrain Worked Before
    constrain_worked_before(Students, PreviousUCsInfo, Vars, WorkedBefore),
    sum(WorkedBefore, #=, SumWorkedBefore),

    %value to minimize
    Min #= SumGPADiffs + SumWorkedBefore,

    %labeling
    reset_timer,
    labeling([minimize(Min),up], Vars),
    print_time,
    fd_statistics.


constrain_size([],_).
constrain_size([H | T],[MinSize,MaxSize]):-
    (H #>= MinSize #/\ H #=< MaxSize) #\/ (H #= 0),
```

```
    constrain_size( T, [MinSize,MaxSize]).


get_elems(_, _, NumGroups,Num,[]):- Num > NumGroups, !.
get_elems(Vars, [MinSize, MaxSize], NumGroups, Num,[Times | T]):-
    count(Num, Vars, #=, Times),
    NextNum is Num + 1,
    get_elems(Vars, [MinSize, MaxSize], NumGroups, NextNum, T).


getGPAs([], [],_,[]).
getGPAs([GPA_A | GPAs], [H|Vars], GroupID,[GroupGPAsH | GroupGPAsT]):-
    H #= GroupID #<=> B,
    GroupGPAsH #= B * GPA_A,
    getGPAs(GPAs, Vars, GroupID, GroupGPAsT).


changeZero([], _, New, New):- !.
changeZero([Head|Tail], Max, Temp, New):-
    (Head #= 0 #/\ Element #= Max)
    #\/
    (Head #\= 0 #/\ Element #= Head),
    append(Temp,[Element],Next),
    changeZero(Tail, Max, Next, New).


constrain_GPA(_, _, NumGroups, Num, []):- Num > NumGroups, !.
constrain_GPA(GPAs, Vars, NumGroups, Num, [DiffsH | DiffsT]):-
    getGPAs(GPAs, Vars, Num, GroupGPAs),
    maximum(MaxGPA, GroupGPAs),
    changeZero(GroupGPAs, MaxGPA, [], NewGroupGPAs),
    minimum(MinGPA, NewGroupGPAs),
    DiffsH #= MaxGPA - MinGPA,
    NextNum is Num + 1,
    constrain_GPA(GPAs, Vars, NumGroups, NextNum, DiffsT).


getGroupIDs([],[],_,[]).
getGroupIDs([CS | RS], [_ | RV], [S1,S2],RestID):-
    CS \= S1, CS \= S2, !,
    getGroupIDs( RS,  RV, [S1, S2], RestID).
getGroupIDs([CS | RS], [CV | RV], [S1,S2],[CurrID|RestID]):-
    (CS = S1 ; CS = S2) , !,
    CurrID #= CV,
    getGroupIDs( RS,  RV, [S1, S2], RestID).
```

```
constrain_worked_before(_, [], _, []).
constrain_worked_before(Students, [CurrPair | RestPairs], Vars, [PairWT | RestWT]):-
    getGroupIDs(Students, Vars,CurrPair, CurrIDs),
    nvalue(DistinctMembers,CurrIDs),
    DistinctMembers #= 1 #<=> PairWT,
    constrain_worked_before(Students,  RestPairs, Vars,  RestWT).


get_group([], _, _, [],1).
get_group([H1 | T1], [H2 | T2], Num, [Elem | Rest],GL):-
    H1 #= Num #<=> B,
    Elem #= B * H2,
    GL #= B + OldGL,
    get_group(T1, T2, Num, Rest,OldGL).


constrain_worked_first_project(_, _, NumGroups, Num):- Num > NumGroups,!.
constrain_worked_first_project(Proj1Vars, Proj2Vars, NumGroups, Num):-
    get_group(Proj1Vars, Proj2Vars, Num, SameGroup,GroupLen),
    nvalue(DistinctMembers,SameGroup),
    DistinctMembers #= GroupLen,
    NextNum is Num + 1,
    constrain_worked_first_project(Proj1Vars, Proj2Vars, NumGroups, NextNum).
}
```

io.pl

```
read_files(StudentsFile, PreviousUCsInfoFile, Proj1ThemesFile, Proj2ThemesFile,
Students, GPAs, PreviousUCsInfo, Proj1Themes, Proj2Themes):-
    open(StudentsFile, read, Str),
    read_students_file(Str, Students, GPAs),
    close(Str),
    open(PreviousUCsInfoFile, read, Str2),
    read_previous_ucs_file(Str2, PreviousUCsInfo),
    open(Proj1ThemesFile, read, Str3),
    read_themes_file(Str3, Proj1Themes),
    open(Proj2ThemesFile, read, Str4),
    read_themes_file(Str4, Proj2Themes),
    close(Str2), !.

read_themes_file(Stream,[]) :-
    at_end_of_stream(Stream), !.

read_themes_file(Stream,[X|L]) :-
```

```
    \+ at_end_of_stream(Stream),
    read(Stream,X),
    read_themes_file(Stream,L).

read_previous_ucs_file(Stream,[]) :-
    at_end_of_stream(Stream), !.

read_previous_ucs_file(Stream,[[X, Y]|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X),
    read(Stream,Y),
    read_previous_ucs_file(Stream,L).

read_students_file(Stream, [], []) :-
    at_end_of_stream(Stream), !.

read_students_file(Stream,[S|Students], [GPA | GPAs]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,S),
    read(Stream,GPA),
    read_students_file(Stream,Students, GPAs).

write_files(Proj1Groups, Proj2Groups, Proj1Themes, Proj2Themes):-
    open('proj1groups.txt', write, Str),
    length(Proj1Themes, Proj1ThemesLen),
    write(Str, 'PROJECT 1 GROUPS\n'),
    write_groups(Str, Proj1Groups, 0, Proj1Themes, Proj1ThemesLen),
    close(Str),
    open('proj2groups.txt', write, Str2),
    length(Proj2Themes, Proj2ThemesLen),
    write(Str2, 'PROJECT 2 GROUPS\n'),
    write_groups(Str2, Proj2Groups, 0, Proj2Themes, Proj2ThemesLen),
    close(Str2), !.

write_group(Stream, []):-write(Stream, '\n'), !.
write_group(Stream, [H | T]):-
    write(Stream, H),
    write(Stream, '  '),
    write_group(Stream, T).

write_groups(Stream, [], _, _, _) :- write(Stream, '\n'), !.
write_groups(Stream, [H | T], Count, Themes, ThemesLen):-
    GroupNum is Count + 1,
    format(Stream, 'Group ~d: ', [GroupNum]),
    write_group(Stream, H),
```

```
    Index is Count mod ThemesLen,
    nth0(Index, Themes, Theme),
    format(Stream, 'Theme: ~s~n~n', [Theme]),
    NextCount is Count + 1,
    write_groups(Stream, T, NextCount, Themes, ThemesLen).

write_ter_group([]):- write('\n'), !.
write_ter_group([H | T]):-
    write(H),
    write('  '),
    write_ter_group(T).

write_ter([], _, _,_) :- write('\n'), !.
write_ter([H | T], Themes, ThemesLen, Count):-
    GroupNum is Count + 1,
    format('Group ~d: ', [GroupNum]),
    write_ter_group(H),
    NextCount is Count + 1,
    Index is Count mod ThemesLen,
    nth0(Index, Themes, Theme),
    format('Theme: ~s~n~n', [Theme]),
    write_ter(T, Themes, ThemesLen, NextCount).
}
```

7.2 Ficheiros de Dados

Seguem um exemplo de ficheiros de teste, que devem ser usados como argumentos do predicado `groups_files/6`.

students.txt
201603846. 165.
201604601. 164.
201606279. 124.
201633611. 179.
201614133. 198.

previousUCsInfo.txt
201603846. 201604601.
201603846. 201606279.

proj1Themes.txt
neutreeko.
sudoku.
teeko.

proj2Themes.txt
quiCa.
doors.

grupos_de_trabalho.

Os resultados são retornados nos ficheiros proj1groups.txt e proj2groups.txt