**3.ABORDAGEM**

Na resolução deste problema na linguagem Prolog foi utilizada uma lista para representar a distribuição de público inicial *(****InputGroups****),* pois para as pessoas do mesmo grupo ficarem em lugares contíguos não é necessário ter em consideração as restantes filas.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

Na posição correspondente ao número do lugar na lista *InputGroups*, está o número do grupo da pessoa inicialmente nesse lugar. Assim, se a tabela 1? representar o público, os números em cada célula o número de cada lugar e as cores os diferentes grupos, a lista tida em consideração para resolver o problema será: *InputGroups=[1,1,2,2,3,3,3,2,2,1,3,3].*

**3.1 VARIÁVEIS DE DECISÂO**

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 10 |
| 4 | 3 | 8 |
| 9 | 5 | 6 |
| 7 | 11 | 12 |

A solução do problema vem na forma de 2 listas que representam a *distribuição de público final*: uma para representar a distribuição dos grupos (***OutputGroups***) e outra que nos diz qual o lugar inicial da pessoa atualmente naquele lugar (***OutputIndexs***) Assim a solução deste problema tendo em conta a distribuição inicial a cima seria: *OutputGroups=[1,1,1,2,2,2,2,3,3,3,3,3]* e *OutputIndexs=[1,2,10,4,3,8,9,5,6,7,11,12].*

Tanto o tamanho da lista *OutputGroups*, como o da lista *OutputIndexs* é igual ao tamanho da lista *InputGroups*. No que toca ao domínio, o da lista *OutputGroups* é de 1 até ao número de grupos e da lista *OutputIndexs* é de 1 até ao número de pessoas.

**3.2 RESTRIÇÕES**

***1. Os elementos da lista OutputIndexs têm que ser todos distintos.***

Como cada posição da lista *OutputIndexs* é constituída pelo número do lugar inicial da pessoa atualmente nessa posição e os números de lugares são únicos foi chamado o seguinte predicado de restrição *all\_distinct(OutputIndexs).*

***2. Na lista OutputGroups o elemento na posição i tem que ser o elemento que está na posição j da InputGroups, sendo a posição j o elemento na posição i da lista OutputIndexs* (*OutputGroups[i] = InputGroups[j] AND j = OutputIndexs[i])***

É necessário garantir que os grupos estão corretamente atribuídos a cada lugar na solução do problema. Para isso foi utilizado o predicado *get\_groups(InputGroups, OutputIndexs, OutputGroups).*

get\_groups(\_,[],[]).

get\_groups(InputGroups, [OutputIndexsH|OutputIndexsT], [OutputGroupsH|OutputGroupsT]):-

element(OutputIndexsH, InputGroups, OutputGroupsH),

get\_groups(InputGroups, OutputIndexsT, OutputGroupsT).

***3. A distância de um elemento até ao próximo do mesmo grupo (caso exista) tem que ser 0***

Como o objetivo do problema é juntar as pessoas do mesmo grupo, impõe-se que a distância de um elemento de um grupo até ao próximo elemento do mesmo grupo caso este exista, para isso é usado o predicado *approximate(OutputGroups)*

approximate([]).

approximate([OutputGroupsH|OutputGroupsT]):-

get\_distance(Distance, NotUnique, OutputGroupsT, OutputGroupsH),

NotUnique #=> Distance #=0,

approximate(OutputGroupsT).

O predicado *get\_distance,* com a ajuda de um autómato, retorna em *Distance* a distância até ao primeiro elemento com o valor *OutputGroupsH* e caso este exista o predicado retorna em *NotUnique* o valor 1, caso contrário o valor 0.

**3.3 FUNÇÃO DE AVALIAÇÃO**

Este problema é um problema de otimização, ou seja, o objetivo não é somente encontrar uma solução, mas sim encontrar a melhor solução.

Neste caso, a melhor solução é a que implica:

1) menor distância percorrida por cada pessoa na mudança de lugar

2) menor número de trocas

Assim, para cada lugar na distribuição final é calculada a diferença desse mesmo lugar e do lugar inicial da pessoa, isto é possível pois a lista *OutputGroups* guarda o lugar inicial na posição final da pessoa. Para isso é usado o seguinte predicado *fill\_differences(OutputIndexs,OutputIndexs, Differences).*

fill\_differences(\_,[],[]).

fill\_differences(OutputIndexs, [OutputIndexsH|OutputIndexsT], [DifferencesH|DifferencesT]):-

element(OutputPos, OutputIndexs, OutputIndexsH),

DifferencesH #= abs(OutputPos-OutputIndexsH),

fill\_differences(OutputIndexs, OutputIndexsT, DifferencesT).

Depois da lista *Differences* ter sido obtida, somam-se todos os elementos desta lista obtendo-se a variável *TotalDifference*

Seguidamente conta-se o número de elementos não zero da lista *Differences*, ou seja, contam-se os elementos que se moveram com a ajuda do predicado *get\_changes(NumOfChanges, Differences),* que usa um autómato para retornar o contador desejado em *NumOfChanges.*

Assim, para minimizar com igual peso o deslocamento das pessoas e o número de deslocamentos significa otimizar uma função (linear) objetivo:

***Minimizar F = 1 \* TotalDifference + 1 \* NumOfChanges***

**3.4 ESTRATÉGIA DE PESQUISA**

**4 RESULTADOS**