Família Numerosa Title

Ricardo Jorge de Araújo Ferreira[200305418]
up200305418@fe.up.pt

Faculdade de Engenharia da Universidade do Porto, Portugal
FEUP - PLOG, Turma, 3MIEIC06,Familia Numerosa_3

Abstract. Neste trabalho foi desenvolvida uma solução para um problema de alocação de horários e de rotas, considerando um conjunto de restrições. O problema abordado no trabalho irá concretamente resolver o problema de alocar pessoas a diferentes actividades e calcular a respectiva rota para levar e recolher cada pessoa. Este problema será resolvido como um problema de optimização e satisfação de restrições. Para tal, será utilizada a linguagem de programação *PROLOG*, e o sistema de desenvolvimento *SICSTUS*, tirando partido da biblioteca *clpfd*, *Constraint Logic Programming over Finite Domains*.

Keywords: Programação por restrições · Restrições em domínios finitos · PROLOG · Scheduling · Routing.

1 Introdução

1.1 Objectivos e motivação

Para este trabalho, é esperado resolver o problema de Famílias Numerosas, em que um pai, apesar do elevado número de filhos, pretende que todos pratiquem uma actividade extracurricular, nomeadamente música ou desporto. A resolução do problema deve ter em conta um conjunto de restrições e optimizações, abordadas em detalhe na Secção 3, para conseguir alocar cada filho a uma actividade diferente e garantir que é possível levar e recolher cada filho a cada local das actividades.

Este problema pode ser divido em dois sub-problemas, a alocação de horário, *Scheduling*, e o cálculo de rotas, *routing*. Ambos os problemas são bastante estudados em ciência de computação e várias soluções são implementadas em PROLOG ou em linguagens semelhantes que permitam programação por restrições. Tanto a alocação de horários como o cálculo de rotas têm imensas aplicações práticas, como alocação de turnos em fábricas, [1], cálculo de circuitos para placas electrónicas, rotas para entrega de encomendas, cálculo de horários e rotas de comboios [2] entre outras [3].

Na referência [2], é apresentado um problema, mais complexo que o estudado neste trabalho, mas igualmente de alocação de horário e cálculo de rotas, neste caso, aplicado a comboios e funcionários dos caminhos de ferro. O problema é resolvido, identificando todos os nós das diversas linhas de comboio e

aplicando restrições simples, em *PROLOG* com a biblioteca *clpfd*, tais como, variável A maior do que X e restrições mais complexas envolvendo condicionais e implicações, entre outros. Uma solução semelhante será apresentada neste trabalho.

1.2 Estrutura do artigo

Na Secção 2, é apresentado o problema que se pretende resolver e as ferramentas a utilizar para a sua resolução.

Na Secção 3, é feita a modelação do problema como um problema de satisfacção de restrições, descrevendo as variáveis, as restrições, funções de avaliação e a estratégia utilizada para ordenar e identificar os valores.

Na Secção 4 é apresentada a visualização da solução final.

Na Secção 5, são mostrados os resultados, para bases de conhecimento com tamanhos diferentes e analisados os respectivos resultados.

Na Secção 6, são apresentadas as conclusões, a aplicabilidade da linguagem utilizada para este tipo de problema e sugestões de melhoria do trabalho.

2 Descrição do Problema

Tendo como motivação o problema de um pai de uma família numerosa, que pretende proporcionar aos seus filhos a possibilidade de frequentarem uma actividade extracurricular, o problema em análise é descrito como um problema de alocação de horários e de optimização de percurso.

Este problema será descrito por um conjunto de restrições e optimizações, como apresentado na Secção 3, tendo em conta o seguinte:

- Os filhos devem frequentar actividades extracurriculares, nomeadamente desporto e música.
- Todas as actividades têm locais e horários diferentes.
- As actividades destinam-se a crianças de idades diferentes.
- A alocação de actividades deve ter em conta a preferência de cada filho, dentro do possível.
- Os filhos mais velhos podem deslocar-se a pé para a sua actividade, para a actividade de um irmão, ou de regresso a casa.
- O pai faz questão de estar nos locais das actividades com uma margem de 5min, para não deixar os filhos muito tempo à espera.
- Percorrer cada caminho, demora um tempo diferente e será mais demorado se for a pé.

3 Abordagem

3.1 Variáveis de Decisão

As seguintes variáveis serão utilizadas para aplicação de restrições e decisão:

- **Children**: Uma lista com identificação de cada filho de 1 até ao número de filhos (depende da base de conhecimento)
- **Activities**: Uma lista com identificação de cada actividade de 1 até ao número de actividades (tipicamente igual ao número de filhos e também dependente da base de conhecimento)
- **Route**: Uma lista onde será apresentada a rota a percorrer, onde serão guardados os ids de cada troço do grafo de caminhos disponíveis que devem ser utilizados. Esta lista irá apresentar por ordem os caminhos a percorrer. Esta lista terá um tamanho igual ao dobro do número de filhos, para garantir que é possível, sair de casa, levar e recolher cada um dos filhos e regressar a casa. Cada elemento da lista pode ter o valor zero, para marcar um ponto que não é visitado, os zeros aparecem no final da lista e podem surgir em casos em que há filhos a percorrem caminhos a pé, significando menos um caminho a percorrer pelo pai. Para além do zero, os restantes valores da lista terão um valor igual aos ids de cada caminho (troço) do grafo. De notar que todos os nós do grafo têm ligação a todos os outros nós.
- **ExcludingRoutes**: Uma lista, que há semelhança da anterior, guarda ids de caminhos do grafo, mas marca os caminhos que são percorridos a pé e como tal excluídos da rota do pai. O domínio desta lista é o mesmo, tendo também zeros a marcar posições não percorridas. No entanto, a ordem nesta lista não é importante, pois rapidamente se identifica o filho que anda a pé e os caminhos que ele pode percorrer, como se verá mais à frente na Secção 4.
- **SumResult**: Uma variável, utilizada para avaliar as preferências de cada filho e tentar melhorar a alocação de cada filho à sua actividade preferida.
- **Count**: Uma variável utilizada para contabilizar o número de caminhos a pé e com o objectivo de ser maximizada para facilitar o percurso do pai, reduzindo os caminhos que este pode percorrer.

Para além destas variáveis, ao longo da execução são também guardadas as seguintes variáveis. Estas variáveis não são utilizadas em labeling, mas estão relacionadas com as anteriores e são utilizadas para aplicar restrições:

- **StartTimes**: Uma lista com os tempos de início de cada actividade, que é posteriormente relacionada com a variável **Activities**. As horas são guardadas no formato *'HHMM'*.
- **EndTimes**: Uma lista com os tempos de fim de cada actividade, que é posteriormente relacionada com a variável **Activities**. As horas são guardadas no formato *'HHMM'*.
- **Ages**: Uma lista com as idades de cada filho, posteriormente relacionada com a variável **Children**.
- **MinAges**: Uma lista com as idades mínimas para frequentar cada actividade, posteriormente relacionada com a variável **Activities**.
- **RoutePaths**: Uma lista com a descrição de cada troço do grafo, onde se guardam as posições interligadas por este caminho e o tempo para o percorrer de carro. Os tempos são guardados em minutos.

São ainda utilizadas mais algumas variáveis para cálculos intermédios, mas sem grande interesse para serem descritas.

3.2 Restrições

Apresenta-se de seguida uma descrição das restrições que devem obrigatóriamente ser respeitadas:

Respeitar as idades mínimas de cada actividade Um filho só pode frequentar uma actividade se a sua idade for maior ou igual à idade mínima da actividade. Os limites de idade são dados pela base de conhecimento e a restrição é implementada da seguinte forma:

```
1  IdadeFilho #>= IdadeMinimaActividade
```

O pai faz questão de estar nos locais das actividades com uma margem de 5min Esta restrição deve ter em conta os tempos de início e de fim de cada actividade, o local de início e de fim de cada troço do grafo e o tempo que demora a percorrer esse troço, para poder verificar se o troço pode ou não ser percorrido e caso não possa, invalidar a possibilidade de esse troço ser incluido na rota. A verificação da margem de 5 min é feita recorrendo ao seguinte conjunto de restrições:

```
1  ((StartTime2 #>= ArrivalTime3 #/\ StartTime2 #=< ArrivalTime2) #\/
2  (StartTime2 #>= ArrivalTime5 #/\ StartTime2 #=< ArrivalTime4) #\/
3  (EndTime2 #>= ArrivalTime3 #/\ EndTime2 #=< ArrivalTime2)
   #\/
4  (EndTime2 #>= ArrivalTime5 #/\ EndTime2 #=< ArrivalTime4)
5  ) #=> (Value #= VoidNumber),
6
7  ((StartTime2 #=< ArrivalTime3 #/\ EndTime2 #=< ArrivalTime3) #\/
8  (StartTime2 #=< ArrivalTime3 #/\ EndTime2 #>= ArrivalTime2) #\/
9  (StartTime2 #=< ArrivalTime3 #/\ EndTime2 #=< ArrivalTime5) #\/
10 (StartTime2 #=< ArrivalTime3 #/\ EndTime2 #>= ArrivalTime4) #\/
11 (StartTime2 #>= ArrivalTime2 #/\ EndTime2 #=< ArrivalTime3) #\/
12 (StartTime2 #>= ArrivalTime2 #/\ EndTime2 #>= ArrivalTime2) #\/
13 (StartTime2 #>= ArrivalTime2 #/\ EndTime2 #=< ArrivalTime5) #\/
14 (StartTime2 #>= ArrivalTime2 #/\ EndTime2 #>= ArrivalTime4) #\/
15 (StartTime2 #>= ArrivalTime4 #/\ EndTime2 #=< ArrivalTime3) #\/
16 (StartTime2 #>= ArrivalTime4 #/\ EndTime2 #>= ArrivalTime2) #\/
17 (StartTime2 #>= ArrivalTime4 #/\ EndTime2 #=< ArrivalTime5) #\/
18 (StartTime2 #>= ArrivalTime4 #/\ EndTime2 #>= ArrivalTime4) #\/
19 (StartTime2 #=< ArrivalTime5 #/\ EndTime2 #=< ArrivalTime3) #\/
20 (StartTime2 #=< ArrivalTime5 #/\ EndTime2 #>= ArrivalTime2) #\/
21 (StartTime2 #=< ArrivalTime5 #/\ EndTime2 #=< ArrivalTime5) #\/
```

```
22 (StartTime2 #=< ArrivalTime5 #/\ EndTime2 #>= ArrivalTime4)
23 ) #=> (Value #=> PathId),
```

Com este conjunto de restrições, vê-se se é possível ir do ponto A para o ponto B e chegar a B antes da hora de início ou de fim, da actividade que aí decorre com uma margem de mais ou menos 5min. As horas de início e de fim da actividade B são confrontadas com os tempos decorridos entre a hora de início ou de fim de A, mais o tempo de viagem mais ou menos a margem. As variáveis **ArrivalTime_** são estes valores calculados. Estes cálculos para a hora de chegada ao ponto B, são feitos previamente, para garantir que somar um determinado número de minutos a uma hora se traduz numa hora correcta. No final das implicações é atribuido um valor à variável **Value**, que pode ser o identificador do caminho, caso o troço não possa ser efectuado e como tal a lista de rotas não o poderá ter ou então um valor por defeito que permite aplicar na mesma o predicado de restrição de caminho mas que não terá efeitos.

Percorrer cada caminho, demora um tempo diferente Esta restrição já está implementada no grafo, ou seja, a descrição dos caminhos na base de conhecimento garante esta diversidade.

Todas as actividades têm locais e horários diferentes Tal como a condição anterior, a base de conhecimento garante esta diversidade. Esta restrição, faz sentido no âmbito do enunciado do problema, numa situação real, faria sentido dois filhos poderem ir para a mesma actividade. Por esta razão, optou-se por garantir que a base de conhecimento tem apenas actividades diferentes, em locais diferentes e com horários diferentes.

Quanto a restrições que devem ser respeitadas, apenas se possível, identificam-se as seguintes: (A optimização destas variáveis irá traduzir a qualidade da solução).

A alocação de actividades deve ter em conta a preferência de cada filho Esta restrição deve ser tida em conta apenas se possível, pois, por exemplo, um filho com 5 anos que tenha preferência por uma actividade para maiores de 16 anos, não deve poder ser alocado a essa actividade. Para respeitar esta restrição da melhor forma, foi incluído na base de conhecimento um predicado que identifica cada filho e a atribuição de um peso a cada actividade em função da sua preferência. Os pesos são múltiplos de 10 e o valor mais alto identifica a actividade preferida. Para cada filho e para cada actividade é criada uma lista de implicações que será avaliada pelo labeling, resultando numa preferência para cada conjunto filho x preferência. O conjunto de preferências é somado e esse valor é passado para o labeling para ser maximizado. Cada implicação de preferência tem o seguinte formato:

```
1 (C#=Cid #/\ A#=Aid) #=> (Pref #= Weight)
```

Os filhos mais velhos podem deslocar-se a pé para a sua actividade, para a actividade de um irmão, ou de regresso a casa Esta restrição é efectuada começando por criar uma lista de todos os filhos que podem andar a pé, em função da informação de idade mínima para andar a pé constante da base de conhecimento. Posteriormente, cria-se uma lista de todos os caminhos que podem ser percorridos a pé, em função do tempo que cada caminho demora e do tempo máximo para andar a pé, que também está definido na base de conhecimento. Com estas informações, quando se faz a avaliação de cada troço do grafo, verifica-se se o troço pode ser percorrido a pé com o predicado:

```
checkIfItsAWalkingRoute(Route,Path,Result).
```

O valor de **Result** pode ser um ou zero, conforme possa ser um troço percorrido a pé ou não. Caso seja possível percorrer a pé, percorre-se a lista de filhos que podem andar a pé para verificar se algum deles está alocado à actividade onde se inícia o grafo. Caso esteja, calcula-se qual será o tempo de chegada ao próximo ponto, tendo em conta o tempo de fim da actividade desse filho mais o tempo que demora a percorrer o caminho. Esta hora de chegada, deve coincidir com a hora de início ou de fim da actividade no ponto seguinte, caso contrário, o filho não pode percorrer esse caminho a pé, pois não irá estar no ponto B quando o pai passar por lá. Caso seja possível percorrer o caminho a pé, utiliza-se o predicado de restrição de caminho para garantir que o pai não percorre esse caminho. No final, se o caminho a pé for válido, é incrementada a variável **Count**, significando mais um caminho a pé. Esta variável é maximizada em labeling para promover os caminhos a pé.

Quando os caminhos vão de casa para a actividade ou da actividade para casa, a estratégia é ligeiramente diferente, pois o tempo de chegada ou de partida não é relevante, basta apenas verificar que o caminho pode ser percorrido a pé, tendo em conta o tempo que demora. O predicado responsável por esta verificação é:

```
checkWalkPathForChild(
        Route,
        [Id|T],
        PathId,
        Children,
        Activities,
        StartTimes,
        EndTimes,
        VoidValue,
        [P1,P2],
        ExcludingRoutes,
        ExcludingActualPosition,
        NewExcludingPosition,
        CountAux,
        NewCount
)
```

3.3 Função de Avaliação

A função de avaliação da solução foi já apresentada na descrição das restrições. Utiliza-se duas variáveis, associadas à restrição de preferências e de caminhos a pé para tentar melhorar a solução. A variável de preferências, que representa a soma global das preferências de cada filho relativamente à actividade atribuida, é maximizada e quanto mais próxima do valor máximo, melhor será a solução de alocação de actividades. O número de caminhos que serão feitos a pé, serve como indicativo para avaliar a solução da rota, quanto mais caminhos a pé, menos trajectos serão feitos de carro e como tal a solução é melhor. Ambas as avaliações estão dependentes dos valores colocados nas bases de conhecimento, que podem ser facilmente manipuladas para pequenas bases de conhecimento, mas para as grandes, caberá em grande parte ao programa a optimização.

3.4 Estratégia de Pesquisa

Como o programa pode receber bases de conhecimento de tamanhos diferentes, começa-se por avaliar a base de conhecimento, para saber o número de filhos, o número de actividades, idades, horas de início e de fim das actividades. Com estes valores são estabelecidos os domínios para cada variável. Os filhos e as actividades são identificados por números sequênciais que começam em 1. A posição nas listas do filho e da actividade que ele frequenta é igual. Relativamente às listas de idades e de horas, como os domínios não são intervalos contínuos, como os ids, utiliza-se a função **global_cardinality** para atribuir o domínio. Para isso, é gerado um intervalo no formato a ser utilizado por esta função, recorrendo ao predicado:

```
1  getGlobalCardinalityDomain([],G,G).
2          getGlobalCardinalityDomain([A|T],B,G):-
3          A1 = [A-1],
4          append(A1,B,R),
5          getGlobalCardinalityDomain(T,R,G).
6
7  getGlobalCardinalityDomain(List,G):-
8          getGlobalCardinalityDomain(List,[],G).
```

Para ligar as diferentes listas utiliza-se o seguinte predicado:

```
1  linkIndexesAndValues(Original,Indexes,Values):-
2          linkIndexesAndValues(Original,Original,Indexes,Values).
3
4  linkIndexesAndValues([],_,_,_).
5  linkIndexesAndValues(Search,Original,Indexes,Values):-
6          min_member(M,Search),
7          delete(Search,M,Rest),
8          nth1(P,Original,M),
9          element(Pos,Indexes,P),
10         element(Pos,Values,M),
```

```
11          linkIndexesAndValues(Rest, Original, Indexes, Values).
```

Este predicado tira partido do facto de na base de conhecimentos os valores aparecem por ordem, como tal, a ordem em que aparece cada idade corresponde ao índice do filho ou da actividade.

Tendo estas variáveis principais conectadas e com domínios estabelecidos, são aplicadas as restrições e avaliações já descritas anteriormente.

Para a alocação às actividades, são realizadas as operações descritas nas respectivas restrições.

Relativamente ao cálcula da rota, para além do já referido, é importante salientar que é gerada uma lista **RoutePaths** onde tem todos os caminhos possíveis, definidos pela base de conhecimento. Primeiro aparecem os caminhos de casa para as actividades, depois os caminhos de cada uma das actividades e por fim os caminhos de regresso a casa. Com este mapa de rotas, são percorridos todos os caminhos para avaliar se o caminho pode ser percorrido a pé ou se é um caminho válido. As posições de cada caminho nesta variável, serão utilizadas como identificador do caminho a colocar na variável **Route**, para ser fácil identificar os caminhos no final.

4 Visualização da Solução

A solução é apresentada nesta fase em modo de listas (esta será uma das melhorias a introduzir no trabalho). No final do cálculo, apresentam-se no ecrã as variáveis

- **SumResult** e a variável **Count**, para avaliação da qualidade da solução.
- Uma lista com os ids dos filhos.
- Uma lista com os ids das actividades, em que as posições correspondem à lista de filhos
- Uma lista com a rota com a sequência de caminhos a efectuar
- Uma lista com os ids dos filhos que podem andar a pé
- Uma lista com os caminhos que serão percorridos a pé
- Uma lista com o grafo, para fácil identificação dos caminhos e posições

5 Resultados

Para testar esta implementação e ter soluções interessantes é importante ter bases de conhecimento sólidas e onde estejam já previstos todos os caminhos e atribuições de actividades a cada filho, caso contrário irá apenas resultar a informação a dizer que não existe solução.

Esta implementação foi testada com uma base de conhecimento com quatro filhos e quatro actividades, que gera um grafo com vinte caminhos, o que é já um valor considerável para avaliar as soluções.

Várias alterações foram introduzidas na base de conhecimento para validar os diferentes aspectos da solução, nomeadamente os factores de qualidade. É

interessante analisar que não aplicando as opções de maximização no labeling a obtenção de resultados é muito mais rápida, mas não é a melhor solução.

Para esta dimensão, não foi necessário colocar timeouts no labeling, obtendo-se soluções em poucos segundos.

Uma solução possível terá a seguinte vizualização:

:: SumResult: 150
:: Count: 1
:: Children: [1,2,3,4]
:: Activities: [2,1,3,4]
:: Route: [1,6,15,10,7,12,18,14]
:: ChildrenCanWalk: [1]
:: WalkingPaths: []
::[[0,1,12],[0,2,18],[0,3,6],[0,4,6],...

6 Conclusões e Trabalho Futuro

A principal conclusão a retirar deste trabalho é a grande aplicabilidade da programação por restrições, são vários os casos reais em que se podem utilizar estas metodologias. O próprio âmbito deste trabalho, é um problema que tem aplicabilidade prática. A utilização destes conceitos com a linguagem de programação *PROLOG* é também bastante interessante. A biblioteca *cplfd* disponibiliza várias ferramentas que permitem um rápido desenvolvimento e com poucas linhas de código. O mesmo desenvolvimento noutras linguagens de programação seria muito mais verboso. Apesar desta aparente simplicidade, a utilização desta linguagem de programação em conjunto com esta biblioteca requer uma forma de pensar bastante diferente do que é habitual em outras linguagens mais comuns. Esta mudança de pensamento, pode levar a seguir caminhos que não resultam em soluções viáveis, levando a implementações desnecessárias. É também muito fácil instanciar variáveis antes de se aplicar o labeling, o que irá originar variáveis baseadas em backtracking, que não representam a solução óptima. Por vezes, detectar este erro é bastante demorado.

A linguagem *PROLOG* é uma linguagem que requer bastante prática e utilização, devido a esta forma de pensar algo peculiar, mas a possibilidade de descrever alguns problemas quase como em linguagem natural tornam o *PROLOG* muito interessante para problemas de lógica.

No desenvolvimento do trabalho houve uma grande preocupação de tornar a solução o mais geral possível, evitando recorrer a variáveis *hard-coded* e o objectivo foi cumprido, sendo possível alterar a base de conhecimento, alterando apenas a linha de include. Esta particularidade, origina linhas de código que parecem um pouco redundantes, mas necessárias para se conseguir instanciar o ponto inicial do problema.

Como trabalho futuro, várias melhorias podem ser feitas neste trabalho, sendo que muitas delas não foram realizadas apenas por restrição de tempo e como tal são de fácil implementação. Entre elas destacam-se as seguintes:

- Melhorar a representação da solução. É relativamente simples, com os valores obtidos, cruzar os resultados com a base de conhecimento e imprimir os locais por onde deve passar o pai, os nomes dos filhos e das actividades e eventualmente criar uma tabela com os dados.
- Explorar o algoritmo com bases de conhecimento de maiores dimensões para avaliar performance.
- Permitir que mais do que um filho possa frequentar a mesma actividade.
- Permitir filhos e actividades com a mesma idade. Esta alteração deve ser relativamente simples, provávelmente retirando a condição de all_distinct deve ser suficiente.
- Permitir actividades com mais do que uma ocorrência e em dias diferentes. Para isto têm que ser revistos os predicados de manipulação de tempos para terem em conta os dias da semana e adicionar mais restrições, tais como, garantir que a rota não começa e acaba em dias diferentes. Durante a semana, a hora de início das actividades deve coincidir com o regresso do pai do trabalho, entre outras.
- Explorar mais opções de labeling para comparar soluções e tentar introduzir mais variáveis de qualidade da solução, como por exemplo, encontrar actividades que levem à rota mais curta. No entanto, esta melhoria terá apenas interesse para bases de conhecimento maiores, eventualmente com 8 ou 16 filhos, o que já deixa de ter aplicabilidade real no âmbito da motição inicial do problema.

References

1. Markus Triska.: Correctness Considerations in CLP(FD) Systems. Dissertation, (2013)
2. Waldemar Kocjan.: Heuristic methods for routing and scheduling. Article, (2001)
3. CLP(FD) Constraint Logic Programming over Finite Domains, http://www.pathwayslms.com/swipltuts/clpfd/clpfd.html.