Trid

Uma Abordagem Lógica

FEUP PLOG, Turma 3MIEIC05, Grupo Trid_2

Antero Gandra - 201607926

Fernando Fernandes – 201505821

Resumo

Face a vasta gama de jogos possíveis para escolha do segundo projeto escolhemos o jogo Trid, que exige ao jogador uma elevada capacidade de raciocínio, tal como nos semelhantes mas conhecidos jogos de sudoku e kakuro, devido á necessidade de prever o resultado das somas, tornando-se um desafio interessante para se resolver.

Começamos por abordar o problema de cima pra baixo (abordagem top down) preocupando-nos primeiro com o funcionamento geral do jogo, começando a aprofundarmo-nos cada vez mais nos detalhes com o passar do tempo.

Para a formulação do projeto foram necessárias várias reuniões para além das aulas práticas. De modo a haver uma melhor dinâmica de grupo, o trabalho foi distribuído pelos dois membros.

Apesar das dificuldades, o trabalho foi concluído com sucesso, sendo que foram cumpridos todos os objetivos que se encontravam planeados.

Com este trabalho aprendemos a resolver um problema de decisão eficientemente com o uso correto da linguagem Prolog e a importância e utilidade da biblioteca clpfd.

Introdução

Para este trabalho tivemos como objetivo, com aplicação das regras da lógica com restrições, obter um programa que simule o jogo de raciocínio Trid, uma vez que, tal como um jogo muito apreciado por nós (Sudoku), este é baseado em somas, apesar de demonstrar uma maior complexidade, quer no desenho do tabuleiro, quer no número de abordagens que se podem tomar.

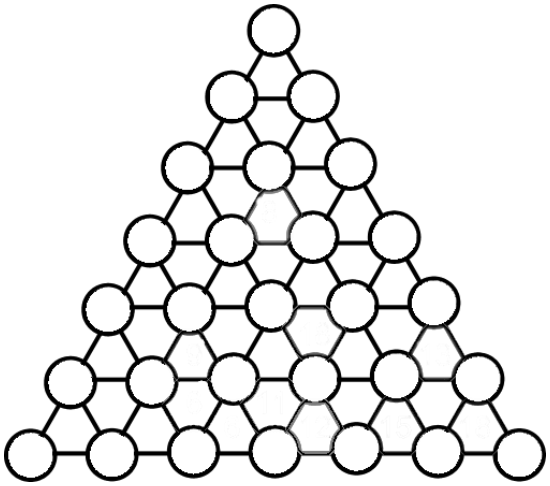
Nesse artigo começaremos por descrever e expor o problema em questão, seguido da abordagem utilizada para a sua resolução. Depois para concluir mostramos a solução em si seguida pelos resultados obtidos por esta.

Regras

Baseado numa variação triangular do jogo Sudoku, Trid consiste num puzzle enigmático introduzido ao mundo em 2009 através de vários concursos de renome internacional sobre jogos de lógica tais como o OAPC (Ogos Atay Puzzle Contest).

O esquema do puzzle consiste num grande triângulo equilátero dividido em vários triângulos mais pequenos (também equiláteros) com os seus vértices destacados (para preenchimento por parte de quem soluciona o enigma).

Esquema:



Este jogo, tal como Sudoku, é também apenas baseado na soma algébrica tendo como restrição de que os números que vão ter de ser adicionados aos vértices têm de resultar no número guardado no interior dos triângulos.

Por associação resultante ao nome, neste puzzle cada valor guardado no interior dos triângulos é que vai ditar os valores guardados nos seus três vértices (daí o nome Trid) sendo que os valores dos vértices vão ter de ser diferentes uns dos outros. Isto vem como consequência da regra do enigma que impõe que cada linha que compõe os triângulos não pode ter valores repetidos.

O tamanho do triângulo varia sendo que o valor máximo de vértices numa linha define o valor máximo que se pode indicar dentro dos vértices. Este detalhe é também resultado da regra escrita anteriormente, de modo a que existam sempre valores suficientes para preenchimento da linha, que nunca se repitam nesta.

Uma estratégia apropriada para a resolução do problema consiste em começar por tratar primeiro de duas das três linhas exteriores do triângulo grande avançando para os triângulos de dentro e finalizar acabando de preencher a linha restante do triangulo.

Abordagem:

Estratégia de Pesquisa:

A estratégia de etiquetagem utilizada foi ff(First Fail), que seleciona a próxima variável mais à esquerda com o menor domínio.

A opção default(leftmost), para casos maiores, 13 por exemplo, experimenta valores sequenciais (1,2,3,4,5, 6...) o que não é de todo adequado, o que se traduz em maiores tempos de resolução

```
tridLabel([]).
tridLabel([H|T]):-
  labeling([ff],H),
  tridLabel(T).
```

Variáveis de Decisão:

As variáveis de decisão, no nosso caso, correspondem aos vértices do triângulo representados por uma lista de listas em Prolog (são os elementos dessa lista). O domínio desses elementos vai de 1 até ao tamanho do lado do triângulo, ou seja, o tamanho da primeira lista dessa matriz.

```
defineDominio(Lado, Piramide):-defineDominioTriangulo(Lado,Piramide).

defineDominioTriangulo(_, []).
defineDominioTriangulo(Lado, [H|T]):-
  domain(H, 1, Lado),
  all_distinct(H),
  defineDominioTriangulo(Lado, T).
```

Restrições:

Cada valor do elemento dos vértices dos triângulos não pode estar repetido nem horizontalmente nem diagonalmente.

Como tal foram usadas as restrições all_distinct para o garantir.

Existem grupos de 3 vértices que formam pequenos triângulos, dentro do maior, e o valor destes pequenos triângulos tem de ser a soma dos valores dos 3 vértices que os compõem.

Para implementar isto em Prolog foi preciso agrupar estes grupos de 3 vértices em listas e usar a restrição sum.

```
dominioDiagonalEsquerda(_,_,1).
dominioDiagonalEsquerda(Triangulo,NumeroElemento,NumeroMaxLista):-
  dominioDiagonalEsquerdaAux(Triangulo, Lista, NumeroElemento, NumeroMaxLista),
  all_distinct(Lista),
  NewNumero is NumeroElemento + 1,
  NewMax is NumeroMaxLista - 1,
  dominioDiagonalEsquerda(Triangulo,NewNumero,NewMax).

dominioDiagonalDireita(_,0).
dominioDiagonalDireita(Triangulo, NumeroElemento):-
  dominioDiagonalDireitaAux(Triangulo, Lista, NumeroElemento),
  all_distinct(Lista),
  NewNumero is NumeroElemento - 1,
  dominioDiagonalDireita(Triangulo,NewNumero).

dominioDiagonalDireitaAux(_,[],-1).
dominioDiagonalDireitaAux([L|R], [H|T], NumeroElemento):-
  nth0(NumeroElemento, L, H),
  NewNumero is NumeroElemento - 1,
  dominioDiagonalDireitaAux(R,T,NewNumero).


dominioDiagonalEsquerdaAux(_,[],_,0).
dominioDiagonalEsquerdaAux([L|R], [H|T], NumeroElemento, NumeroMaxLista):-
  nth0(NumeroElemento, L, H),
  NewMax is NumeroMaxLista - 1,
  dominioDiagonalEsquerdaAux(R,T,NumeroElemento, NewMax).
```

```
dominioSomas(_, []).
dominioSomas(Triangulo, [Val - H|T]):-
  formaLista(Triangulo, H, Lista),
  !,
  sum(Lista, #=, Val),
  dominioSomas(Triangulo, T).

formaLista(_,[],[]).
formaLista(Triangulo, [C|R], [H|T]):-
  formaListaAux(Triangulo, C, H),
  formaLista(Triangulo, R, T).

formaListaAux([Linha|Resto], Contador, Valor):-
  formaListaAuxLinha(Linha, Resto, Contador, Valor).

formaListaAuxLinha([H|_],_, 1, H).
formaListaAuxLinha([],Resto,Contador,Valor):-formaListaAux(Resto, Contador,Valor).
formaListaAuxLinha([_|T], Resto, Contador,Valor):-
  Cont is Contador - 1,
  formaListaAuxLinha(T, Resto, Cont, Valor).
```

Visualização da Solução:

Para implementar um tabuleiro triangular em Prolog, primeiro reverte-se a matriz que o representa de modo a que o primeiro elemento seja a lista com um elemento, e calcula-se quantas listas tem a matriz.

```
mostraTrid(Triangulo):-
  write('\n'),
  length(Triangulo, Tam),
  Tamanho is Tam + 10,
  reverse(Triangulo, TrianguloAux),
  mostraTridAux(TrianguloAux, Tamanho).

mostraTridAux([],_).
mostraTridAux([H|T], Tamanho):-
  mostraEspaco(Tamanho),
  mostraLinha(H),
  NewTam is Tamanho - 1,
  mostraTridAux(T, NewTam).

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

mostraEspaco(1).
mostraEspaco(Tamanho):-
  write('  '),
  NewTam is Tamanho - 1,
  mostraEspaco(NewTam).
```

Depois imprime-se lista a lista, começando na nova ordem revertida, imprimindo um número de espaços equivalente ao tamanho da matriz (calculado anteriormente) e que vai sendo decrementado em 1, à medida que se passa para uma lista nova.

É de salientar que para não haver confusão na visualização do resultado devido ao excesso de informação só imprimimos os resultados dos vértices, uma vez que os valores interiores do triângulo são apenas necessários na indicação para efetuar os cálculos. O triângulo impresso fica assim equilátero.

Resultados:

De modo a demonstrar o funcionamento do nosso programa simulamos 2 casos possíveis.

- Com indicações dos valores interiores dos triângulos:

Nº máximo de vértices= 5;

As indicações são:

- a soma do 1º do 2º e do 6º vértice têm de dar 6;
- a soma do 2º do 3º e do 7º vértice têm de dar 12;
- a soma do 3º do 4º e do 8º vértice têm de dar 8;
- a soma do 6º do 7º e do 10º vértice têm de dar 9;
- a soma do 8º do 9º e do 12º vértice têm de dar 6;
- a soma do 10º do 11º e do 13º vértice têm de dar 12.

```
          5
        4   1
      3   5   2
    2   4   1   3
  1   3   5   2   4
```

-Sem indicações:

Nº máximo de vértices= 10;

(exemplo dado para demonstrar que se verifica sempre a regra que não permite a repetição dos valores numa linha)

```
            4
          5   3
        9   7   2
      10   6   8   9
    7   9   4   2   8
  8   5   6   3   10   1
2   4   1   5   7   3   6
  6   1   7   8   3   2   4   5
3   8   9   1   2   4   5   6   7
1   2   3   4   5   6   7   8   9   10
```

Este enigma é normalmente feito com os valores dos vértices entre 1 e 7 uma vez que, o número total de combinações, mesmo com a restrição que proíbe a repetição dos mesmos valores numa linha, cresce fatorialmente mediante o aumento do número de vértices numa linha.

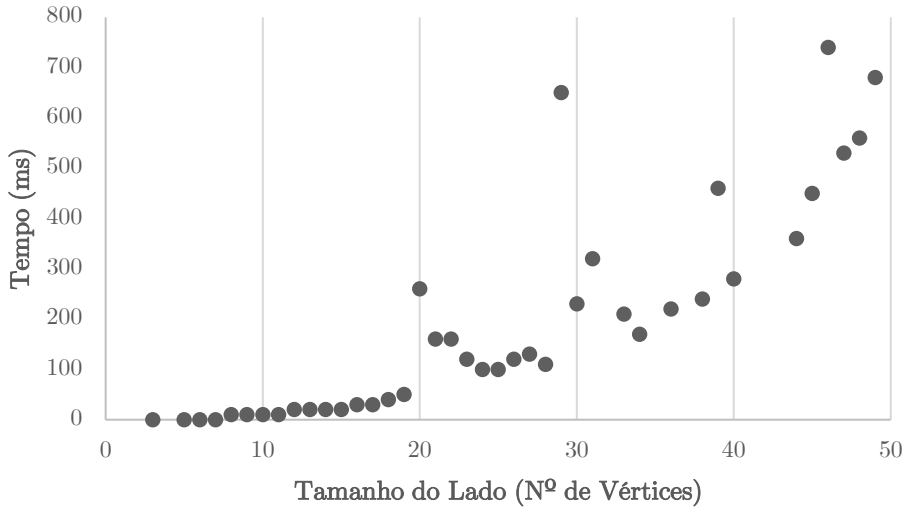
Tabela 1- Relação entre o Número de Vértices numa linha e as suas combinações possíveis

Número de Vértices numa linha	Número de combinações numa linha
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Mesmo assim a nossa solução, face um número de vértices menores que 20 possui resultados imediatos. Apesar disto, face alguns valores, detetamos raros casos em que, na medição dos tempos isso não se ocorreu.

Com a medição dos tempos, como seria de esperar, verifica-se que com um aumento da complexidade do problema houve também um consequente aumento do tempo necessário para a sua resolução.

Gráfico 1- Relação entre Tempo e o Nº Máximo de Vértices numa Linha



Conclusões

Com o uso das novas bibliotecas conseguimos de uma maneira simples e eficaz arranjar soluções rápidas a problemas que se fossem resolvidos manualmente necessitariam de grande tempo para serem solucionados.

Os resultados obtidos demonstram que este problema não foge á exceção sendo que o puzzle agora é solucionado de maneira intuitiva, com a demonstração correta dos resultados, na sua forma triangular.

Como o esquema do puzzle é complexo isso limitou-nos a só poder imprimir resultados com, no máximo, 2 algarismos. Valores mais elevados teriam como consequência haver uma deformação das formas triangulares, demonstrando um resultado de difícil compreensão.

Tirando isso, a nossa solução demonstra uma resolução bastante eficiente ao problema, tendo em conta todas as regras condicionais contidas no enigma.

A capacidade de demonstração do esquema triangular demonstrou-se como sendo a nossa maior dificuldade, uma vez que, como o esquema tem tamanho variável, isso implicou um constante reposicionamento dos valores, de modo a que o triangulo se mantivesse equilátero.

De modo a melhorar o trabalho podíamos realizar tarefas extras ao que nos propuseram, tais como uma interface com o utilizador mais intuitiva, de modo a que o utilizador possa resolver por ele mesmo, com o computador a dar dicas.

Este trabalho demonstrou-se de elevada importância para projetos futuros, nomeadamente na área da inteligência artificial. Conseguimos, assim, meter o computador a resolver problemas na área de cálculo e de ordenação e a raciocinar de maneira a criar equilíbrio e organização (ao fazer com que os valores se posicionassem nos vértices corretos para os triângulos ficarem corretamente preenchidos).

Bibliografia:

Web:

https://www.gmpuzzles.com/blog/2013/11/puzzle-robot-21-trid/

http://www.sachsentext.de/en/taxonomy/term/403