PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

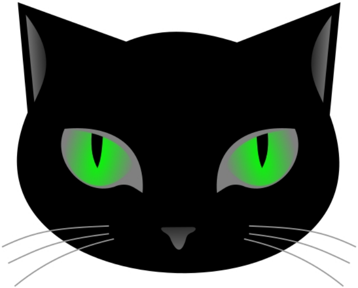
claudio.sa@udesc.br

Departamento de Ciência da Computação – DCC
Centro de Ciências e Tecnológias – CCT
Universidade do Estado de Santa Catarina – UDESC

7 de maio de 2019

Programação por Restrições

- Conceituar a PR
- Princípios
- 03 exemplos
- 03 técnicas
- Aprendizagem da PR via estudos de casos
- Vamos dividir esta seção



Programação por Restrições (PR) – I

- A Programação por Restrições (PR) é conhecida por *Constraint Programming* ou simplesmente **CP**

Programação por Restrições (PR) – I

- A Programação por Restrições (PR) é conhecida por *Constraint Programming* ou simplesmente **CP**
- Uma poderosa teoria (e técnica) que contorna a complexidade de certos problemas exponenciais

Programação por Restrições (PR) – I

- A Programação por Restrições (PR) é conhecida por *Constraint Programming* ou simplesmente **CP**
- Uma poderosa teoria (e técnica) que contorna a complexidade de certos problemas exponenciais
- A **PR** encontrava-se inicialmente dentro da IA e PO, mas como várias outras áreas, tornaram-se fortes e autônomas. Atualmente uma área de pesquisa bem forte em alguns países.

Programação por Restrições (PR) – I

- A Programação por Restrições (PR) é conhecida por *Constraint Programming* ou simplesmente **CP**
- Uma poderosa teoria (e técnica) que contorna a complexidade de certos problemas exponenciais
- A **PR** encontrava-se inicialmente dentro da IA e PO, mas como várias outras áreas, tornaram-se fortes e autônomas. Atualmente uma área de pesquisa bem forte em alguns países.
- Nesta seção, temos 3 exemplos ilustrar conceitos da **PR**

- *Aproximadamente* o algoritmo da **PR** é dado:

Programação por Restrições (PR) – II

- *Aproximadamente* o algoritmo da **PR** é dado:
  1. Avaliar algebricamente os domínios das variáveis com suas restrições
  2. Intercala iterativamente a propagação de restrições com um algoritmo de busca
  3. A cada variável instanciada, o processo é repetido sobre as demais variáveis, reduzindo progressivamente o espaço de busca
  4. Volte ao passo inicial até que os domínios permaneçam estáticos e que as variáveis apresentem instâncias consistentes

Programação por Restrições (PR) – II

- *Aproximadamente* o algoritmo da **PR** é dado:
  1. Avaliar algebricamente os domínios das variáveis com suas restrições
  2. Intercala iterativamente a propagação de restrições com um algoritmo de busca
  3. A cada variável instanciada, o processo é repetido sobre as demais variáveis, reduzindo progressivamente o espaço de busca
  4. Volte ao passo inicial até que os domínios permaneçam estáticos e que as variáveis apresentem instâncias consistentes
- Este núcleo é uma busca por constantes otimizações

Programação por Restrições (PR) – II

- *Aproximadamente* o algoritmo da **PR** é dado:
  1. Avaliar algebricamente os domínios das variáveis com suas restrições
  2. Intercala iterativamente a propagação de restrições com um algoritmo de busca
  3. A cada variável instanciada, o processo é repetido sobre as demais variáveis, reduzindo progressivamente o espaço de busca
  4. Volte ao passo inicial até que os domínios permaneçam estáticos e que as variáveis apresentem instâncias consistentes
- Este núcleo é uma busca por constantes otimizações
- Uma das virtudes da PR: a legibilidade e clareza de suas soluções, conhecidos como **modelos**

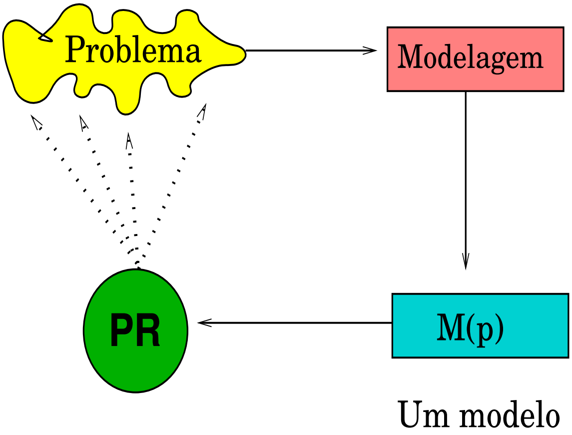
Programação por Restrições (PR) – III

- Problemas combinatoriais com domínio nos inteiros são bons candidatos a serem resolvidos por PR

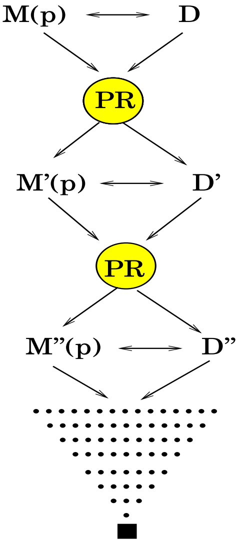
- Problemas combinatoriais com domínio nos inteiros são bons candidatos a serem resolvidos por PR
- Quando temos problemas que precisamos conhecer **todas** as respostas, não apenas a melhor resposta

Programação por Restrições (PR) – III

- Problemas combinatoriais com domínio nos inteiros são bons candidatos a serem resolvidos por PR
- Quando temos problemas que precisamos conhecer **todas** as respostas, não apenas a melhor resposta
- Quando necessitamos de respostas *precisas* e não apenas as aproximadas. Há um custo computacional a ser pago aqui!


Problema
Modelagem
M(p)
PR
Um modelo

Fluxo de Cálculo da PR



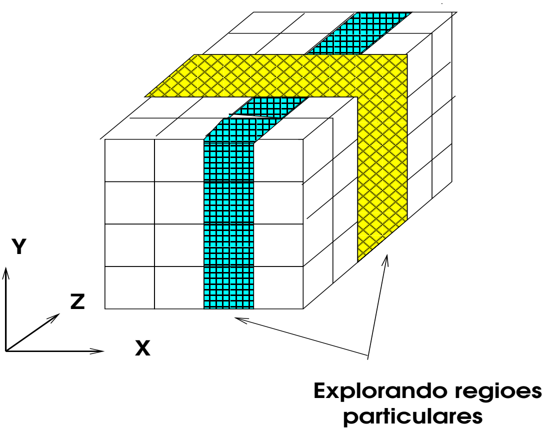


Figura 1: Realizar buscas com regiões reduzidas – promissoras (regiões factíveis de soluções)

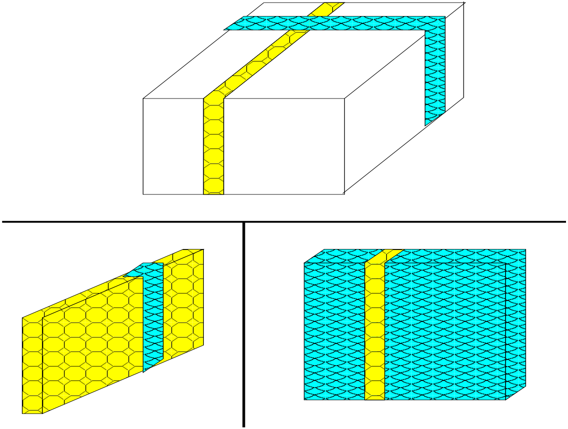


Figura 2: Redução de um CP em outros sub-problemas CPs equivalentes

A PR tem os seguintes elementos:

A PR tem os seguintes elementos:

- Um conjunto de **variáveis**: X_1, X_2, X_3, ..., X_n

Conceitos

A PR tem os seguintes elementos:

- Um conjunto de **variáveis**: X_1, X_2, X_3, ..., X_n
- Um conjunto de **domínios** dessas variáveis: D_{X_1}, D_{X_2}, D_{X_3}, ..., D_{X_n}

Conceitos

A PR tem os seguintes elementos:

- Um conjunto de **variáveis**: X_1, X_2, X_3, ..., X_n
- Um conjunto de **domínios** dessas variáveis: D_{X_1}, D_{X_2}, D_{X_3}, ..., D_{X_n}
- Finalmente, as **restrições**, que são relações n-árias entre estas variáveis

A PR tem os seguintes elementos:

- Um conjunto de **variáveis**: X_1, X_2, X_3, ..., X_n
- Um conjunto de **domínios** dessas variáveis: D_{X_1}, D_{X_2}, D_{X_3}, ..., D_{X_n}
- Finalmente, as **restrições**, que são relações n-árias entre estas variáveis
- Exemplo: $D_{X_1} = D_{X_2} = \{3, 4\}$ e $X_1 \neq X_2$

PR e Picat

- Para o exemplo anterior um código em Picat é dado por:

- Para o exemplo anterior um código em Picat é dado por:
  - `[X1, X2] ::  3..4`
  - `X1 #!= X2`

PR e Picat

- Para o exemplo anterior um código em Picat é dado por:
  - `[X1, X2] :: 3..4`
  - `X1 #!= X2`
- Em resumo, as relações da PR tem o símbolo '#'

PR e Picat

- Para o exemplo anterior um código em Picat é dado por:
  - `[X1, X2] ::  3..4`
  - `X1 #!= X2`
- Em resumo, as relações da PR tem o símbolo '#'
- Para tornar toda esta sintaxe da PR disponível, Picat tem um módulo para suporte da PR:
  `import cp`

Exemplos

1. Soma de dois números primos (problema *ad-hoc*)

Exemplos

1. Soma de dois números primos (problema *ad-hoc*)
2. Escala (simplificada) de consultórios médicos (uso de matriz)

Exemplos

1. Soma de dois números primos (problema *ad-hoc*)
2. Escala (simplificada) de consultórios médicos (uso de matriz)
3. Caixeiro-viajante (uso de matriz binária de decisão) – diferente da solução do Hakank, aqui discutida

Basicamente, 3 problemas distintos!

Exemplo – 01 – Soma de Números Primos

- Dado um número par qualquer, N_{PAR}, encontre dois de números primos, N_1 e N_2, diferentes entre si, que somados deêm este número par.

Exemplo – 01 – Soma de Números Primos

- Dado um número par qualquer, N_{PAR}, encontre dois de números primos, N_1 e N_2, diferentes entre si, que somados deêm este número par.
- Exemplo:
  Seja o PAR $= 18$
  Uma solução:
  $N_1 = 7$ e $N_2 = 11$
  pois
  $N_1 + N_2 = 18$

- N_1 e N_2 assumem valores no domínio dos números primos. Logo, é importante ter os números primos prontos!

- N_1 e N_2 assumem valores no domínio dos números primos. Logo, é importante ter os números primos prontos!
- A soma destes números é o par fornecido como entrada, N_{PAR}: $N_1 + N_2 = N_{PAR}$

Modelagem do Problema

- N_1 e N_2 assumem valores no domínio dos números primos. Logo, é importante ter os números primos prontos!
- A soma destes números é o par fornecido como entrada, N_{PAR}: $N_1 + N_2 = N_{PAR}$
- N_1 e N_2 são diferentes entre si $N_1 \neq N_2$

Modelagem do Problema

- N_1 e N_2 assumem valores no domínio dos números primos. Logo, é importante ter os números primos prontos!
- A soma destes números é o par fornecido como entrada, N_{PAR}: $N_1 + N_2 = N_{PAR}$
- N_1 e N_2 são diferentes entre si $N_1 \neq N_2$
- Como são inteiros: $N_1 < N_{PAR}$ e $N_2 < N_{PAR}$ Sim, é óbvio, mas isto faz uma redução significativa de domínio!

- Acompanhar as explicações do código de: https://github.com/claudiosa/CCS/blob/master/picat/soma_N1_N2_primos_CP.pi
- Confira a execução e testes

Código em Partes

```
modelo =>
    PAR = 382,
    Variaveis = [N1,N2],
    % Gerando um domino soh de primos
    % L_dom = [I : I in 1..1000, eh_primo(I) == true],    %OU
    L_dom = [I : I in 1..1000, prime(I)],
    Variaveis :: L_dom,
```

Código em Partes

```
modelo =>
    PAR = 382,
    Variaveis = [N1,N2],
    % Gerando um domino soh de primos
    % L_dom = [I : I in 1..1000, eh_primo(I) == true],   %OU
    L_dom = [I : I in 1..1000, prime(I)],
    Variaveis :: L_dom,
```

- Uma ótima estratégia: sair com um **domínio de números candidatos**!

Código em Partes

```
modelo =>
    PAR = 382,
    Variaveis = [N1,N2],
    % Gerando um domino soh de primos
    % L_dom = [I : I in 1..1000, eh_primo(I) == true],   %OU
    L_dom = [I : I in 1..1000, prime(I)],
    Variaveis :: L_dom,
```

- Uma ótima estratégia: sair com um **domínio de números candidatos**!
- O par da entrada: **382**
- Quanto maior este valor, maior o número de soluções?

```
    % RESTRICOES
    N1 #!= N2,
    N1 #< PAR,
    N2 #< PAR,
    N1 + N2 #= PAR,

% A BUSCA
solve([ff], Variaveis),
  % UMA SAIDA
printf("\n  N1: %d\t N2: %d", N1,N2),
printf("\n..................................")
.
```

Código em Partes

```
import cp.

% main => modelo .
% main ?=> modelo, fail.
% main =>  true.

main =>
    L = findall(_, $modelo),
    writef("\n Total de solucoes:  %d \n", length(L)) .
```

Saída – I

```
Picat> cl('soma_N1_N2_primos_CP').
Compiling:: soma_N1_N2_primos_CP.pi
** Warning  : redefine_preimported_symbol(math): prime / 1
soma_N1_N2_primos_CP.pi compiled in 7 milliseconds
loading...

yes

Picat> main.

  N1: 3  N2: 379
.....................................
  N1: 23  N2: 359
.....................................
  N1: 29  N2: 353
.....................................
```

```
 .....................................
  N1: 353  N2: 29
.....................................
  N1: 359  N2: 23
.....................................
  N1: 379  N2: 3
.....................................
 Total de solucoes:  18

yes

Picat>
```

- Seja um Posto Atendimento Médico, um PA, com 4 consultórios e 7 especialidades médicas

- Seja um Posto Atendimento Médico, um PA, com 4 consultórios e 7 especialidades médicas
- O problema é distribuir estes médicos nestes 4 consultórios tal que alguns requisitos sejam atendidos (restrições satisfeitas)

Exemplo – 02 – Escala de Consultórios

- Seja um Posto Atendimento Médico, um PA, com 4 consultórios e 7 especialidades médicas
- O problema é distribuir estes médicos nestes 4 consultórios tal que alguns requisitos sejam atendidos (restrições satisfeitas)
- A abordagem aqui é ingênua e sem muitos critérios

- Vamos usar uma matriz bi-dimensional para representar o problema. Linhas $\leftrightarrow$ consultórios (1 a 4), e as colunas $\leftrightarrow$ dias da semana (1 a 5)

- Vamos usar uma matriz bi-dimensional para representar o problema. Linhas $\leftrightarrow$ consultórios (1 a 4), e as colunas $\leftrightarrow$ dias da semana (1 a 5)
- Esta matriz será preenchida com valores/códigos de 1 a 7, de acordo com a especialidade médica.

Modelagem do Problema

- Vamos usar uma matriz bi-dimensional para representar o problema. Linhas $\leftrightarrow$ consultórios (1 a 4), e as colunas $\leftrightarrow$ dias da semana (1 a 5)
- Esta matriz será preenchida com valores/códigos de 1 a 7, de acordo com a especialidade médica.
- Assim o domínio da matriz `Quadro` (4×5) será preenchida com um destes códigos.

Modelagem do Problema

- Vamos usar uma matriz bi-dimensional para representar o problema. Linhas $\leftrightarrow$ consultórios (1 a 4), e as colunas $\leftrightarrow$ dias da semana (1 a 5)
- Esta matriz será preenchida com valores/códigos de 1 a 7, de acordo com a especialidade médica.
- Assim o domínio da matriz `Quadro` (4×5) será preenchida com um destes códigos.
- Vamos utilizar restrições globais: `member` e `all_different`

Modelagem do Problema

- Vamos usar uma matriz bi-dimensional para representar o problema. Linhas $\leftrightarrow$ consultórios (1 a 4), e as colunas $\leftrightarrow$ dias da semana (1 a 5)
- Esta matriz será preenchida com valores/códigos de 1 a 7, de acordo com a especialidade médica.
- Assim o domínio da matriz `Quadro` (4×5) será preenchida com um destes códigos.
- Vamos utilizar restrições globais: `member` e `all_different`
- As restrições globais se aplicam sobre um conjunto de variáveis.

Modelagem – Comentários

- A fase de busca e propagação do comando `solve(Critérios, Variáveis)`, há dezenas de combinações possíveis: consultar o guia do usuário

- A fase de busca e propagação do comando `solve(Critérios, Variáveis)`, há dezenas de combinações possíveis: consultar o guia do usuário
- Tem-se os predicados extras ... são muitos, todos os da CP

Modelagem – Comentários

- A fase de busca e propagação do comando `solve(Critérios, Variáveis)`, há dezenas de combinações possíveis: consultar o guia do usuário
- Tem-se os predicados extras ... são muitos, todos os da CP
- Finalmente, exemplos sofisticados– de PR com PICAT: `http://www.hakank.org/picat/` – ***My Picat page*** – por Hakan Kjellerstrand

- Acompanhar as explicações do código de: https://github.com/claudiosa/CCS/blob/master/picat/horario_medico_CP.pi
- Confira a execução e testes

```
modelo =>
    Dias = 5, % segunda= 1, ...., sexta-feira = 5
    Consultorio = 4,
    L_dom = [ oftalmo, otorrino, pediatra,  gineco,
%                 1        2           3         4
              cardio, dermato, clin_geral ],
%                 5       6        7
   Quadro = new_array(Consultorio, Dias ), %% Lin x Col
   Quadro :: 1 .. L_dom.len , %% operador len . "eh colado"
...
```

```
%% O medico 2 NUNCA trabalha no consultorio 1
foreach ( J in 1 .. Dias )
    Quadro[1,J] #!= 2
end,

%% O medico 5 NUNCA trabalha no consultorio 4
foreach ( J in 1 .. Dias )
    Quadro[4,J] #!= 5
end,
...
```

Código em Partes

```
%% O Clin Geral deve vir o maior numero de dias ...
%% Esta restricao eh matematicamente é HARD
 foreach ( I in 1 .. Consultorio )
   member(7,[Quadro[I,J] : J in 1..Dias])
 end,

%% Ninguém trabalha no mesmo consultorio em dias seguidos
foreach ( J in 1 .. Dias )
    all_different( [Quadro[I,J] : I in 1..Consultorio] )
end,

%% Ninguém trabalha no mesmo dia em mais de um consultorio
 foreach ( I in 1 .. Consultorio )
    all_different( [Quadro[I,J] : J in 1..Dias] )
 end,
...
```

Código em Partes

```
% A BUSCA
solve([ff], Quadro),
  % UMA SAIDA

   printf("\n Uma escolha:"),
   print_matrix( Quadro ),
   print_matrix_NAMES( Quadro , L_dom ),
 printf(".............................\n") .
```

```
print_matrix_NAMES( M, Lista ) =>
 L = M.length,
 C = M[1].length,
  nl,
   foreach(I in 1  .. L)
     foreach(J in 1  ..  C)
      printf(":%w \t" , print_n_lista( M[I,J], Lista) )
     % printf("(%d,%d): %w " , I, J, M[I,J] ) -- FINE
     end,
     nl
   end.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
print_n_lista( _, [] ) =  [].
print_n_lista( 1, [A|_] ) = A.
print_n_lista( N, [_|B] ) = print_n_lista( (N-1), B ) .
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Saída - I

```
Picat> cl('horario_medico_CP.pi').
Compiling:: horario_medico_CP.pi
horario_medico_CP.pi compiled in 10 milliseconds
loading...

yes

Picat> main

 Uma escolha:
7 1 3 4 5
4 7 2 3 1
1 3 7 5 2
3 2 1 7 4
```

Saída - II

```
:clin_geral  :oftalmo  :pediatra  :gineco  :cardio
:gineco  :clin_geral  :otorrino  :pediatra  :oftalmo
:oftalmo  :pediatra  :clin_geral  :cardio  :otorrino
:pediatra  :otorrino  :oftalmo  :clin_geral  :gineco
...................................
yes
```

Saída - III

```
$ time(picat horario_medico_CP.pi )

 Uma escolha:
7 1 3 4 5
4 7 2 3 1
1 3 7 5 2
3 2 1 7 4

:clin_geral  :oftalmo  :pediatra  :gineco  :cardio
:gineco  :clin_geral  :otorrino  :pediatra  :oftalmo
:oftalmo  :pediatra  :clin_geral  :cardio  :otorrino
:pediatra  :otorrino  :oftalmo  :clin_geral  :gineco
....................................
real 0m0,023s
user 0m0,007s
sys 0m0,013s
[ccs@gerzat picat]$
```

Exemplo – 03 – Caixeiro-Viajante

- Este é um exemplo clássico ⇒ um NP-Completo ⇒ boas soluções apenas com Colônia de Formigas (técnica da Computação Evolucionária)

Exemplo – 03 – Caixeiro-Viajante

- Este é um exemplo clássico ⇒ um NP-Completo ⇒ boas soluções apenas com Colônia de Formigas (técnica da Computação Evolucionária)
- Neste exemplo o Problema do Caixeiro-Viajante (do inglês: TSP – *Travelling Salesman Problem*) é discutido com **dois modelos da PR** destacando dois pontos desta técnica:
  - Uso de outras restrições globais: `element` e `circuit`
  - Uso de variáveis de decisão binária, neste exemplo, uma matriz binária.

Exemplo – 03 – Caixeiro-Viajante

- Este é um exemplo clássico ⇒ um NP-Completo ⇒ boas soluções apenas com Colônia de Formigas (técnica da Computação Evolucionária)
- Neste exemplo o Problema do Caixeiro-Viajante (do inglês: TSP – *Travelling Salesman Problem*) é discutido com **dois modelos da PR** destacando dois pontos desta técnica:
  - Uso de outras restrições globais: `element` e `circuit`
  - Uso de variáveis de decisão binária, neste exemplo, uma matriz binária.
- Tecnicamente, este problema tem muitas aplicações similares!

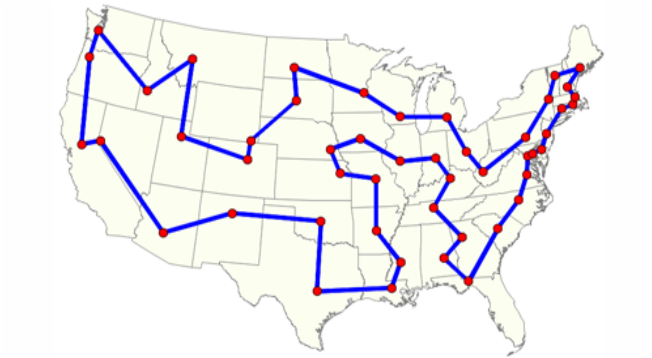


Figura 3: Passar por algumas cidades uma única vez e retornar a cidade de origem

- Usaremos a modelagem matemática de uma *matriz binária de decisão*

Modelagem – Comentários

- Usaremos a modelagem matemática de uma *matriz binária de decisão*
- Esta abordagem é bem conhecida, porém, e eficiente para alguns tipos de problemas semelhantes

Modelagem – Comentários

- Usaremos a modelagem matemática de uma *matriz binária de decisão*
- Esta abordagem é bem conhecida, porém, e eficiente para alguns tipos de problemas semelhantes
- Idéia da *matriz binária de decisão*: N cidades, logo, há possíveis conexões entre elas.

Modelagem – Comentários

- Usaremos a modelagem matemática de uma *matriz binária de decisão*
- Esta abordagem é bem conhecida, porém, e eficiente para alguns tipos de problemas semelhantes
- Idéia da *matriz binária de decisão*: N cidades, logo, há possíveis conexões entre elas.
- Comentários desta modelagem ao longo do código e detalhes do formalismo matemático em qualquer livro da área de combinatória, PO, e modelagem matemática

Modelagem – Comentários

- Usaremos a modelagem matemática de uma *matriz binária de decisão*
- Esta abordagem é bem conhecida, porém, e eficiente para alguns tipos de problemas semelhantes
- Idéia da *matriz binária de decisão*: N cidades, logo, há possíveis conexões entre elas.
- Comentários desta modelagem ao longo do código e detalhes do formalismo matemático em qualquer livro da área de combinatória, PO, e modelagem matemática
- Contudo, duas novas restrições globais são usadas e precisam ser entendidas: `element` e `circuit`

Restrição global: *element*

```
element_ex(Vars)  =>
    X :: 1..4, %% NUM de indices da lista
    element(X , [22, 33, 44, 55], Index),
    Vars = [X , Index],
    solve(Vars).
```

Restrição global: *element*

```
exe04 =>
    Todas_Sol = findall(Uma_Sol , $element_ex(Uma_Sol)),
    foreach( X in Todas_Sol )
      printf("\n Sol %w", X)
    end,
    printf("\n Total de SOL: %d", Todas_Sol.len).
```

Restrição global: *element*

```
exe04 =>
    Todas_Sol = findall(Uma_Sol , $element_ex(Uma_Sol)),
    foreach( X in Todas_Sol )
      printf("\n Sol %w", X)
    end,
    printf("\n Total de SOL: %d", Todas_Sol.len).
```

Saída:

```
 Sol [1,22]
 Sol [2,33]
 Sol [3,44]
 Sol [4,55]
 Total de SOL: 4
```

Restrição global: *circuit*

```
circ_ex(L) =>
    L = [X1,X2,X3,X4],
    L :: 1..4, %% NUM de indices da lista
    circuit(L),
    solve(L).
```

Restrição global: *circuit*

```
exe02 =>
    Todas_Sol = findall( Uma_Sol , $circ_ex(Uma_Sol)),
    foreach( X in Todas_Sol )
      printf("\n Sol %w", X)
    end,
    printf("\n Total de SOL: %d", Todas_Sol.len).
```

Restrição global: *circuit*

```
exe02 =>
    Todas_Sol = findall( Uma_Sol , $circ_ex(Uma_Sol)),
    foreach( X in Todas_Sol )
      printf("\n Sol %w", X)
    end,
    printf("\n Total de SOL: %d", Todas_Sol.len).
```

Saída:

```
 Sol [2,3,4,1]
 Sol [2,4,1,3]
 Sol [3,1,4,2]
 Sol [3,4,2,1]
 Sol [4,1,2,3]
 Sol [4,3,1,2]
 Total de SOL: 6
```

Código Completo

- Acompanhar as explicações do 1º. modelo: `https://github.com/claudiosa/CCS/blob/master/picat/tsp_ESTUDO_hakan.pi`
- Acompanhar as explicações do 2º. modelo: `https://github.com/claudiosa/CCS/blob/master/picat/tsp_CP.pi`
- Confira a execução e testes

1º. Modelo para o TSP – Nilsson e Hakan

```
% Original formulation from Nilsson cited above.
% Codifificado por HAKAN e CCS
tsp_test(nilsson, Cidades, Custo) =>
   Cidades   = [X1,X2,X3,X4,X5,X6,X7],
   %% a matriz adjacencia - do mapa - 7 cidades
   element(X1,[ 0, 4, 8,10, 7,14,15],C1),
   element(X2,[ 4, 0, 7, 7,10,12, 5],C2),
   element(X3,[ 8, 7, 0, 4, 6, 8,10],C3),
   element(X4,[10, 7, 4, 0, 2, 5, 8],C4),
   element(X5,[ 7,10, 6, 2, 0, 6, 7],C5),
   element(X6,[14,12, 8, 5, 6, 0, 5],C6),
   element(X7,[15, 5,10, 8, 7, 5, 0],C7),
   Custo #= C1+C2+C3+C4+C5+C6+C7 ,
   circuit( Cidades ) ,
   solve([$min(Custo)], Cidades).
```

Saída

```
$ picat tsp_ESTUDO_hakan.pi
Cidades: [2,7,1,3,4,5,6] Custo: 34
 A viagem:
Da cidade 1 --> 2 custa: 4  Acumulado: 4
Da cidade 2 --> 7 custa: 5  Acumulado: 9
Da cidade 7 --> 6 custa: 5  Acumulado: 14
Da cidade 6 --> 5 custa: 6  Acumulado: 20
Da cidade 5 --> 4 custa: 2  Acumulado: 22
Da cidade 4 --> 3 custa: 4  Acumulado: 26
Da cidade 3 --> 1 custa: 8  Acumulado: 34
```

Saída

```
$ picat tsp_ESTUDO_hakan.pi
Cidades: [2,7,1,3,4,5,6] Custo: 34
 A viagem:
Da cidade 1 --> 2 custa: 4  Acumulado: 4
Da cidade 2 --> 7 custa: 5  Acumulado: 9
Da cidade 7 --> 6 custa: 5  Acumulado: 14
Da cidade 6 --> 5 custa: 6  Acumulado: 20
Da cidade 5 --> 4 custa: 2  Acumulado: 22
Da cidade 4 --> 3 custa: 4  Acumulado: 26
Da cidade 3 --> 1 custa: 8  Acumulado: 34
```

- A importância deste modelo: facilmente se entende o TSP

Saída

```
$ picat tsp_ESTUDO_hakan.pi
Cidades: [2,7,1,3,4,5,6] Custo: 34
 A viagem:
Da cidade 1 --> 2 custa: 4  Acumulado: 4
Da cidade 2 --> 7 custa: 5  Acumulado: 9
Da cidade 7 --> 6 custa: 5  Acumulado: 14
Da cidade 6 --> 5 custa: 6  Acumulado: 20
Da cidade 5 --> 4 custa: 2  Acumulado: 22
Da cidade 4 --> 3 custa: 4  Acumulado: 26
Da cidade 3 --> 1 custa: 8  Acumulado: 34
```

- A importância deste modelo: facilmente se entende o TSP
- Hakan fez uma versão genérica para este modelo, de bom desempenho!

Saída

```
$ picat tsp_ESTUDO_hakan.pi
Cidades: [2,7,1,3,4,5,6] Custo: 34
 A viagem:
Da cidade 1 --> 2 custa: 4  Acumulado: 4
Da cidade 2 --> 7 custa: 5  Acumulado: 9
Da cidade 7 --> 6 custa: 5  Acumulado: 14
Da cidade 6 --> 5 custa: 6  Acumulado: 20
Da cidade 5 --> 4 custa: 2  Acumulado: 22
Da cidade 4 --> 3 custa: 4  Acumulado: 26
Da cidade 3 --> 1 custa: 8  Acumulado: 34
```

- A importância deste modelo: facilmente se entende o TSP
- Hakan fez uma versão genérica para este modelo, de bom desempenho!
- O 2º. modelo tem importância como técnica para PR!

2º. Modelo para o TSP – Usando Matriz de Decisão

```
import cp,util.
matriz_adj(Matrix) =>
   Matrix =
       [[ 0, 4, 8,10, 7,14,15],
        [ 4, 0, 7, 7,10,12, 5],
        [ 8, 7, 0, 4, 6, 8,10],
        [10, 7, 4, 0, 2, 5, 8],
        [ 7,10, 6, 2, 0, 6, 7],
        [14,12, 8, 5, 6, 0, 5],
        [15, 5,10, 8, 7, 5, 0]].
```

- Os dados são os mesmos do exemplo anterior
- Poderia ser feita leitura via arquivos: ver exemplos de entrada e saída no github
- Comentários no código e aúdio

Código em Partes

```
tsp_D(Matriz, Cidades, M_Decisao, Custo) =>
   Len = Matriz.length, %% N x N cidades
   Cidades = new_list(Len), %%% 1a. dimensao
   Cidades :: 1..Len,
   % grafo de DECISAO que representa o resultado dos nos escolhidos
   M_Decisao = new_array (Len, Len),
   M_Decisao :: 0..1 ,
```

Código em Partes

```
tsp_D(Matriz, Cidades, M_Decisao, Custo) =>
   Len = Matriz.length, %% N x N cidades
   Cidades = new_list(Len), %%% 1a. dimensao
   Cidades :: 1..Len,
   % grafo de DECISAO que representa o resultado dos nos escolhidos
   M_Decisao = new_array (Len, Len),
   M_Decisao :: 0..1 ,

%  calculate upper and lower bounds of the Costs list -- HAKAN
   % repensar MELHORAR .....
   SOMA_Dists = sum([Matriz[I,J] : I in 1..Len,
                 J in 1..Len, Matriz[I,J] > 0]),
   MinDist = 0,
   MaxDist = SOMA_Dists,
   Custo :: 0..MaxDist,
```

```
% Se NAO HOUVER CONEXAO ou  ARCO = 0 entao não há conexão
  foreach(I in 1..Len , J  in 1..Len)
    (Matriz[I,J] #= 0) #=> (M_Decisao[I,J] #= 0)
  end,
```

Código em Partes

```
% Se NAO HOUVER CONEXAO ou  ARCO = 0 entao não há conexão
  foreach(I in 1..Len , J  in 1..Len)
    (Matriz[I,J] #= 0) #=> (M_Decisao[I,J] #= 0)
  end,

% Para todas linhas, a soma das colunas é igual a 1
% UMA: uma saida como caminho a ser traçado e somente UMA
  foreach(I in 1..Len)
    sum([M_Decisao[I,J] : J in 1..Len, I != J]) #= 1
  end,
```

Código em Partes

```
% Se NAO HOUVER CONEXAO ou  ARCO = 0 entao não há conexão
  foreach(I in 1..Len , J  in 1..Len)
    (Matriz[I,J] #= 0) #=> (M_Decisao[I,J] #= 0)
  end,

% Para todas linhas, a soma das colunas é igual a 1
% UMA: uma saida como caminho a ser traçado e somente UMA
  foreach(I in 1..Len)
    sum([M_Decisao[I,J] : J in 1..Len, I != J]) #= 1
  end,

% Para todas colunas, a soma das linhas é igual a 1
% UMA: uma chegada ao nó de destino e somente UMA chegada
  foreach(J in 1..Len)
   sum([M_Decisao[I,J] : I in 1..Len, I != J]) #= 1
  end,
```

Código em Partes

```
%% Relacionar as escolhas da M_Decisao com a
%%  ... sequencia das Cidades.
  foreach(I in 1..Len , J  in 1..Len)
    ( M_Decisao[I,J] #= 1 ) #<=> ( Cidades[I] #= J )
  end,

  %% garante o circuito entre os nós selecionados
  circuit(Cidades),
```

Código em Partes

```
%% Relacionar as escolhas da M_Decisao com a
%%  ... sequencia das Cidades.
  foreach(I in 1..Len , J  in 1..Len)
    ( M_Decisao[I,J] #= 1 ) #<=> ( Cidades[I] #= J )
  end,

  %% garante o circuito entre os nós selecionados
  circuit(Cidades),


%% Função custo a ser minimizada
 Custo #= sum([M_Decisao[I,J] * Matriz[I,J] :
             I in 1..Len , J  in 1..Len]),

 %% Vars para BUSCA
 Vars = [Cidades, M_Decisao], %% OU  Cidades ++ M_Decisao
 solve([$min(Custo)], Vars ).
```

```
$ picat tsp_CP.pi
M_Decisao: {{0,1,0,0,0,0,0},{0,0,0,0,0,0,1},{1,0,0,0,0,0,0},{0,0,1,0,0,
DESTINOS:
      1 2 3 4 5 6 7
 1 -> 0 1 0 0 0 0 0
 2 -> 0 0 0 0 0 0 1
 3 -> 1 0 0 0 0 0 0
 4 -> 0 0 1 0 0 0 0
 5 -> 0 0 0 1 0 0 0
 6 -> 0 0 0 0 1 0 0
 7 -> 0 0 0 0 0 1 0
Sequência das Cidades: [2,7,1,3,4,5,6]
Custo: 34

 A viagem:
Da cidade 1 --> 2 custa: 4  Acumulado: 4
..........................................
Da cidade 3 --> 1 custa: 8  Acumulado: 34
```

As funções/regras de saída não foram apresentadas!

- Há outros métodos para se resolver estes problemas. Exemplo: Programação Linear, Buscas Heurísticas, AGs, Busca Gulosa etc

Resumindo a PR

- Há outros métodos para se resolver estes problemas. Exemplo: Programação Linear, Buscas Heurísticas, AGs, Busca Gulosa etc
- As restrições globais se aplicam sobre um conjunto de variáveis e há muitas outras importantes disponíveis no Picat

Resumindo a PR

- Há outros métodos para se resolver estes problemas. Exemplo: Programação Linear, Buscas Heurísticas, AGs, Busca Gulosa etc
- As restrições globais se aplicam sobre um conjunto de variáveis e há muitas outras importantes disponíveis no Picat
- A área é extensa e Picat adere há todos requisitos da PR

Resumindo a PR

- Há outros métodos para se resolver estes problemas. Exemplo: Programação Linear, Buscas Heurísticas, AGs, Busca Gulosa etc
- As restrições globais se aplicam sobre um conjunto de variáveis e há muitas outras importantes disponíveis no Picat
- A área é extensa e Picat adere há todos requisitos da PR
- Resumo da PR: segue por uma notação/manipulação algébrica restrita, simplificar e bissecionar as restrições, instanciar variáveis, verificar inconsistências, avançar sobre as demais variáveis, até que todas estejam instanciadas.

Resumindo a PR

- Há outros métodos para se resolver estes problemas. Exemplo: Programação Linear, Buscas Heurísticas, AGs, Busca Gulosa etc
- As restrições globais se aplicam sobre um conjunto de variáveis e há muitas outras importantes disponíveis no Picat
- A área é extensa e Picat adere há todos requisitos da PR
- Resumo da PR: segue por uma notação/manipulação algébrica restrita, simplificar e bissecionar as restrições, instanciar variáveis, verificar inconsistências, avançar sobre as demais variáveis, até que todas estejam instanciadas.
- Enfim, agora é o momento de praticar e aprimorar os conhecimentos $\Rightarrow$ Bons códigos!