Grupo Nº 38



**Inteligência Artificial**

1.º Semestre 2014/2015

**Fill-a-Pix**

Relatório de Projecto

Índice

[1 Implementação Tipos e Representação Problema PSR 2](#_Toc404722800)

[1.1 Tipos Abstractos de Informação 2](#_Toc404722801)

[1.2 Representação do problema Fill-a-Pix como PSR 3](#_Toc404722802)

[2 Implementação Procuras e Funções Obrigatórias 4](#_Toc404722803)

[2.1 Fill-a-pix🡪psr 4](#_Toc404722804)

[2.2 Psr->Fill-a-pix 4](#_Toc404722805)

[2.3 Heurística de Grau 4](#_Toc404722806)

[2.4 Heurística MRV 4](#_Toc404722807)

[2.5 Procura-Retrocesso e Inferência 4](#_Toc404722808)

[3 Optimizações, Heurísticas e Técnicas adicionais utilizadas 5](#_Toc404722809)

[3.1 Optimizações especificas para o problema Fill-a-Pix 5](#_Toc404722810)

[3.2 Criação/Combinação de Heurísticas 5](#_Toc404722811)

[3.3 Utilização de técnicas adicionais 5](#_Toc404722812)

[4 Estudo Comparativo 6](#_Toc404722813)

[4.1 Critérios a analisar 6](#_Toc404722814)

[4.2 Testes Efectuados 6](#_Toc404722815)

[4.3 Resultados Obtidos 6](#_Toc404722816)

[4.4 Comparação dos Resultados Obtidos 6](#_Toc404722817)

[4.5 Escolha do resolve-best 6](#_Toc404722818)

# Implementação Tipos e Representação Problema PSR

## Tipos Abstractos de Informação

Neste ponto devem ser abordados, pelo menos, os seguintes tópicos:

•Descrição detalhada das estruturas de dados utilizadas no programa.

•Justificação das escolhas efectuadas em relação a possíveis alternativas e às operações implementadas.

Se implementarem mais do que um tipo psr para o projecto, deverão descrever os vários tipos psr implementados e justificar a sua implementação.

**Restrição**

O tipo restrição é representado por uma estrutura composta por 2 campos:

1. Campo guarda a lista de variáveis envolvidas na restrição.
2. Campo guarda o predicado que é usado para verificar se a restrição está ou não a ser cumprida.

**Var**

Tipo abstracto que guarda informação sobre cada variável do PSR:

1. Campo guarda o nome da variável *String*.
2. Campo que guarda o valor atribuído a essa variável.
3. Campo que guarda uma lista correspondente ao domínio da variável.

**PSR**

O tipo PSR é representado por uma estrutura composta por 3 campos:

1. Campo que guarda uma hash-table em que a chave é o nome da variável *String* e o valor para a entrada é a estrutura Var que representa essa variável.
2. Campo que guarda uma lista dos nomes das variáveis pela mesma ordem usada quando se constrói o psr.
3. Campo que guarda a lista de restrições pela mesma ordem que é fornecida ao construtor do psr.

No início da implementação efetuou-se a implementação do PSR de uma maneira mais simples. Em vez de usar uma hash-table para guardar as estruturas **Var** usou-seuma lista, mas para aumentar a velocidade dos acessos do tipo aceder unicamente aos atributos de uma variável (ex: psr-altera-dominio! ), em vez de ter que percorrer a lista para encontrar a estrutura **Var** correspondente os acessos pela hash são praticamente instantâneos.

A lista de variáveis guardada no 2º campo do PSR serve para conservarmos a ordem original, para quando for preciso devolver lista em que a ordem tem que respeitar a original podemos usar esta como auxiliar, é um *tradeoff* de alguma memória por tempo nos acessos.

Ainda que não implementada podíamos usar ter uma lista na estrutura **Var** com um índice para a lista de restrições para aumentar a velocidade de acesso às restrições de cada variável, para evitar a procura exaustiva da lista de restrições. Assim temos que percorrer a lista toda de restrições para encontrar as respectivas de uma determinada variável.

## Representação do problema Fill-a-Pix como PSR

Nesta secção deverão descrever como é que decidiram representar o puzzle Fill-a-Pix como um problema de Satisfação de Restrições. Ou seja, deverão indicar as variáveis, os domínios, e as restrições escolhidas para representar um puzzle Fill-a-Pix. Deverão também justificar a escolha efectuada, comparando com possíveis alternativas de representação.

**Variáveis**

As variáveis correspondentes a cada quadrícula do puzzle são representadas por uma *string* no formato “L C” em que L corresponde á linha e C á coluna respectiva da quadrícula, deste modo é fácil transformar a *string* em dois números para indexar a quadrícula.

**Domínios**

Os domínios correspondem a uma lista de valores no caso do Fill-a-Pix os valores possível são 0 ou 1, 0 caso a quadricula não deva ser pintada e preto caso contrário.

**Restrições**

Ao percorrer o tabuleiro Fill-a-Pix as quadrículas que tiverem **NIL** como valor são ignoradas as que tiverem um valor diferente de **NIL,** (0 – 9 ) nessa altura é criado um restrição correspondente com as variáveis correspondentes e um predicado dependente do valor da quadricula.

Para implementarmos as restrições usámos 3 predicados de validação distintos, todos os tipos de predicado aquando da sua criação guardam a lista de variáveis a que correspondem:

* O predicado **pred-9** é usado para quando se tem a certeza que todas as variáveis dessa restrição devem ter obrigatoriamente todos os valores iguais a 1. Ao chamar este predicado caso uma variável tenha o valor 0 ele devolve imediatamente **NIL** caso só encontre **NIL’s** e 1’s ele devolve **T** .
  + Qualquer quadrícula que tenha o valor 9 deverá criar um **pred-9.**
  + As quadrículas nas bordas do tabuleiro à excepção dos cantos caso tenham o valor 6 também deve ser criado um **pred-9** porque sabemos que as 6 variáveis correspondentes às quadrículas que envolvem devem ter o valor 1.
  + As quadrículas nos cantos do puzzle que tiverem o valor 4 também devem criar um **pred-9**

Porque no canto devido às regras do Fill-a-Pix a quadrícula só afecta as 3 quadrículas em redor dessa mais ela própria.

* O predicado **pred-0** é usado quando se sabe que todas as variáveis dessa restrição devem ter obrigatoriamente 0 como valor, ou seja quando se encontra uma quadrícula com o número 0 deve-se criar um **pred-0.** Ao chamar este predicado caso uma variável tenha o valor 1 ele devolve imediatamente **NIL** caso só encontre **NIL’s** e 0’s ele devolve **T**.
* O predicado **pred-geral** é usado em todas as outras situações que não são abrangidas pelo **pred-0** e pelo **pred-1**. Este predicado tem algumas particularidades, para além de guardar não só a lista de variáveis guarda também o valor da quadrícula. Ao chamar este predicado ele conta o número de variáveis não atribuídas e o número de variáveis atribuídas do seu grupo de variáveis, caso o número de variáveis não atribuídas mais o número de 1 seja igual ou superior ao valor da quadrícula (que gerou a restrição) e a contagem das variáveis com valor 1 não ultrapasse o valor da quadrícula ele devolve **T** caso uma das condições anteriores falhe ele retorna **NIL**.

# Implementação Procuras e Funções Obrigatórias

Nesta secção deverão descrever a implementação das várias funções/heurísticas e mecanismos de inferência pedidos explicitamente no enunciado. No entanto, deverão apenas focar-se na descrição de particularidades que não estejam especificadas no documento de esclarecimento. Ou seja, **não queremos ler o pseudocódigo e respectivos comentários** criados por nós. Em particular devem descrever as seguintes funções/algoritmos:

## Fill-a-pix🡪psr

Deverão descrever brevemente o algoritmo de conversão de um tabuleiro Fill-a-Pix para o tipo PSR. Devido à sua complexidade, é importante descrever o processo de criação e teste de restrições.

## Psr->Fill-a-pix

Aqui deverão descrever o algoritmo oposto, i.e. a conversão de um PSR resolvido para um tabuleiro Fill-a-Pix.

## Heurística de Grau

Deverão descrever brevemente como é que implementaram a heurística de grau.

## Heurística MRV

Deverão descrever brevemente como é que implementaram a heurística MRV.

## Procura-Retrocesso e Inferência

Nesta secção deverão descrever algum detalhe que considerem relevante na implementação das procuras por retrocesso com os vários tipos de inferência. Mais uma vez, aqui não é pretendido que façam copy-paste do pseudocódigo, mas sim que descrevam algum pormenor que achem importante. Por exemplo, o que usaram para representar uma inferência, e como é que essa é aplicada/removida ao psr.

# Optimizações, Heurísticas e Técnicas adicionais utilizadas

Neste capítulo deverão descrever com algum detalhe 3 tipos de estratégias usadas para tornar o vosso algoritmo de procura mais eficiente.

## Optimizações especificas para o problema Fill-a-Pix

Nesta secção deverão descrever optimizações efectuadas nas funções de modo a tornar a procura por retrocesso mais eficiente para um puzzle especifico do Fill-a-Pix. Podem também descrever outros tipos de optimizações efectuadas ao pseudocódigo original fornecido. Por exemplo, se não houver a restrição de que as coisas têm que ser feitas por uma determinada ordem, é possível criar versões ligeiramente mais eficientes da função vizinhos-não-atribuidos, da função psr-variaveis-não-atribuidas, etc.

## Criação/Combinação de Heurísticas

Se criarem alguma heurística adicional para a escolha de variável/valor, ou se usarem alguma combinação de heurísticas deverão descrevê-las nesta secção.

## Utilização de técnicas adicionais

Nesta secção final deverão descrever outras técnicas adicionais utilizadas para tentar tornar o algoritmo de procura mais eficiente. Alguns exemplos (que podem funcionar ou não) são:

* Conflict-directed backjumping.
* Execução algoritmo MR3 antes de iniciar a procura.
* Utilização de restrições globais para testes de consistência e inferência.

# Estudo Comparativo

Pretende-se estudar a eficácia dos vários algoritmos e variantes implementados para a resolução de um problema Fill-a-Pix.

## Critérios a analisar

Descrever quais os critérios a ser usados para comparar as várias variantes e justificar a sua escolha.

## Testes Efectuados

Descrever os problemas utilizados para efectuar o estudo. Justificar a sua *relevância* para os critérios a comparar.

## Resultados Obtidos

Enumerar os resultados obtidos, sob a forma mais adequada (tabela e gráficos, se necessário).

## Comparação dos Resultados Obtidos

Analisar, caso a caso, os resultados, focando a diferença de desempenho das várias variantes entre si. A análise não deve ser somente uma descrição dos resultados obtidos, mas sim uma análise dos factores que possam ter conduzido a esses resultados.

## Escolha do resolve-best

Com base nos resultados obtidos na comparação anterior, devem descrever e justificar quais as variantes escolhidas para implementar a vossa função resolve-best.