Grupo Nº _42_

TÉCNICO LISBOA

Inteligência Artificial

1.º Semestre 2014/2015

Fill-a-Pix

Relatório de Projecto

Índice

1 Implementação Tipos e Representação Problema PSR

1.1 Tipos Abstractos de Informação

Para a implementação do tipo Restrição usou-se apenas uma lista porque era um tipo bastante simples só com dois elementos.

Para implementarmos o PSR, utilizámos uma lista para guardar os domínios de cada variável e outra lista para as várias restrições do problema. Utilizámos uma hash-table em que a chave é uma variável para guardar um par com a posição do domínio dessa variável na lista de domínios e o segundo elemento do par era o valor atribuído à variável. Utilizámos ainda outra hash-table em que a chave é novamente a variável e o valor guardado é uma lista com as restrições em que essa variável está inserida.

Relativamente ao uso de listas, utilizou-se esta estrutura em detrimento, por exemplo de vectores por serem de fácil utilização e manipulação e aplicação de funções como mapcar e reverse.

Foi escolhida uma hash-table para se ter um acesso rápido utilizando as variáveis como chave em vez de um vector ou lista de modo a evitar percorrer-los à procura do elemento pretendido.

1.2 Representação do problema Fill-a-Pix como PSR

Para as variáveis, o nome escolhido é uma string com um inteiro entre 0 e N*M-1 em que o N e o M são, o número de colunas e linhas da matriz do jogo. O domínio das variáveis é o intervalo entre 0 e 1 em que 0 representa uma posição da área de jogo que não está pintada, e a quando a variável tem o valor 1 é porque está pintada. O domínio é representado por uma lista com elementos, neste caso os inteiros 0 e 1.

Um predicado fill-a-pix verifica se o número de casas preenchidas é superior ao valor pretendido (mesmo que algumas variáveis não estejam atribuídas) e o mesmo com o número de zeros, sendo o resultado então falso. Se não, devolve verdade se alguma variável não está atribuída e soma todas as variáveis, testando se o resultado é igual ao valor pretendido.

2 Implementação Procuras e Funções Obrigatórias

2.1 Fill-a-pix→psr

Começa-se por criar todas as variáveis do psr. Um fill-a-pix tem n*m variáveis (linha * coluna). Para dar um nome único a cada variável, escolheu-se numerar as posições de 0 a (n*m -1).

Para preencher o domínio, fez-se uma lista com n*m elementos com domínio 0. Há medida que se encontra números no fill-a-pix, atualiza-se o domínio correto (0 1). Para isto foi necessário a implementação de um algoritmo que descobre todas as posições adjacentes validas. Este algoritmo cria uma lista com as 9 posições adjacentes e elimina as posições inválidas a seguir (posição negativa/posição que excede o numero máximo de linhas ou colunas).

Finalmente, para criar as restrições usou-se as mesmas posições adjacentes calculadas no domínio para as variáveis afetadas. Por fim, chegou-se à conclusão que um predicado fill-a-pix teria de verificar se o número de casas preenchidas é superior ao valor pretendido (mesmo que algumas variáveis não fossem atribuídas) e o mesmo com o número de 0s, sendo o resultado então falso. Se não, devolve verdade se alguma variável não está atribuída e soma todas as variáveis, testando se o resultado é igual ao valor pretendido.

2.2 Psr->Fill-a-pix

Este algoritmo percorre as variáveis todas do fill-a-pix e preenche um array de dimensão linhas*colunas. A primeira variável corresponde à posição (0 0) e a segunda à posiçao (0 1). Percorre-se então as variáveis todas do psr, e após ter preenchido m (numero de colunas) variáveis, incrementa-se o valor da linha.

Isto repete-se até o array estar preenchido.

2.3 Heurística de Grau

Heurística-grau deve devolver a variável com grau maior, ou seja, a variável com mais restrições comuns com as outras variáveis não atribuídas.

Para tal, percorre uma lista de variáveis não atribuídas e procura por restrições comuns com todas as outras variáveis não atribuídas, devolvendo a variável com mais restrições em comum.

2.4 Heurística MRV

Herustica-mrv deve devolver a variável não atribuída com domínio menor. Este algoritmo percorre então a lista de variáveis não atribuídas e devolve a variável com menor elementos.

2.5 Procura-Retrocesso e Inferência

Criou-se um tipo de dados para representar a inferência. O tipo escolhido foi uma lista de dois elementos. Sendo o primeiro elemento a variável e o segundo o domínio. Escolheu-se uma lista e não um par pois tendo um par com uma lista no segundo elemento seria equivalente a ter uma lista apenas, o que não era o pretendido.

Quando o algoritmo de procura recebe inferências do algoritmo mac ou fc, este cria e guarda todos os domínios (em forma de inferência) que vão ser alterados para os poder repor mais tarde. Se não conseguir devolver um psr completo, repõe os domínios de origem.

3 Optimizações, Heurísticas e Técnicas adicionais utilizadas

3.1 Optimizações especificas para o problema Fill-a-Pix

Pensou-se em implementar uma optimização para o resolve-best uma modificação no algoritmo fill-a-pix->psr em que caso a soma dos valores das variáveis necessárias for igual ao número de variáveis desse predicado, sabe-se que todas essas casas vão ter o valor 1. Como tal, o domínio dessas variáveis não é um intervalo entre 0 e 1 mas sim 1. Assim, não é necessário criar um predicado para essa restrição. No entanto não chegámos a implementar esta funcionalidade.

3.2 Criação/Combinação de Heurísticas

Não se implementou nenhuma herurística para além das listadas no enunciado.

3.3 Utilização de técnicas adicionais

Na procura-retrocesso-best fez-se apenas uma pequena modificação na procura-retrocesso-mrv em que se o domínio da variável escolhida tiver apenas um elemento, ao invés de fazer um teste de consistência, atribui-se logo esse valor à variável.

4 Estudo Comparativo

4.1 Critérios a analisar

Os critérios utilizados são o número de testes de consistência realizados para resolver o problema, o tempo que leva a terminar e a memória utilizada pelo algoritmo. Considera-se que estes três critérios são necessários para avaliar a eficiência do algoritmo, sendo que quanto menores forem os seus valores, melhor será a sua eficiência. No entanto, dá-se mais importância ao número de testes que o algoritmo faz.

4.2 Testes Efectuados

Para comparar a procura-retrocesso-simples com a procura retrocesso-grau utilizámos os testes “e1_1” e “e1_2” presentes no ficheiro “exemplo.lisp”.

4.3 Resultados Obtidos

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

Algoritmo	Problema	Testes	Tempo (s)	Espaço (KB)
PRS	E1_1	69	0,002	107
PRG	E1_1	77	0,006	524
PRS	E1_2	382	0,003	182
PRG	E1_2	59	0,007	318

Algoritmo	Problema	Testes	Tempo (s)	Espaço (KB)
FC	E4	4301	0,08	2434
MAC	E4	8577	0,15	4134
FC	E7	568777	30,36	171609,3
MAC	E7	901330	56,3	290638,78

4.4 Comparação dos Resultados Obtidos

Pode-se observar que nos algoritmos simples e grau há um caso em que o número de testes efetuados é inferior usando o simples. No entanto, no exemplo a seguir o algoritmo grau faz 323 passos a menos. Percebe-se, então, que a implementação simples depende maioritariamente do tabuleiro fill-a-pix, e nos casos piores vai fazer muito mais testes. O procura-retrocesso-grau, apesar de em certos casos fazer mais testes do que o simples, não depende tanto de como o tabuleiro está feito, fazendo na média menos testes do que o simples.

4.5 Escolha do resolve-best

Com base nos resultados, o algoritmo FC seria a melhor escolha para usar na procura resolve-best.