Inteligência Artificial

Projecto

5 de Novembro de 2010

1 Introdução

O objectivo deste projecto é estudar formas de resolver problemas que correspondem a uma generalização de problemas Sudoku, utilizando algumas das técnicas estudadas em Inteligência Artificial.

2 O jogo

Dada uma grelha 9×9 parcialmente preenchida com inteiros entre 1 e 9, o objectivo do jogo é preencher completamente a grelha de forma a que cada um dos dígitos ocorra uma e uma só vez em cada linha, coluna e bloco (por bloco entende-se cada uma das 9 sub-grelhas 3×3).

Por exemplo, o tabuleiro representado na Figura 1 ilustra um tabuleiro Sudoku parcialmente preenchido e o tabuleiro representado na Figura 2 representa o mesmo tabuleiro, agora completamente preenchido.

O problema que pretendemos resolver corresponde a uma generalização dos problemas Sudoku uma vez que os problemas Sudoku têm apenas uma solução (ou uma forma de preencher os tabuleiros), enquanto que alguns dos problemas que propomos poderão ser preenchidos de mais de uma forma.

Este jogo pode-se também generalizar para tabuleiros de outros tamanhos, embora sempre com tamanhos que são quadrados perfeitos, isto é, de dimensão $2^2 \times 2^2, 3^2 \times 3^2, 4^2 \times 4^2$, etc. Nas Figuras 3 e 4 apresentamos, respectivamente, exemplos de dois tabuleiros de dimensões 4×4 e 16×16.

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

Figure 1: Exemplo de um tabuleiro Sudoku 9×9 parcialmente preenchido.

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

Figure 2: O mesmo exemplo completamente preenchido.

	1	4	
1			
3	2		

Figure 3: Exemplo de um tabuleiro Sudoku 4×4 parcialmente preenchido.

3		1								6					16
			9						4						
				13				3			7			11	
						6						3			
	3						11				6		13		2
					3			4		16					
		14												9	
	13				12										1
				11											
11							10		16						
			8								10				5
						12		11							
		10		9			15				12				
	11									4			6		
			12						1					5	
16		4				8									

Figure 4: Exemplo de um tabuleiro Sudoku 16×16 parcialmente preenchido.

3 O trabalho

Pretende-se estudar a aplicação de técnicas de IA, nomeadamente estratégias de procura em espaço de estados, que permitam resolver o problema de completar tabuleiros de Sudoku parcialmente preenchidos.

Para estudar a aplicação das estratégias de procura, deve-se desenvolver a função `procura`, que recebe como argumentos o nome de um ficheiro em que está definido o problema de Sudoku (o formato deste ficheiro é descrito abaixo) e um símbolo que identifica o tipo de procura que se pretende fazer e que devolve um objecto do tipo tabuleiro (deve ser definido o tipo `tabuleiro` com a interface – construtores, selectores, testes, transformador de saída – descrita em anexo) devidamente preenchido.

Os símbolos que identificam os diferentes tipos de procura são `:profundidade`, `:largura`, `:retrocesso` e `:informada`, correspondendo, respectivamente, à procura em profundidade primeiro, procura em largura primeiro, procura por retrocesso e procura por retrocesso recorrendo à utilização de informação heurística.

3.1 Descrição de um problema de Sudoku

Os problemas de Sudoku que vamos tratar devem ser descritos num ficheiro, cujo conteúdo deve ser o seguinte: uma linha com o inteiro que representa a dimensão do problema de Sudoku a tratar (dim) seguida de dim linhas, cada uma com dim colunas de inteiros entre 0 e dim, sendo que 0 representa uma posição por preencher. Por exemplo, o ficheiro com o seguinte conteúdo

```
9
0 1 4 0 0 0 0 0 0
2 0 0 5 0 4 0 0 0
0 0 0 0 0 0 5 0 0
1 0 5 0 0 0 7 0 2
0 0 2 1 0 5 3 0 6
3 9 0 0 8 2 0 4 0
4 0 0 0 6 0 0 0 7
0 2 0 0 5 0 9 0 0
0 6 0 8 0 0 0 0 1
```

representa o problema apresentado na Figura 1.

3.2 As soluções de problemas Soduku

Dado um problema de Sudoku, a função `procura`, quando receber o nome de um ficheiro que contenha a descrição de um problema de Sudoku e um símbolo que representa o tipo de procura a desenvolver, deve produzir um tabuleiro de Sudoku completamente preenchido, que deve ser um objecto do tipo *tabuleiro*, cuja interface se apresenta em anexo.

Refira-se que pode ser utilizada uma implementação próxima da referida no anexo, embora possa ser utilizada outra, desde que observe a interface do tipo usada no anexo.

Se utilizassemos a implementação do tipo `tabuleiro` descrita no anexo e se o ficheiro "problema.sudoku" contivesse o problema referido acima, poder-se-ia ter a seguinte interacção:

```
> (procura "problema.sudoku" :profundidade)
((5 1 4 9 7 6 2 3 8)
 (2 8 7 5 3 4 6 1 9)
 (6 3 9 2 1 8 5 7 4)
 (1 4 5 6 9 3 7 8 2)
 (8 7 2 1 4 5 3 9 6)
 (3 9 6 7 8 2 1 4 5)
```

```
(4 5 1 3 6 9 8 2 7)
(7 2 8 4 5 1 9 6 3)
(9 6 3 8 2 7 4 5 1))
```

4 Avaliação

A avaliação é feita da seguinte forma:

- 75% para avaliar a adequação da implementação das funções pedidas, incluindo questões de eficiência computacional;
- 25% para o relatório. O relatório (de no máximo 7 páginas) deve discutir o trabalho desenvolvido, incluindo
  - apresentação das várias estratégias de procura utilizadas;
  - comparação de resultados obtidos;
  - avaliação critica das estratégias aplicadas, incluindo as conclusões a que foi possível chegar;
  - e outros aspectos que se entendam relevantes.

5 Entrega dos trabalhos

A entrega do trabalho deve ser feita via sistema Fénix até às 23h59 do dia 6 de Dezembro. O trabalho a entregar deve ser constituído por um ficheiro de texto de nome "<grupo>.lisp" (em que <grupo> é substituído pelo número do grupo) contendo o código Lisp desenvolvido e por um ficheiro pdf de nome "<grupo>.pdf" contendo o relatório. Estes dois ficheiros devem ser agrupados num ficheiro em formato TAR.

O código Lisp, que deve obedecer à norma ANSI e compilar sem qualquer aviso (warning), deve ser executado sem qualquer erro. Para garantir que obedecem à norma ANSI, se utilizarem o CLISP para desenvolver o código, devem ter o cuidado de utilizá-lo com a flag -ANSI. Vai ser utilizado para os testes o compilador de SBCL.

Apêndice

A O tipo abstracto de informação *tabuleiro*

O objectivo do TAI *tabuleiro* é representar um problema de Sudoku. Estes problemas são grelhas quadradas $2^n \times 2^n$ para $n \in \{1, 2, 3, ...\}$, em que cada posição da grelha guarda um número entre 0 e 2^n.

Nos exemplos que apresentamos, vamos implementar os objectos do tipo *tabuleiro* como listas de listas, embora fique explicitamente expresso que se pode utilizar outra implementação.

O tipo *tabuleiro* é definido através da seguinte interface:

- construtor `faz-tabuleiro`: *inteiro* $\times$ *inteiro* $\rightarrow$ *tabuleiro*
  Recebe como argumentos um inteiro que representa o número de linhas (e também de colunas) do tabuleiro que se vai construir e um outro inteiro e retorna um tabuleiro que tem esse outro inteiro em todas as posições.

```
> (setf tab1 (faz-tabuleiro 9 0))
((0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0))
```

- construtor `tabuleiro-poe-numero`: *tabuleiro* $\times$ *inteiro* $\times$ *inteiro* $\times$ *inteiro* $\rightarrow$ *tabuleiro*
  Esta função retorna como resultado um novo tabuleiro, copiado a partir daquele que recebeu como 1º argumento mas com o inteiro recebido como 2º argumento na posição cuja linha corresponde ao 3º argumento e cuja coluna corresponde ao 4º argumento. Por exemplo,

```
> (setf tab2 (tabuleiro-poe-numero tab1 4 0 2))
((0 0 4 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 0)
```

```
(0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0)
(0 0 0 0 0 0 0 0 0))
```

- selector `tabuleiro-numero`: *tabuleiro* × *inteiro* × *inteiro* → *inteiro*
  Retorna o inteiro que está no tabuleiro recebido como 1º argumento na posição cuja linha corresponde ao 2º argumento e coluna corresponde ao 3º argumento. Por exemplo,

```
> (tabuleiro-numero tab2 0 2)
4
> (tabuleiro-numero tab2 0 0)
0
```

- selector `tabuleiro-dimensao`: *tabuleiro* →*inteiro*
  Retorna a dimensão do tabuleiro.

```
> (tabuleiro-dimensao tab2)
9
```

- reconhecedor `tabuleiro-p`: *universal* → *booleano*
  Esta função reconhece objectos do tipo *tabuleiro*. O reconhecedor retorna `T` se o objecto recebido for um tabuleiro e `NIL` caso contrário. Por exemplo,

```
> (tabuleiro-p (faz-tabuleiro 9 0))
T
> (tabuleiro-p '((1 2 3)))
NIL
```

- transformador de entrada `le-tabuleiro`: *stream* → *tabuleiro*
  Recebe como argumento uma stream, lê da stream um inteiro dim correspondente à dimensão do tabuleiro e para cada uma das dim linhas, lê da stream dim inteiros correspondentes ao conteúdo das colunas. Por fim, retorna o tabuleiro lido. Por exemplo, se `stream` for uma stream que contém a sequência de números

```
9
0 1 4 0 0 0 0 0 0
```

```
2 0 0 5 0 4 0 0 0
0 0 0 0 0 0 5 0 0
1 0 5 0 0 0 7 0 2
0 0 2 1 0 5 3 0 6
3 9 0 0 8 2 0 4 0
4 0 0 0 6 0 0 0 7
0 2 0 0 5 0 9 0 0
0 6 0 8 0 0 0 0 1
```

obtém-se o seguinte comportamento:

```
> (setf tab3 (le-tabuleiro stream))
((0 1 4 0 0 0 0 0 0)
 (2 0 0 5 0 4 0 0 0)
 (0 0 0 0 0 0 5 0 0)
 (1 0 5 0 0 0 7 0 2)
 (0 0 2 1 0 5 3 0 6)
 (3 9 0 0 8 2 0 4 0)
 (4 0 0 0 6 0 0 0 7)
 (0 2 0 0 5 0 9 0 0)
 (0 6 0 8 0 0 0 0 1))
```

- transformador de saída `escreve-tabuleiro`: *tabuleiro* $\rightarrow$ *NIL*
  Recebe como argumento um tabuleiro. Escrevem-se as linhas do tabuleiro sequencialmente (uma em cada linha de texto), sendo os inteiros escritos usando-se a directiva de formatação "~S" do format e retorna NIL. Por exemplo,

```
> (escreve-tabuleiro tab3)
9
0 1 4 0 0 0 0 0 0
2 0 0 5 0 4 0 0 0
0 0 0 0 0 0 5 0 0
1 0 5 0 0 0 7 0 2
0 0 2 1 0 5 3 0 6
3 9 0 0 8 2 0 4 0
4 0 0 0 6 0 0 0 7
0 2 0 0 5 0 9 0 0
0 6 0 8 0 0 0 0 1
NIL
```

O NIL é escrito pelo ciclo Read-eval-print, depois de ter escrito o tabuleiro. Nota: Sugere-se que se observe a descrição do procedimento `format` no Hyperspec.