INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
UNIVERSIDADE DE SÃO PAULO

SCC0230 – Inteligência Artificial

Métodos de Busca
Modelagem Computacional do Problema das N-Rainhas

Elias Italiano Rodrigues – 7987251
Rodolfo Megiato de Lima – 7987286
Rodrigo Rusa – 7986970

São Carlos, SP
6 de outubro de 2014

Sumário

1. Introdução

Muitos dos problemas em computação podem ser resolvidos quando modelados como um espaço de estados, tornando-se a partir daí problemas de certa forma genéricos que podem ser solucionados com métodos de busca tradicionalmente conhecidos. Os estados do problema podem compor um grafo que possui nós e transições representando algumas características específicas do problema em si. É apenas nessa modelagem que se utiliza o conhecimento sobre o domínio do que se deseja resolver, pois após este ser caracterizado como o espaço de estados, basta apenas aplicar um método de busca e procurar por um estado solução do problema.

Na área de Inteligência Artificial, essa técnica de modelagem de problemas como espaço de estados e aplicação de algoritmos de busca para solução dos problemas é muito comum e na maioria das vezes é bem viável. Sendo assim, o objetivo desse primeiro trabalho da disciplina é a aplicação dessa técnica em um problema já conhecido: N-Rainhas.

2. O Problema das N-Rainhas e a Modelagem dos Estados

O problema das N-Rainhas consiste em posicionar as rainhas (peças de xadrez que podem se movimentar e capturar peças adversárias na vertical, horizontal e diagonal quantas casas forem possíveis sem transpor peças intermediárias) em um tabuleiro análogo ao de xadrez mas com N x N casas, sem que estas rainhas se ataquem.

Esse problema é tradicionalmente conhecido na área de Inteligência Artificial e pode ser modelado como um espaço de estados e aplicadas técnicas de buscas para se chegar a uma solução. Para resolver esse problema com técnicas usuais, como uma busca cega por exemplo que é de complexidade exponencial, uma entrada N muito grande pode não se tornar viável. Para isso, utiliza-se de técnicas com heurística como a busca informada e modelagem como problema de satisfação de restrições para que o problema possa ser escalado em maiores números e também possa ser resolvido em um tempo hábil.

Sendo assim, o desenvolvimento desse trabalho segue essa linha em que serão utilizadas essas três técnicas de solução do problema: como busca cega, a busca em profundidade (DFS); como busca informada, o Hill-Climbing; e no problema de satisfação de restrições, as heurísticas de mínimos valores remanescentes (MRV) e verificação adiante (Forward-Checking).

Para modelar o problema como espaços de estados, foi considerado cada estado como uma configuração de todas as N-Rainhas posicionadas no tabuleiro de forma que cada uma se encontra em uma coluna. Dessa forma, apenas as linhas são contabilizadas e assim já existe uma restrição natural de que cada rainha não pode encontrar com uma outra na vertical, restando apenas verificar a consistência do atual estado como solução no que diz respeito as direções horizontal e diagonal de cada uma das rainhas no tabuleiro.

3. Justificativas Para a Adoção dos Algoritmos e Estruturas de Dados

Os algoritmos utilizados, como dito no item anterior desse relatório foram: a busca em profundidade (DFS) como o método de busca cega; o Hill-Climbing como o método de busca informada; e na modelagem como problema de satisfação de restrições, as heurísticas de mínimos valores remanescentes (MRV) e verificação adiante (Forward-Checking).

Para o método de busca cega, a DFS foi escolhida devido a sua característica de guardar menos estados em memória em comparação com a BFS e como não há necessidade de obter a melhor solução, mas sim resolver o problema, obtamos pela DFS como busca não informada. Além disso, o algoritmo DFS serviu como base para implementar o algoritmo da busca com satisfação de restrições.

Como busca informada, a escolha do Hill-Climbing foi devido ao uso de pouca memória por ser uma busca local, a possibilidade de encontrar uma solução em espaços de grande número de estados como nesse cenário e pela aplicação da heurística de mínimos conflitos (contagem dos conflitos para cada par de rainhas), já que o Hill-Climbing visa mínimos ou máximos globais. Assim, espera-se que esse algoritmo escontre o mínimo global, onde o número de conflitos é zero, mas pode não apresentar solução caso alcance mínimos locais.

E finalmente, o problema sendo modelado como problema de satisfação de restrições é perfeito para abordar a ideia das N-Rainhas, já que de forma natural do problema, ele nada mais é do que satisfazer as restrições das rainhas não poderem se encontrar nas três direções já citadas (vertical, horizontal e diagonal) no tabuleiro. Então, foram utilizadas as heurísticas de verificação adiante e mínimos valores remanescentes para lidar com essas restrições, já que o primeiro verifica se a mudança de estado não deixou outra rainha sem possíveis movimentos evitando caminhos não promissores, e o segundo em que se escolhe a rainha que tem o menor número de casas legais possíveis para ser a próxima a ser posicionada no tabuleiro.

Para a modelagem dos dados do problema, a estrutura de dados mais simples que poderia ser utilizada para representar as rainhas e seus estados é um vetor de inteiros, onde o índice representa a coluna em que a rainha está (seu identificador, já que cada uma a princípio já está alocada a uma coluna) e o seu valor correspondente no vetor representa a linha dessa rainha. Também foi utilizado uma matriz de inteiros para armazenar os valores disponíveis dos domínios de cada rainha quando se realiza busca com satisfações de restrições.

4. Especificações de Sistema e Compilação

A implementação da solução das N-Rainhas foi desenvolvida em linguagem C, utilizando o compilador GCC e o sistema operacional GNU/Linux.

5. Compilação e Execução do Programa

Estando do diretório onde encontra-se o código-fonte, execute o seguinte comando para compilar o programa:

```
gcc -o n_queens n_queens.c -O2
```

Para executar o programa, execute o seguinte comando substituindo os argumentos como explicado adiante:

```
./n_queens <method> <n> [<draw>]
```

`<method>`: o tipo de algoritmo deseja para resolver o problema.
Pode assumir os seguintes valores:

- `0` : Hill-Climbing
- `1` : DFS
- `2` : Constraints (Satisfação de Restrições)

`<n>`: a quantidade de rainhas deseja

`[<draw>]`: argumento opcional, informe `0` para não desenhar o tabuleiro na tela.

A saída do programa, além de dizer se uma solução foi encontrada ou não, mostra a solução ou então o estado final que o programa chegou a partir de um desenho que representa o tabuleiro e as N-Rainhas posicionadas nele.

6. Experimento Realizados

Foram realizados vários experimentos para que se averiguasse a corretude da solução. Começando pelos casos triviais, quando se possui apenas uma rainha que a solução é imediata. Nos casos de 2 ou 3 rainhas, não há solução. Também foram realizados experimentos com valores maiores para verificar sua corretude.

Segue abaixo um *exemplo* saída de uma execução utilizando o método de busca DFS com 8 rainhas no tabuleiro:

```
     Estado inicial gerado:

  Q = [ 1  1  7  5  0  7  1  6 ]

---------------------------------
|   |   |   |   | 0 |   |   |   |
---------------------------------
| 0 | 0 |   |   |   |   | 0 |   |
---------------------------------
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   | 0 |   |   |   |   |
---------------------------------
|   |   |   |   |   |   |   | 0 |
---------------------------------
|   |   | 0 |   |   | 0 |   |   |
---------------------------------

            Rainhas: 8
           Metodo: DFS
            Resolvido!
       Movimentos: 1485548

           Estado final:

  Q = [ 0  4  7  5  2  6  1  3 ]

---------------------------------
| 0 |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   | 0 |   |
---------------------------------
|   |   |   |   | 0 |   |   |   |
---------------------------------
|   |   |   |   |   |   |   | 0 |
---------------------------------
|   | 0 |   |   |   |   |   |   |
---------------------------------
|   |   |   | 0 |   |   |   |   |
---------------------------------
|   |   |   |   |   | 0 |   |   |
---------------------------------
|   |   | 0 |   |   |   |   |   |
---------------------------------
```

7. Análise dos Resultados e Complexidades do Algoritmos

Quantidade de movimentos executados pelos algoritmos em função de N

Algoritmo	Complexidade	N = 8	N = 10	N = 50	N = 100	N = 1000
Hill-Climbing	O(N^3)	*	*	*	*	*
DFS	O(N^N)	1 485 548	286 609 045	**	**	**
Constraints	O(N^N)	75	35	2 068	185	1 007

* raramente encontra uma solução
** não computável

Para os algoritmos Hill-Climbing e DFS, o estado inicial é gerado aleatoriamente. Isso tem implicações na quantidade de movimentos executados durante a resolução, porém não foi considerado nessa análise uma vez que já não é possível se resolver o problema para um valor grande de N.

O algoritmo Hill-Climbing por não realizar backtracking após a tomada de uma decisão (guloso), efetuará N movimentos no pior caso até encontrar uma solução ou não. A cada movimento feito é conferido se o estado atual é solução, o que tem complexidade O(N^2). Logo, o Hill-Climbing possui complexidade O(N^3). Importante notar que raramente ele encontra uma solução.

No pior caso do algoritmo DFS ele percorrerá todo o espaço de busca, ou seja, todas as combinaçõs possíveis das rainhas no tabuleiro, o que tem complexidade O(N^N) e o torna inviável para resolver o problema para um valor grande de N.

Assim como para DFS, o pior caso da Busca com Restrições (Constraints) percorrerá também todo o espaço de busca tendo complexidade O(N^N). Porém, essa complexidade é mais teórica do que prática, pois as heurísticas aplicadas (Forward-Checking e MRV) “podam” a árvore de estados e acaba por diminuir drasticamente o espaço de busca.

8. Considerações Finais

Podemos verificar com a implementação do problema das N-Rainhas e aplicação dos testes, que o problema pode ser escalado de forma muito satisfatória se aplicadas algumas heurísticas em vez de simplesmente se executar buscas cegas, que exploram todos os estados possíveis podendo inviabilizar execuções para valores de N não tão grandes.

Analisando separadamente cada um dos algoritmos aplicados, podemos perceber que a busca não-informada (DFS) sempre encontra a solução, mas não em um tempo hábil dependendo de N. Já a busca informada (Hill-Climbing) termina em tempo satisfatório, mas na maioria dos casos não chega a uma solução, pois converge para mínimos locais. A modelagem da aplicação como problema de satisfação de restrições é perfeita para esse caso, sempre encontrando a solução rapidamente e para valores de N elevados, devido a natureza do problema que permite o uso das heurísticas que podam o espaço de busca eficientemente.

O problema das N-Rainhas não era grande novidade, pois trata-se de um problema tradicional. Porém, não havíamos pensado em soluções e implementações para esse problema. A princípio pensou-se em implementá-lo em Prolog, mas diante da necessidade de fazê-lo em tempo hábil, optou-se por programar em linguagem C que já é de maior domínio.

9. Bibliografia

Slides da Professora Solange Oliveira Rezende.
Disponível em: <http://tidia-ae.usp.br/>. Acesso: 05 de outubro de 2014.