MAC–IME–USP

THIAGO MOURA WITT
NUSP:4878021
EMAIL: WITT@IME.USP.BR

MAC 5739 – Introdução à Inteligência Artificial

O problema das *N*-Rainhas

Entrega: 14/10/2003

O Problema das *N*-Rainhas

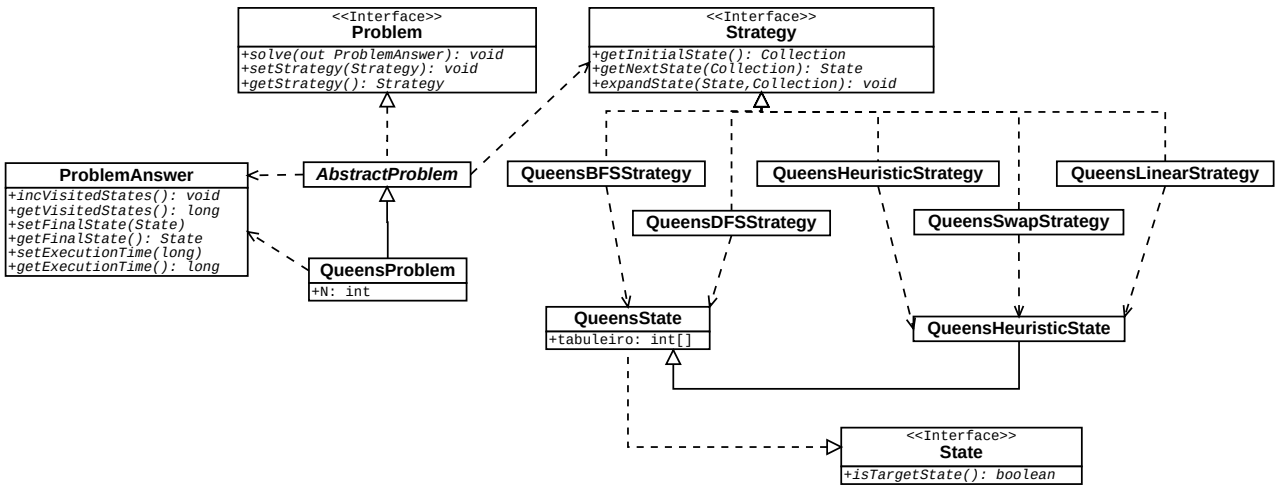
O problema das 8 Rainhas consiste em dispor 8 rainhas em um tabuleiro de xadrez (8x8) de modo que nenhuma esteja ameaçada por outra, ou seja, não há mais que uma rainha em cada linha, coluna ou diagonal do tabuleiro. O problemas das *N* Rainhas é uma generalização do problema acima, devemos dispor *N* rainhas sobre um tabuleiro *N*x*N* de modo que nenhuma esteja ameaçada. O problema das *N* Rainhas possui solução para todo $N \geq 4$.

Um exemplo de aplicação prática deste problema está na computação ótica, onde é necessário alinhar unidades óticas de processamento em uma matriz bi-dimensional, de modo a maximizar largura de banda de comunicação. Ou seja, devemos dispor as unidades óticas numa matriz de forma que as unidades estejam diretamente acessíveis das oito direções possíveis (duas na vertical, duas na horizontal, e quatro nas diagonais). A solução para este problema é a solução para o problema das rainhas.

Este artigo faz uma comparação entre algumas diferentes técnicas de busca cega e busca informada para encontrar uma solução para o problema das *N*-Rainhas.

Arquitetura do Sistema

O sistema foi desenhado para se aproveitar das técnicas de orientação a objetos e proporcionar reutilização de código entre as diferentes estratégias de busca. Foi criada uma classe chamada *AbstractProblem*, que possui uma referência para um objeto do tipo *Strategy*, responsável por prover as operações específicas de cada estratégia utilizada. A classe *AbstractProblem* possui também uma implementação padrão de um algoritmo simples de busca que utiliza as operações do objeto *Strategy* para realizar a busca. Deste modo, é possível escolher a estratégia utilizada pelo algoritmo de busca em tempo de execução. Segue um diagrama UML do sistema:



Armazenamento de Estado e Verificação de Estado-Meta

Considere uma solução qualquer para o problema das *N*-Rainhas. Sabemos que não podem haver 2 rainhas na mesma linha ou na mesma coluna, pois elas estariam se atacando, portanto temos uma relação um pra um entre rainhas e colunas e entre rainhas e linhas. Então, como sabemos que haverá apenas uma rainha em cada coluna do tabuleiro torna-se desnecessário o uso de uma matriz *N*x*N* para armazenar o estado do tabuleiro, bastando um vetor com *N* posições para isso, de modo que a posição *i* do vetor corresponde à coluna *i* do tabuleiro e o valor armazenado nesta posição indica a linha em que a rainha está posicionada. Esta estrutura de armazenamento foi usada em todos os algoritmos implementados para este problema.

Nos algoritmos de busca cega, o tabuleiro começa vazio e, a cada passo é adicionada, se possível, uma rainha ao mesmo, na coluna vazia mais à esquerda garantindo, deste modo, que a última coluna do tabuleiro possuirá uma Rainha se e só se uma solução para o problema foi encontrada. Portanto a verificação de estado-meta é feita simplesmente verificando-se se a última coluna do tabuleiro possui uma Rainha.

Já nos algoritmos de busca informada, o tabuleiro sempre começa com rainhas em todas as colunas, e a partir daí o algoritmo começa a mover as rainhas de modo a chegar a uma solução. Neste caso a verificação de estado-meta é feita percorrendo o vetor em busca de duas rainhas na mesma linha ou diagonal.

Estratégias de busca

Busca Cega

Os algoritmos de busca cega partem de um tabuleiro vazio e vão construindo a solução da coluna 1 para a coluna *N*, uma coluna por vez. O estado inicial é o tabuleiro vazio e a expansão de um estado consiste em encontrar uma posição válida na coluna vazia mais à esquerda possível.

- Busca em largura

  A busca em largura utiliza uma fila para armazenar os estados possíveis. A cada passo da busca o primeiro estado

da fila é removido e analisado e, caso não seja um estado-meta, é expandido e os estados obtidos a partir desta expansão são inseridos no final da fila.

Tempo de execução desta estratégia:

N	4	8	12	16
Estados Visitados	16	1966	841990	Out of Memory
Tempo de Execução (ms)	11	93	40965	-

Este método é o mais ineficiente para este problema, o número de estado possíveis cresce exponencialmente com o tamanho do tabuleiro e a busca em largura precisa armazenar uma parte muito grande destes estados na memória simultaneamente, o que inviabiliza o uso desta estratégia para tabuleiros com N grande.

- Busca em profundidade

  A busca em largura utiliza uma pilha para armazenar os estados visitados. A cada passo da busca o estado no topo da pilha é analisado e caso não seja um estado-meta, o próximo estado não visitado a partir de uma expansão deste estado é inserida na pilha e, caso não hajam mais expansões possíveis, ele é removido da pilha e o estado anterior é analisado (backtrack). Neste algoritmo, o número máximo de estados armazenados em memória durante a execução é N, portanto o problema de falta de memória não se repetirá aqui a menos que o N seja extraordinariamente grande.

  Tempo de execução desta estratégia:

N	4	8	12	16	20	24	28
Estados Visitados	13	219	511	20089	399251	823193	6012569
Tempo de Execução (ms)	11	40	48	563	16806	57318	644569 (~10m30s)

  Este algoritmo já apresenta uma grande melhora em relação à busca em largura, apesar de manter o caráter exponencial e ainda não conseguir encontrar em tempo razoável a solução para $N > 24$.

Busca Informada

Os algoritmos de busca informada partem de um estado inicial onde todas as rainhas já estão posicionadas no tabuleiro, uma por coluna. A partir daí as rainhas são trocadas de lugar de acordo com uma heurística até que se alcance o estado meta. A busca heurística não apresenta muitas vantagens sobre a busca cega para $N \leq 12$, mas para N maiores ela é muito mais eficiente e consegue calcular soluções para problemas com N muito maiores.

Dada a natureza não determinística destas soluções, as medidas de tempo de execução e estados visitados apresentadas são a média aritmética de dez execuções consecutivas do algoritmo para o mesmo N.

- Heurística de Minimização de Conflitos

Inicialmente, todas as rainhas são dispostas no tabuleiro, na diagonal principal. Cada passo da busca procura pela rainha mais ameaçada (em uma posição com maior número de conflitos) e a move para uma das linhas com menor conflito (no caso de empate escolhe aleatoriamente).

Esta estratégia apresentou problemas em sua primeira versão, onde sempre a rainha mais ameaçada era escolhida para a troca de posição. Em alguns casos o algoritmo entrava em loop infinito por cair em um circuito no espaço de estados. Para resolver este problema, foi utilizada uma estratégia semelhante à *Hill Climbing*, de modo que a rainha escolhida nem sempre fosse a com maior número de ataques, mas sim uma outra rainha escolhida aleatoriamente.

Tempo de execução desta estratégia:

N	4	8	12	16	20	24	28	50	100	1000
Estados Visitados	167	206	366	604	706	897	909	853	1473	6931
Tempo de Execução (ms)	54	66	74	91	95	100	121	162	578	232277 (~4m)

Vemos como o uso de uma boa heurística pode reduzir drasticamente o número de estados visitados no decorrer da busca e permitir o cálculo da solução para instâncias muito maiores do problema do que a busca cega permitiria.

- Heurística de Troca de Posições

  Esta heurística requer um tabuleiro inicial em que as rainhas estão dispostas de forma que não existam duas rainhas distintas na mesma linha ou coluna. A cada passo da busca é gerada aleatoriamente uma permutação do vetor que representa o tabuleiro, então todas as posições são testadas duas a duas e, caso a troca de posições entre as duas rainhas diminua o número de conflitos no tabuleiro, a troca é feita.

  Tempo de execução desta estratégia:

N	4	8	12	16	20	24	28	50	100	1000	10000
Estados Visitados	2	6	12	13	12	17	19	13	7	3	3
Tempo de Execução (ms)	29	31	42	47	52	61	66	77	81	370	29014

  Esta heurística foi baseada em [SG90]. Note que, com o crescimento do N, o número de permutações necessárias antes de se encontrar uma solução vai diminuindo, ou seja, quanto maior o N, maior a probabilidade do algoritmo encontrar a solução nas primeiras permutações geradas. Existem versões aperfeiçoadas deste algoritmo descritas em [SG91b, SG91a].

Estratégia Linear

Esta estratégia foi implementada para efeito de comparação com os métodos de busca. Ela utiliza um algoritmo linear apresentado em [Ber91] que calcula a posição de cada rainha no tabuleiro em tempo constante por rainha. O número de estados visitados é sempre 1 porque o algoritmo já devolve a solução como estado inicial. Esta implementação não foi otimizada, por esse motivo vemos que para $N = 4$ os algoritmos de busca cega ainda são mais rápidos.

Tempo de execução desta estratégia:

N	4	8	12	16	20	24	28	50	100	1000	10000
Estados Visitados	1	1	1	1	1	1	1	1	1	1	1
Tempo de Execução (ms)	26	26	26	26	27	27	28	29	36	70	2379

Vemos que este algoritmo é realmente eficiente. Seu único ponto fraco consiste na classe restrita de soluções obtidas, apesar de sempre determinar uma solução para o problema, isso é tudo o que ele pode fazer, ele determina exatamente uma possível configuração do tabuleiro para cada *N* dado.

Tabelas de comparação

Aqui reunimos os dados de execução de todos os algoritmos citados durante o documento para facilitar a comparação dos resultados.

Estados Visitados

N	4	8	12	16	20	24	28	50	100	1000	10000
Largura	16	1966	841990	-	-	-	-	-	-	-	-
Profundidade	13	219	511	20089	399251	823193	6012569	-	-	-	-
Min. de Conflitos	167	206	366	604	706	897	909	853	1473	6931	-
Troca de Posições	2	6	12	13	12	17	19	13	7	3	3
Linear	1	1	1	1	1	1	1	1	1	1	1

Tempo de execução (ms)

N	4	8	12	16	20	24	28	50	100	1000	10000
Largura	11	93	40965	-	-	-	-	-	-	-	-
Profundidade	11	40	48	563	16806	57318	644569	-	-	-	-
Min. de Conflitos	54	66	74	91	95	100	121	162	578	232277	-
Troca de Posições	29	31	42	47	52	61	66	77	81	370	29014
Linear	26	26	26	26	27	27	28	29	36	70	2379

Referências Bibliográficas

[Ber91] B. Bernhardsson. Explicit solution to the *n*-queens problems for all *n*. *ACM SIGART Bulletin*, 2:7, 1991.

[SG90] Rok Sosič and Jun Gu. A polynomial time algorithm for the *N*-queens problem. *SIGART Bulletin*, 1:7–11, 1990.

[SG91a] Rok Sosič and Jun Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 2:22–24, 1991.

[SG91b] Rok Sosič and Jun Gu. Fast search algorithms for the *n*-queens problem. *IEEE TRansactions on Systems, Man and Cybernetics*, 21:1572–1576, 1991.