Cobertura de um Tabuleiro de Xadrez por Oito Peças

Renato Fonseca F. Werneck

10 de Janeiro de 2000

1 Introdução

Este documento apresenta um algoritmo elaborado com o objetivo de responder à seguinte pergunta: é possível cobrir todas as 64 casas de um tabuleiro de xadrez com as oito peças mais poderosas (rei, rainha, torres, bispos e cavalos)? Note que cada uma das 64 casas (inclusive as oito ocupadas) devem estar sob ataque de pelo menos uma das peças. A partir do algoritmo elaborado, provou-se que a resposta a essa pergunta é "não". Não há um arranjo capaz de cobrir todas as casas; o melhor que se pode fazer é cobrir 63 delas.

Notação Ao longo desse documento, principalmente nas figuras, serão feitas referências às peças utilizando a seguinte abreviação, baseada em seus nomes em inglês:

- `B`: bispo preto (***B****lack Bishop*);
- `K`: rei (***K****ing*);
- `N`: cavalo (*K****N****ight*);
- `Q`: rainha (***Q****ueen*);
- `R`: torre (***R****ook*);
- `W`: bispo branco (***W****hite Bishop*).

2 Análise do Problema

Não há um algoritmo polinomial conhecido para o problema proposto.

A estratégia de resolução mais simplista é a busca exaustiva: basta testar todas as possíveis configurações das peças no tabuleiro e verificar se alguma delas cobre as 64 casas. O número de configurações pode ser calculado de forma simples: o bispo preto pode ocupar 32 casas; o branco, outras 32; às demais peças, consideradas em uma ordem arbitrária, restam 62, 61, 60, 59, 58 e 57 casas, respectivamente. Portanto, o número total de configurações é:

$$C_T = 32 \cdot 32 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57 = 45.323.933.368.320 \approx 4,5 \cdot 10^{13} \quad (1)$$

A busca exaustiva, portanto, requereria a análise de aproximadamente 45 trilhões de configurações. Conforme se verá nas seções que se seguem, muitas das configurações têm propriedades comuns que permitem sua eliminação sem a necessidade de uma análise individualizada.

2.1 Eliminação dos Pares

No conjunto das peças consideradas, os dois cavalos são perfeitamente equivalentes, assim como as duas torres. O valor calculado na equação 1 considera implicitamente que as peças de cada par são diferentes, o que não é verdade. Portanto, o valor calculado está superdimensionado. Ele se reduz à metade devido à equivalência dos cavalos e é novamente dividido por dois quando se considera a equivalência entre as torres.

Repare que, apesar de haver dois bispos, eles não são equivalentes, uma vez que um só pode ocupar as casas pretas e o outro, as brancas. Assim sendo, a estratégia utilizada para tratar cavalos e torres não se aplica a eles.

2.2 Simetria

Outra maneira de reduzir o número de estados a processar é analisar situações de simetria. Em particular, uma única peça, considerada isoladamente no tabuleiro, pode ocupar apenas 10 casas completamente distintas, mostradas na figura 1. Cada uma das demais casas é equivalente a uma das assinaladas, a menos de rotações do tabuleiro e/ou reflexões em relação à vertical ou horizontal.

```
X . . . . . . .
X X . . . . . .
X X X . . . . .
X X X X . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Figura 1: Dez possíveis posições para a primeira peça

Portanto, se a busca por soluções for feita considerando-se as peças sempre em uma mesma ordem, basta que sejam analisadas as configurações em que a primeira delas está em uma das 10 posições assinaladas (nesse caso, a primeira peça não poderá ser um bispo, já que ele está restrito a casas de uma só cor). No caso específico do algoritmo implementado, foi utilizada a seguinte ordem.

1. rainha (`Q`)
2. torre (`R`)
3. torre (`R`)
4. bispo branco (`W`)
5. bispo preto (`B`)
6. cavalo (`N`)
7. cavalo (`N`)
8. rei (`K`)

De acordo com essa organização, portanto, a rainha só precisa ser testada em 10 diferentes posições, e não em 64. Isso representa uma redução de 85,5% o número de configurações analisadas.

Repare que as peças foram organizadas de acordo com seu potencial de cobertura, o número de casas que são capazes atacar a partir de sua posição. Enquanto a primeira delas (a rainha) ataca no mínimo 21 casas (quando está em um dos quatro "cantos" do tabuleiro), nenhuma das três últimas (os cavalos e o rei) pode atacar sozinha mais de oito posições. Conforme se verá na seção 3, essa particular ordenação das peças é importante para a implementação de estratégias de *prunning*, capazes de reduzir sensivelmente o número de estados analisados.

Outras Peças A simetria pode ser utilizada para limitar o número de posições não só no caso da rainha, a primeira peça, mas também das três peças seguintes (duas torres e bispo branco). Entretanto, como o número de linhas e colunas é par e só há uma rainha, não é possível haver simetria vertical ou horizontal. O único eixo de simetria possível é a diagonal "principal" do tabuleiro (do canto superior esquerdo ao inferior direito).

A figura 1 mostra que tabuleiro será simétrico em relação a essa diagonal quando a rainha for colocada em quatro das dez posições assinaladas. Nesses casos, basta testar as configurações em que a segunda peça (a primeira torre) é colocada sobre a diagonal ou acima dela. Isso reduz o número de posições da primeira torre de 63 (já desconsiderada a posição ocupada pela rainha) para 35.

O mesmo raciocínio pode ser aplicado para a segunda torre, para o bispo branco e para o bispo preto. Repare que é possível haver simetria em um tabuleiro contendo apenas a rainha e as duas torres mesmo que nem todas as peças estejam sobre a diagonal principal. Basta que as torres estejam em posições simétricas em relação à diagonal, ou seja, que o número da coluna em que está uma delas seja igual à linha em que está a outra (e vice-versa).

No momento da inserção de qualquer das demais peças (cavalos e rei), certamente não haverá simetria, já que há um único bispo preto e ele jamais estará na diagonal principal, composta por casas brancas.

3 *Prunning*

As estratégias descritas anteriormente têm a finalidade de reduzir o espaço de busca. De fato, considerando-se as técnicas em conjunto, o número de tabuleiros a analisar reduz-se de aproximadamente $4,5 \cdot 10^{13}$ para menos de $1,4 \cdot 10^{12}$. Apesar de 30 vezes menor que o original, o número de estados ainda é relativamente grande. Para que o problema fosse resolvido em 24 horas, por exemplo, seria necessário processar mais de 16 milhões de estados por segundo.

No entanto, é possível reduzir ainda mais o número de estados analisados aplicando-se técnicas de *prunning*. A idéia básica é a seguinte: se, a partir da análise de um tabuleiro em que estão colocadas k peças ($k < 8$), chegarmos à conclusão de que as peças restantes jamais serão capazes de atacar todas as casas não-cobertas, os estados formados a partir desse tabuleiro devem ser simplesmente descartados.

Nas subseções que se seguem, serão analisados os métodos utilizados para fazer o *prunning* da árvore de resolução do problema.

3.1 Contagem

Considere a última peça colocada, o rei. Essa peça é capaz de cobrir, na melhor das hipóteses, oito casas de um tabuleiro. A contribuição do rei para a cobertura total do tabuleiro, no entanto, pode ser bem menor devido a uma série de fatores (que podem coexistir):

- quando colocado nas bordas do tabuleiro, o rei cobre apenas cinco casas (ou três, se estiver em um dos quatro cantos);
- algumas das casas vizinhas ao rei no tabuleiro podem já estar cobertas por outras peças;
- o rei pode bloquear a ação de um bispo, torre ou rainha; na pior hipóteses, o rei pode até diminuir o número de casas cobertas do tabuleiro.

De qualquer forma, é possível garantir que, se um tabuleiro tiver mais de oito casas não-cobertas, um rei jamais será capaz de cobri-lo por completo. Portanto, se, após a inserção da sétima peça, o número de posições sob ataque for inferior a 56, não é necessário analisar cada uma das posições possíveis para o rei.

O mesmo raciocínio vale para as demais peças. A sétima peça colocada é um cavalo, que, assim como o rei, pode cobrir apenas oito posições. Se um tabuleiro com as seis primeiras peças tiver menos de 48 casas cobertas, pode-se fazer o *prunning* dos ramos que o sucedem (as duas últimas peças — um cavalo e um rei — podem cobrir no máximo 16 posições). Pela mesma razão, para a sexta peça, o outro cavalo, o número mínimo de casas cobertas para que não se realize o *prunning* é 40.

No algoritmo implementado, o *prunning* por contagem simples é aplicado apenas para as três últimas peças. Para as demais, o mesmo raciocínio poderia ser aplicado, mas o número de situações em que de fato ocorreria o *prunning* seria muito reduzido. A subseção seguinte apresenta um método mais eleborado utilizado no quinto nível da árvore, correspondente ao segundo bispo (o preto).

3.2 Diagonais

O bispo preto pode ser inserido em no máximo 32 posições. De acordo com o que foi discutido na subseção anterior, para que ele seja útil é necessário que, após sua inserção, pelo menos 40 casas estejam cobertas. Um único bispo é capaz de cobrir até 13 casas, mas em muitas das posições o número efetivamente coberto é muito menor, especialmente considerando o fato de que ele pode bloquear a rainha ou as torres anteriormente inseridas.

A estratégia utilizada para *prunning*, no caso do bispo preto, foi o cálculo, para cada uma das possíveis posições, de um limite superior para o número de casas cobertas após a inserção do bispo. Trata-se simplesmente da contagem do número de casas ainda não cobertas que são atacadas pelo bispo. Esse é um limite superior porque não são descontadas as perdas relativas às "sombras" geradas pelo bispo, ou seja, as casas que deixarão de ser cobertas devido à inserção da nova peça.

Após as contagens, só são consideradas as posições em que o limite superior é pelo menos 40. Conforme visto anteriormente, esse é o número mínimo de casas que devem estar cobertas antes da inserção do primeiro cavalo.

3.3 Rei

A última peça inserida é o rei. Dadas as características de seu movimento, todas as casas cobertas por ele devem estar contidas em um quadrado cujo lado é composto por três casas. Portanto, se forem encontradas duas casas descobertas distantes entre si em mais de duas unidades (considerando-se movimentos na diagonal, vertical ou horizontal), pode-se garantir que não será possível cobrir todo o tabuleiro, independentemente da posição de inserção do rei. Em vista disso, o *prunning* do ramo pode ser efetuado.

3.4 Cavalos

A técnica da contagem pode ser elaborada ainda mais para o caso dos cavalos. Considere o segundo cavalo (sétima peça). Suponha que, após a inserção da sexta peça, haja 49 casas cobertas no total. Nesse caso, o sétimo cavalo e o rei deverão cobrir ao todo 15 casas. Como o rei pode cobrir no máximo oito, as sete restantes deveriam ser cobertas pelo cavalo. Para isso, ele não pode estar a menos de duas casas das bordas, pois, quando isso ocorre, pelo menos dois de seus possíveis movimentos não podem ser realizados.

Em resumo, o procedimento utilizado é o seguinte: calcula-se o número mínimo m de casas que devem ser cobertas pelo cavalo (considerando-se que as peças seguintes terão eficiência máxima, ou seja, cobrirão o maior número possível de casas). Se $m \leq 4$, permite-se que o cavalo seja colocado em qualquer posição do tabuleiro. Se $5 \leq m \leq 6$, não se admite que sejam consideradas as 28 casas das bordas. Se $m \geq 7$, o cavalo fica restrito apenas às 16 posições centrais do tabuleiro.

É interessante notar que a estratégia descrita pode ser aprimorada de forma a considerar os "cantos" do tabuleiro, que apresentam restrições à movimentação dos cavalos ainda maiores que as bordas. Isso não foi feito na implementação testada.

3.5 Cavalo e Rei

A última das técnicas de *prunning* apresentada é executada após a inserção do primeiro cavalo, a fim de eliminar configurações que não poderiam ser cobertas com o segundo cavalo e o rei.

A estratégia se baseia no fato de que o conjunto das posições potencialmente cobertas por um cavalo é descontínuo. Um cavalo é capaz de cobrir posições nas duas colunas à sua esquerda e nas duas à direita, mas nunca na própria coluna onde ele se encontra. O mesmo ocorre com as linhas. O rei, por outro lado, é capaz de cobrir casas em até 3 linhas (e colunas) consecutivas.

Considere as duas peças em conjunto. Uma delas cobre casas em quatro colunas distintas e a outra, em três. No entanto, o número máximo de colunas *consecutivas* cobertas pelo par é cinco, e não sete. Se o rei cobrir casas da coluna em que está o cavalo, as outras colunas sobre as quais ele terá alguma ação necessariamente serão atacadas pelo cavalo. Assim sendo, serão 5 as colunas sob ataque: duas exclusivamente pelo cavalo, uma apenas pelo rei e duas por ambos. Considere o outro caso: se o rei não cobrir a coluna em que está o cavalo, o número de colunas atacadas será no máximo 7, se o rei estiver a exatamente quatro colunas de distância. No entanto, como os dois pares de colunas sob ataque do cavalo estarão separados, o número máximo de colunas atacadas pelas duas peças em conjunto não pode ultrapassar 5.

Como conseqüência desses fatos, pode-se utilizar a seguinte estratégia: se, depois de inserido o primeiro cavalo, houver seis colunas (ou linhas) consecutivas com elementos não cobertos (independentemente do número ou da posição desses elementos), pode-se fazer o *prunning* do ramo. As duas últimas peças não seriam capazes de cobrir as casas restantes.

4 Implementação

Esta seção apresenta algumas características relevantes sobre a implementação do algoritmo para o problema das oito peças. Trata-se de uma estratégia de *branch-and-bound* com *backtracking*.

A árvore de resolução tem exatamente nove níveis. Inicia-se o primeiro nível com o tabuleiro vazio. Cada um dos estados analisados no segundo nível possui a rainha em uma posição diferente. No terceiro nível, cada nó tem o par (rainha, torre) em uma diferente configuração. Seguindo-se essa lógica, apenas os nós do nono nível – as folhas – possuem tabuleiros completos, ou seja, com as oito peças.

Um aspecto importante da implementação é a maneira pela qual as peças são adicionadas e, depois da chamada recursiva, retiradas do tabuleiro. Em especial, é importante que seja possível determinar, em todos os níveis, quais e quantas são as casas cobertas (ou não-cobertas). Essas informações são essenciais para o posicionamento das peças em níveis posteriores (afinal, duas peças não podem ocupar a mesma casa) e para a aplicação das estratégias de *prunning*.

Na implementação realizada, associaram-se a cada uma das posições do tabuleiro duas variáveis: uma delas (*board*) representa conteúdo da casa (qual peça ela contém ou a indicação de que a casa está vazia); a segunda variável (*count*) representa o número de peças que cobrem a posição.

A variável *count* permite que peças sejam adicionadas de forma eficiente. Quando uma peça é adicionada em uma posição p, o algoritmo incrementa o valor de *count* para todas as posições diretamente atacadas por ela. Além disso, ele determina (percorrendo a linha, a coluna e as diagonais que contêm p) quais das peças inseridas anteriormente atacam p. Para cada uma delas, a peça adicionada em p criará uma "sombra"; para as posições na sombra, o valor de *count* é decrementado.

O número total de casas cobertas (*covered*) é atualizado toda vez que algum valor *count* é reduzido de um para zero (*covered* é decrementado) ou aumentado de zero para um (*covered* é incrementado).

Para restaurar o tabuleiro ao seu estado "original" após uma chamada recursiva, a alternativa mais simples seria criar um cópia do tabuleiro antes de qualquer alteração e recuperá-la depois da chamada. Utilizou-se uma alternativa que, apesar de ter uma implementação mais complexa, garante maior eficiência. Sempre que o valor de *count* associado a uma casa é alterado, as coordenadas da casa e o valor original da variável são inseridos em uma pilha. Após a chamada recursiva, os elementos da pilha são removidos e utilizados para reconstituir o tabuleiro.

Para implementar eficientemente as estratégias *prunning* associadas ao rei individualmente (seção 3.3) e ao par cavalo-rei (3.5), associou-se uma variável a cada fila (linha e coluna) representando o número de casas desprotegidas. Sempre que a variável *count* correspondente a uma casa tem seu valor mudado de um valor positivo para 0 (ou de 0 para um valor positivo), as variáveis associadas à linha e à coluna são alteradas convenientemente.

5 Resultados

Utilizando-se as técnicas descritas nas seções anteriores, foi possível verificar que não é possível cobrir as 64 casas de um tabuleiro de xadrez utilizando apenas as oito peças mais poderosas.

A implementação do algoritmo foi feita em C++ e compilada com otimização total (*flag* `-O3`) pelo compilador `GCC`. Quando executado em um AMD K6-2 de 350 MHz, o programa precisou de 1h29min para analisar os cerca de 1,26 bilhão de nós da árvore de resolução (1.262.089.450, mais precisamente). Foram processados, portanto, aproximadamente 234 mil estados por segundo.

Repare que o número total de estados é mais de 1000 vezes menor que o limite superior mostrado em 3, de $1,4 \cdot 10^{12}$. Considerando-se o valor de $4,5 \cdot 10^{13}$ inicialmente apresentado, o número de estados de fato analisados é cerca de 35 mil vezes menor. Esses números demonstram a eficiência das estratégias de *prunning*.

De acordo com o discutido na seção 4, um ramo só terá nove nós quando o rei for de fato inserido em alguma posição; se qualquer uma das estratégias de *prunning* for bem-sucedida, o número de folhas será menor. De fato, observou-se que, dos aproximadamente 1,26 bilhão de estados, apenas 32 eram folhas. A tabela 1 mostra o número de estados em cada um dos demais níveis.

nível	peça	estados
1	-	1
2	Q	10
3	R	518
4	R	16.730
5	W	507.454
6	B	14.960.361
7	N	693.492.750
8	N	553.111.594
9	K	32

Tabela 1: Estados analisados em cada nível

A tabela permite que se observe claramente a eficiência das estratégias de *prunning*, especialmente nos dois últimos níveis. Até o sétimo nível, o número de estados cresce exponencialmente. Não fosse o *prunning*, o mesmo deveria ocorrer nos níveis seguintes. No entanto, o que se observa é uma pequena diminuição no número de estados entre o sétimo e o oitavo níveis e uma queda muito expressiva no último.

Soluções A figura 2 mostra a melhor solução encontrada pelo algoritmo. Apenas 61 das 64 casas do tabuleiro estão cobertas. Estão desprotegidas duas casas da sexta coluna (uma na sétima linha, outra na oitava) e a casa que contém o rei.

```
R . . . . . . .
. . . . N . . .
. . Q . . B . .
. . . . . . . .
. . . . . . . R
. . . . . . . .
W . . K N . . .
. . . . . . . .
```

Figura 2: Tabuleiro com 61 casas cobertas

Como algumas das estratégias de *prunning* se baseiam no fato de que apenas configurações com exatamente 64 casas cobertas são aceitáveis, não há garantias de que o algoritmo encontrará uma solução ótima, ou seja, que cubra o maior número possível de casas. Tudo o que o algoritmo é capaz de garantir é que não existe uma solução em que todo o tabuleiro esteja coberto.

De fato, é possível obter uma solução com 63 casas cobertas, conforme mostra a figura 3 (a casa não coberta é a vizinha da esquerda de `W`, o bispo branco). Como não existe maneira de cobrir as 64 casas, pode-se garantir que essa configuração é ótima. Essa solução foi encontrada a partir da execução de uma versão anterior do programa, em que as restrições ao posicionamento do rei não eram tão fortes (não havia a restrição de que todas as casas não cobertas estivessem contidas em um quadrado de lado 3).

```
Q . . . . . . .
. . N . . . R .
. . . . . . . .
. . . . . N . .
. . . B . . W .
. . K . . . . .
. . . . . . . .
. . . . . . . R
```

Figura 3: Solucão ótima, com 63 casas cobertas

6 Conclusão

O problema apresentado é um excelente exemplo de como técnicas adequadas permitir a resolução de problemas que, a princípio, parecem praticamente insolúveis. Considerações de simetria e estratégias de *prunning* permitiram que se reduzisse o número de estados analisados por um fator de 35 mil.

Uma extensão interessante desse trabalho seria generalizar as técnicas apresentadas, para que fossem resolvidos problemas semelhantes para conjuntos arbitrários de peças e tabuleiros de qualquer tamanho.

Para problemas relacionados que possuem solução, pode ser interessante utilizar técnicas adicionais para se chegar a elas mais rápido. Um problema semelhante ao discutido, por exemplo, é encontrar um tabuleiro em que todas as casas *vazias* estejam sob ataque de alguma peça (ou seja, não há a exigência que de que qualquer das oito casas ocupadas pelas peças esteja sob ataque). Esse problema tem diversas soluções (uma delas é mostrada na figura 4). Soluções podem ser encontradas em segundos se for utilizado um sistema de *branching* guloso, que dá preferência às posições que maximizam o número de casas cobertas. Além disso, limitar o número de subárvores de cada nó da árvore de solução a uma constante pequena (4 ou 5) também contribui para uma diminuição no tempo de execução. No entanto, nesse caso, perde-se a garantia de correção do algoritmo: se a constante for muito pequena, ele pode não encontrar a solução.

```
R . . . . . . .
. R . . . . . .
. . . . . B . .
. . . . . N . W
. . . . Q . . .
. N . . . . . .
. . . . . . K .
. . . . . . . .
```

Figura 4: Tabuleiro em que as 56 casas vazias estão sob ataque