Atividade Extra 2 - Backtracking e Branch And Bound

Felipe Campolina Soares de Paula[1]

[1]ICEI - Pontifícia Universidade Católica de Minas Gerais

1. Introdução

Neste trabalho, exploram-se os métodos de backtracking e branch-and-bound, destacando sua relevância e aplicabilidade na resolução de problemas computacionais que envolvem busca e otimização. O método de backtracking é uma técnica de busca por tentativa e erro, frequentemente aplicada em contextos onde é necessário percorrer todas as possíveis configurações para resolver problemas de decisão ou otimização [Knuth 1977]. Por outro lado, o branch-and-bound é uma abordagem sistemática para resolver problemas de otimização global que consiste em dividir o espaço de busca em subconjuntos menores, mais gerenciáveis e explorar essas ramificações de maneira eficiente [Land and Doig 1960].

Ambos os métodos compartilham a filosofia de explorar o espaço de soluções de uma maneira estruturada, porém diferem significativamente em suas abordagens e eficácia dependendo do tipo de problema tratado. Este trabalho visa não apenas descrever os fundamentos teóricos de cada método, mas também demonstrar através de exemplos de caso selecionados como essas técnicas podem ser aplicadas para encontrar soluções ótimas ou satisfatórias em problemas complexos de diferentes áreas, como inteligência artificial, pesquisa operacional e ciência da computação [Russell and Norvig 2020, Hillier and Lieberman 2001].

Ao proporcionar uma análise comparativa dessas estratégias de busca, busca-se oferecer uma visão clara de suas potencialidades e limitações, fornecendo assim um guia prático para pesquisadores e profissionais da área que necessitem escolher a técnica mais adequada para seus desafios específicos.

2. Backtracking

O backtracking é uma estratégia de busca recursiva que melhora a abordagem de força bruta ao resolver problemas complexos de decisão, otimização e enumeração, onde a solução envolve múltiplas escolhas sequenciais. Enquanto a força bruta tenta todas as possíveis configurações sem qualquer discriminação, o backtracking aprimora essa técnica ao permitir a reversão de decisões anteriores, eliminando caminhos claramente sem saída e reduzindo significativamente o espaço de busca. Essa metodologia de tentativa e erro é eficaz em cenários onde é essencial explorar todas as configurações possíveis, mas com o benefício adicional de interromper a exploração quando uma condição de falha é identificada, evitando assim a avaliação desnecessária de soluções inviáveis.

2.1. Princípios e Conceitos Fundamentais

Os termos fundamentais relacionados ao backtracking incluem[gee 2022]:

- **Candidato:** Um candidato é uma escolha potencial ou elemento que pode ser adicionado à solução atual para construir progressivamente uma solução completa.

- **Solução:** A solução é uma configuração válida e completa que satisfaz todas as restrições do problema.
- **Solução Parcial:** Uma solução parcial é uma configuração intermediária ou incompleta que está sendo construída durante o processo de backtracking.
- **Espaço de Decisão:** O espaço de decisão compreende o conjunto de todos os candidatos ou escolhas possíveis em cada ponto de decisão.
- **Ponto de Decisão:** Um ponto de decisão é uma etapa específica no algoritmo onde um candidato é escolhido e adicionado à solução parcial.
- **Solução Viável:** Uma solução viável é uma solução parcial ou completa que adere a todas as restrições impostas.
- **Caminho sem Saída:** Um beco sem saída ocorre quando uma solução parcial não pode ser estendida sem violar as restrições.
- **Retroceder (Backtrack):** O backtrack envolve desfazer decisões anteriores e retornar a um ponto de decisão anterior para explorar outras possibilidades.
- **Espaço de Busca:** O espaço de busca inclui todas as combinações possíveis de candidatos e escolhas.
- **Solução Ótima:** Em problemas de otimização, a solução ótima é a melhor solução possível, que maximiza ou minimiza a função objetivo.

Além disso, o backtracking é uma técnica de busca recursiva que, apesar de poder resolver qualquer problema de satisfação de restrições, nem sempre é a abordagem mais eficiente. Embora o backtracking possa, em teoria, ser aplicado a uma vasta gama de problemas, sua complexidade exponencial o torna menos eficiente do que outras abordagens, como programação dinâmica ou algoritmos gulosos, que operam em tempo logarítmico ou polinomial para certos tipos de problemas. Contudo, ainda existem muitos problemas para os quais o backtracking é a única solução viável.

Problema: Imagine que você está em um labirinto complexo com múltiplos caminhos e obstáculos. Sua tarefa é encontrar um caminho até a saída.

Por que a programação dinâmica falha: A solução para um determinado ponto no labirinto não depende apenas de soluções para outros pontos, mas também da sequência específica de movimentos realizados para chegar lá, o que a programação dinâmica não pode efetivamente modelar sem uma representação exaustiva de todas as possíveis sequências de estados.

Por que os algoritmos gulosos falham: Um algoritmo guloso poderia optar por sempre seguir o caminho que parece mais curto ou mais promissor, o que poderia facilmente levar a um beco sem saída ou não garantir a rota mais curta, dado que a decisão local ótima não necessariamente resulta na solução global ótima.

Por que o Backtracking funciona: O backtracking explora ativamente todas as possíveis direções em cada junção do labirinto, marcando caminhos já testados e revertendo quando um caminho leva a um beco sem saída. Esse método permite uma exploração completa e sistemática, garantindo que todas as opções sejam consideradas e que a rota de saída, se existir, seja encontrada.

2.2. Exemplo de Funcionamento

Nesta seção, será mostrado o funcionamento do algoritmo de backtracking a partir de um exemplo fictício, utilizando o problema das N Rainhas.

2.2.1. Exemplo: O Problema das N Rainhas

Considere um tabuleiro de xadrez de 4x4 onde queremos posicionar 4 rainhas de forma que nenhuma rainha esteja em posição de atacar outra. O algoritmo de backtracking explora as possíveis posições para cada rainha, uma linha por vez, e retrocede quando uma configuração inviável é encontrada.

2.2.2. Visualização Passo a Passo com Explicações Detalhadas

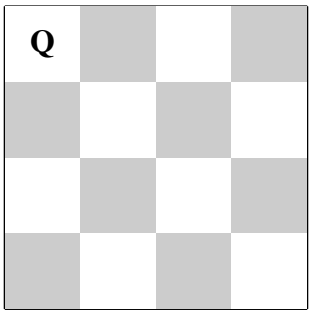

Figure 1. Passo 1: Colocação da primeira rainha na posição (0,0). Não há conflitos, pois é a primeira rainha no tabuleiro.

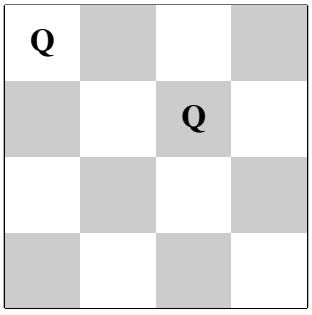

Figure 2. Passo 2: A segunda rainha é colocada na posição (1,2). Esta posição é escolhida pois não está em linha, coluna ou diagonal com a primeira rainha.

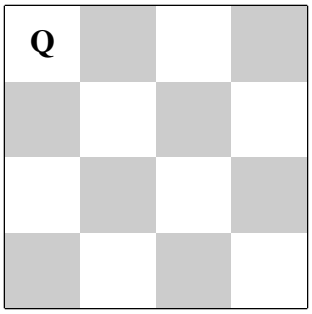

Figure 3. Passo 3: Retrocesso. Não há posições viáveis para a terceira rainha na terceira linha, levando a um retrocesso. A busca então revisita a colocação da segunda rainha. Assim, a segunda rainha é removida devido à falta de posições viáveis para a terceira rainha.

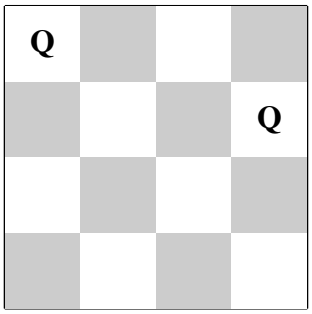

Figure 4. Passo 4: A segunda rainha é reposicionada na posição (1,3).

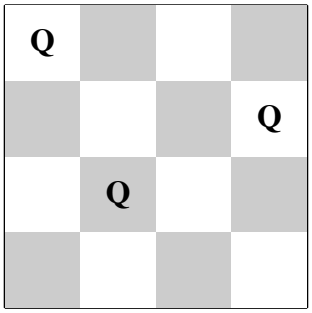

Figure 5. Passo 5: A terceira rainha é corretamente colocada na posição (2,1), onde não é atacada pelas duas rainhas anteriores.

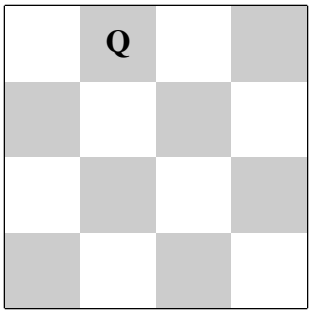

Figure 6. Passo 6: Retrocesso. Não há posições viáveis para a quarta rainha, levando a um retrocesso. Assim reposiciona rainha 1 para (0,1)

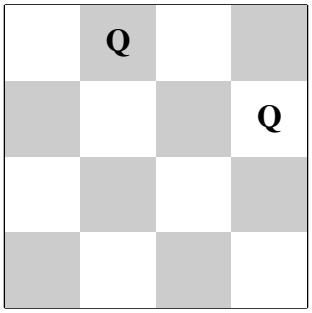

Figure 7. Passo 7: A segunda rainha é posicionada na (1,3)

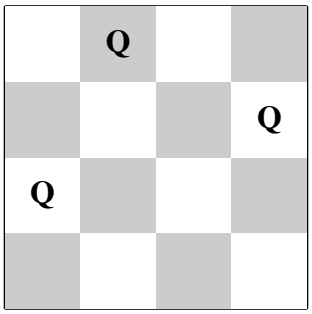

Figure 8. Passo 8: A terceira rainha é posicionada na (2,0)

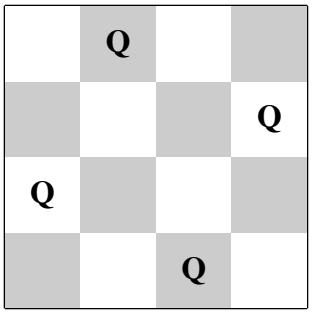

Figure 9. Passo 9: A quarta rainha é posicionada na (3,2)

2.2.3. Pseudo-Código

Algorithm 1 Pseudocódigo para encontrar soluções usando backtracking

```
1:  function ENCONTRAR_SOLUCOES(parametros)
2:      if solucao_valida then
3:          armazena_solucao
4:          return
5:      end if
6:      for cada opcao do
7:          if opcao_valida then
8:              Aplica(opcao)
9:              ENCONTRAR_SOLUCOES(parametros)
10:             Retroceder(remove opcao)
11:         end if
12:     end for
13:     return
14: end function
```

2.2.4. Aplicações do Backtracking

O backtracking é uma estratégia algorítmica empregada em uma variedade de campos para resolver problemas computacionais que envolvem a busca por soluções ótimas ou aceitáveis sob determinadas restrições. Esta técnica é especialmente eficaz em cenários onde todas as possíveis configurações de um problema precisam ser examinadas para encontrar a solução mais adequada. A seguir, são apresentadas algumas das principais aplicações do backtracking:

Problemas de Satisfação de Restrições

Problemas como o Sudoku e o N-Rainhas são exemplos clássicos onde o backtracking é utilizado para encontrar uma configuração de elementos que satisfaça todas as restrições impostas. No Sudoku, o objetivo é preencher uma grade 9x9 com dígitos de 1 a 9 de forma que cada linha, cada coluna e cada um dos nove subgrades 3x3 contenham todos os dígitos de 1 a 9. No problema das N-Rainhas, deve-se posicionar N rainhas em um tabuleiro de xadrez N x N de modo que nenhuma rainha ataque outra.

Otimização Combinatória

O backtracking é aplicado em problemas de otimização combinatória, como o Problema do Caixeiro Viajante (PCV), onde o objetivo é encontrar o menor caminho possível que visita cada cidade uma vez e retorna à cidade de origem. A capacidade de explorar sistematicamente todas as permutações possíveis de cidades faz do backtracking uma ferramenta valiosa para esses tipos de problemas, especialmente quando o número de cidades não é excessivamente grande.

Geração de Permutações

A geração de todas as permutações possíveis de um dado conjunto é outra aplicação direta do backtracking. Este processo é útil em vários contextos, como criptografia para testar diferentes combinações de chaves ou em bioinformática para analisar todas as possíveis sequências de aminoácidos em um peptídeo.

Decisão e Coloração de Grafos

O backtracking também é amplamente usado em problemas de decisão e coloração de grafos, como a coloração de mapas e a verificação de bipartidade de grafos. Por exemplo, o problema de coloração de grafos envolve atribuir cores a todos os vértices de um grafo de tal maneira que dois vértices adjacentes não compartilhem a mesma cor, usando o menor número possível de cores.

Jogos e Puzzles

Em jogos e puzzles, o backtracking pode ser utilizado para resolver enigmas que requerem movimentos sequenciais baseados em decisões, como em labirintos ou outros jogos lógicos onde é necessário encontrar uma sequência de ações que leve à vitória ou solução do puzzle.

Estas aplicações demonstram a flexibilidade e a potência do backtracking em diversas áreas, desde jogos e entretenimento até problemas científicos e matemáticos complexos.

3. Branch And Bound(Ramificar e limitar)

O algoritmo de Branch and Bound é uma técnica sistemática e versátil utilizada para resolver problemas de otimização, particularmente aqueles de natureza combinatória e inteira [Clausen 1999]. Este método distingue-se por sua abordagem estruturada que divide o problema original em subproblemas menores, mais manejáveis, que são explorados de maneira sequencial [Morrison 2016]. A essência do Branch and Bound reside na combinação de dois conceitos operacionais fundamentais: ”branching”, que expande a árvore de decisão ao dividir um problema em vários subproblemas, e ”bounding”, que utiliza limites superiores e inferiores para avaliar as regiões do espaço de busca, permitindo assim a poda de ramificações que não podem conter a solução ótima [Land and Doig 2010]. Esta técnica é amplamente aplicada em uma variedade de campos, desde pesquisa operacional e engenharia até economia e ciência da computação, proporcionando soluções eficientes para problemas que de outra forma seriam proibitivamente caros ou inviáveis de resolver.

3.1. Princípios e Conceitos Fundamentais

O algoritmo de Branch and Bound baseia-se em uma estrutura de árvore de decisões para explorar sistematicamente todas as soluções possíveis de um problema de otimização. A eficácia deste método deriva de sua capacidade de eliminar grandes partes do espaço de busca que não contêm a solução ótima, usando uma combinação de técnicas de ramificação (branching) e limitação (bounding). A seguir, descrevemos os principais conceitos envolvidos:

- **Branching:** Este processo envolve a divisão do problema original em subproblemas menores e mais gerenciáveis, chamados de ”nós” da árvore de busca. Cada nó representa uma decisão parcial ou um subconjunto do espaço de soluções. A árvore de busca é expandida através de sucessivas ramificações até que as soluções sejam suficientemente refinadas.
- **Bounding:** Após a ramificação, o algoritmo emprega técnicas de limitação para estimar limites superiores ou inferiores para a solução ótima nos subproblemas. Estes limites são usados para avaliar se um subconjunto pode conter uma solução válida e potencialmente ótima. Se um limite indicar que um subproblema não pode produzir uma solução melhor do que a melhor solução já encontrada, esse subproblema é descartado ou ”poda” da árvore de busca.
- **Poda:** A poda é o processo de eliminar ramos da árvore de busca que não são viáveis ou que não podem melhorar a solução atual. Este processo reduz significativamente o número de configurações que precisam ser examinadas.

A combinação desses elementos permite que o Branch and Bound seja uma ferramenta poderosa para problemas complexos de decisão e otimização, proporcionando uma maneira estruturada e eficiente de navegar por grandes espaços de soluções.

3.2. Exemplo de Funcionamento

Nesta seção, será mostrado o funcionamento do algoritmo de Branch and Bound a partir de um exemplo fictício, utilizando o problema das N Rainhas.

3.2.1. Exemplo: O Problema das N Rainhas

O problema das N Rainhas envolve posicionar N rainhas em um tabuleiro de xadrez de N x N de modo que nenhuma rainha esteja em posição de atacar outra. Para um tabuleiro de 4x4, o algoritmo de Branch and Bound explora sistematicamente todas as possíveis configurações de posicionamento das rainhas, enquanto evita eficientemente caminhos que não levam a uma solução.

3.2.2. Visualização Passo a Passo com Explicações Detalhadas

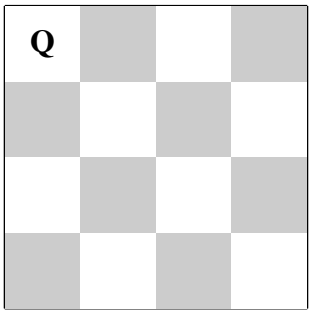

Figure 10. Passo 1: Colocação da primeira rainha na posição (0,0).

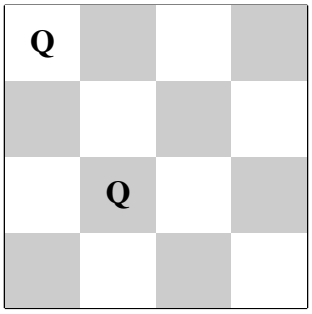

Figure 11. Passo 2: Na segunda tentativa, tentamos colocar uma rainha na (2,1), e logo depois removemos ela por causa do backtracking.

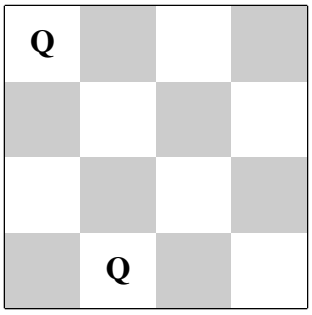

Figure 12. Passo 3: Na terceira tentativa, tentamos colocar uma rainha na (3,1).

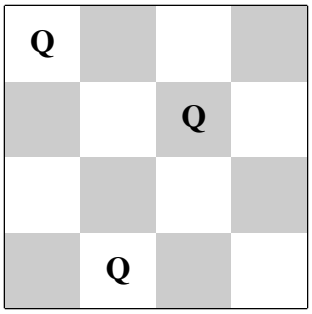

Figure 13. Passo 4: Na quarta tentativa, tentamos colocar uma rainha na (1,2). Entretnto, o alogitimo percebe que não existe outra possibilidade de posicionamento sem gerar conflito, assim , voltamos ao tabuleiro vazio para reposicionamento da primeria rainha

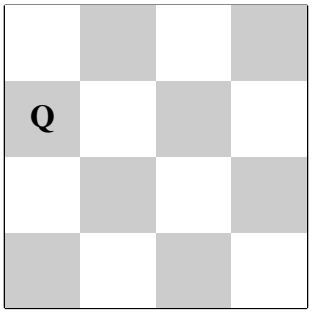

Figure 14. Passo 5:Reposicionamos a primeira rainha para (1,0)

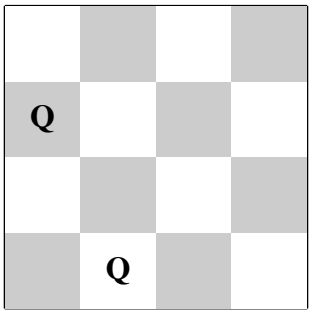

Figure 15. Passo 6:Reposicionamos a segunda rainha para (3,1)

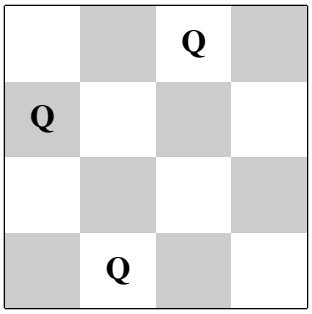

Figure 16. Passo 7:Reposicionamos a terceira rainha para (0,2)

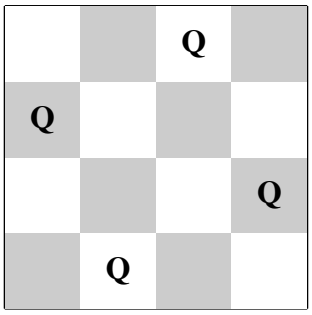

Figure 17. Passo 8:Posicionamos a quarta rainha para (2,3)

3.3. Pseuso-Código

Algorithm 2 Exemplo de Branch and Bound

Dados: Matriz de custo $M[n][n]$
Resultado: Atribuição de trabalhos a cada trabalhador segundo o custo ótimo

```
1:  function CustoMin(M)
2:      while Verdadeiro do
3:          E ← MenorCusto()
4:          if E é um nó folha then
5:              imprime()
6:              retorna
7:          end if
8:          for cada filho S de E do
9:              Adiciona(S)
10:             S.parent ← E
11:         end for
12:     end while
13: end function
```

3.4. Aplicaçoes

O algoritmo Branch and Bound tem uma ampla gama de aplicações em problemas de otimização combinatória. Alguns dos problemas mais comuns incluem:

- **Problema do Caixeiro Viajante (Traveling Salesman Problem - TSP):** Dado um conjunto de cidades e as distâncias entre elas, o objetivo é encontrar o menor caminho que visita cada cidade exatamente uma vez e retorna à cidade de origem.
- **Problema da Mochila (Knapsack Problem):** Dado um conjunto de itens, cada um com um peso e um valor, e uma mochila com capacidade limitada, o objetivo é maximizar o valor total dos itens que podem ser colocados na mochila sem exceder sua capacidade.
- **Programação Linear Inteira (Integer Linear Programming - ILP):** O Branch and Bound é comumente usado para resolver problemas de programação linear onde algumas ou todas as variáveis de decisão devem ser inteiros.
- **Problema das N-Rainhas:** O Branch and Bound pode ser aplicado para encontrar todas as soluções possíveis para o problema de colocar N rainhas em um tabuleiro de xadrez de tamanho N × N sem que elas se ataquem.
- **Roteamento de Veículos:** Em problemas de roteamento de veículos, o Branch and Bound pode ser utilizado para encontrar o caminho mais eficiente para múltiplos veículos atenderem a um conjunto de clientes.

Essas são apenas algumas das muitas aplicações do algoritmo Branch and Bound em problemas do mundo real.

4. Comparação entre algorimos

Nessa seção sera apresentado uma tabela que mostra uma comparação entre os algoritimos:

Parâmetro	Backtracking	Branch and Bound
Abordagem	O Backtracking é usado para encontrar todas as soluções possíveis disponíveis para um problema. Quando percebe que fez uma escolha ruim, desfaz a última escolha retrocedendo. Ele pesquisa a árvore de espaço de estados até encontrar uma solução para o problema.	O Branch and Bound é usado para resolver problemas de otimização. Quando percebe que já possui uma solução ótima melhor do que a que a pré-solução leva, ele abandona essa pré-solução. Ele pesquisa completamente a árvore de espaço de estados para obter uma solução ótima.
Travessia	O Backtracking atravessa a árvore de espaço de estados de maneira DFS (Busca em Profundidade).	O Branch and Bound atravessa a árvore de qualquer maneira, DFS ou BFS (Busca em Largura).
Função	O Backtracking envolve uma função de viabilidade.	O Branch and Bound envolve uma função de limite.
Problemas	O Backtracking é usado para resolver problemas de decisão.	O Branch and Bound é usado para resolver problemas de otimização.
Busca	No backtracking, a árvore de espaço de estados é pesquisada até obter a solução.	No Branch and Bound, como a solução ótima pode estar em qualquer lugar da árvore de espaço de estados, a árvore precisa ser completamente pesquisada.
Eficiência	O Backtracking é mais eficiente.	O Branch and Bound é menos eficiente.
Aplicações	Útil na resolução do Problema das N-Rainhas, Soma de Subconjuntos, Problema do Ciclo de Hamilton, Problema de Coloração de Grafos.	Útil na resolução do Problema da Mochila, Problema do Caixeiro Viajante.
Solução	Na busca bem-sucedida da solução na árvore de espaço de estados, a pesquisa é interrompida.	A árvore de espaço de estados inteira é pesquisada para encontrar uma solução ótima.

References

(2022). Introduction to backtracking - data structure and algorithm tutorials - geeks for geeks. Acessado em 10 de Junho de 2024.

Clausen, J. (1999). Branch and bound algorithms—principles and examples. *Department of Computer Science, University of Copenhagen*, 99:1–30.

Hillier, F. S. and Lieberman, G. J. (2001). *Introduction to Operations Research*. McGraw-Hill, 7 edition.

Knuth, D. E. (1977). *The Art of Computer Programming, Volume 4*, volume 4. Addison-Wesley.

Land, A. H. and Doig, A. G. (1960). An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520.

Land, A. H. and Doig, A. G. (2010). *Branch and Bound: A Survey*. Wiley.

Morrison, D. R. (2016). *Branch, Bound, and Remember*, pages 113–130. Springer.

Russell, S. J. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson, 4 edition.