# Uma argumentação entre *Backtracking* e *Branch and Bound*

**Gustavo Lopes Rodrigues**[1], **Rafael Amauri Diniz Augusto**[2]

[1]Instituto de Ciências Exatas e Informática –
Pontifícia Universidade Católica de Minas Gerais(PUC-MG)

***Abstract.*** *Branch and Bound and Backtracking are two strategies in algorithm development. The objective of this project for the discipline of Algorithm Design and Analysis is talk about such strategies indicating, at least, the concepts, when it should be used, in addition to showing examples. We will also present the main similarities and differences between them. Also, if there are similarities or differences with other strategies such as Greedy Approach, Dynamic Programming and Division and Conquer, we will present them in a way to broaden the discussion*

***Resumo.*** *Branch and Bound e Backtracking são duas estratégias no desenvolvimento de algoritmos. O objetivo deste trabalho feito para a disciplina de Projeto e Análise de Algoritmos é discorrer sobre tais estratégias indicando, pelo menos, os conceitos, quando se deve utilizar, além de mostrar exemplos. Apresentaremos também as principais semelhanças e diferenças entre elas. Ainda, se houver semelhanças ou diferenças com outras estratégias como Abordagem Gulosa, Programação Dinâmica e Divisão e Conquista, apresentaremos elas de forma a ampliar a discussão*

## 1. Informações gerais

Em discussões sobre desenvolvimento de algoritmos, a escolha da estratégia para resolver um problema é fundamental, visto que existe métodos diferentes, que possuem aplicações diferentes, e com complexidades diferentes. Os principais em questão, são o de *Backtracking*, e *Branch and Bound*. Ambos são relativamente parecidos em conceito, porém, possuem implementações diferentes, o que resulta em diferentes circunstâncias na qual um é usado no lugar do outro.

A partir deste artigo, iremos analisar os dois algoritmos, olhando com cuidado para os conceitos para então construir códigos em Python para resolver problemas conhecidos da computação.Por fim, iremos dar uma breve olhada em outras estratégias, e comparar com as abordagens já mencionadas.

## 2. Backtracking

Backtracking é um tipo de algoritmo que representa um refinamento da busca por força bruta, em que múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas. O termo foi cunhado pelo matemático estado-unidense D. H. Lehmer na década de 1950.

O procedimento é usado em linguagens de programação como Prolog. Uma busca inicial em um programa nessa linguagem segue o padrão busca em profundidade, ou seja, a árvore é percorrida sistematicamente de cima para baixo e da esquerda para direita.

Quando essa pesquisa falha, ou é encontrado um nodo terminal da árvore, entra em funcionamento o mecanismo de backtracking. Esse procedimento faz com que o sistema retorne pelo mesmo caminho percorrido com a finalidade de encontrar soluções alternativas.

### 2.1. Problema do labirinto

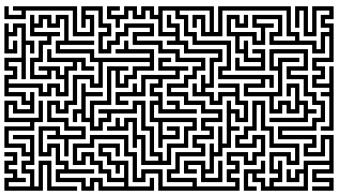

**Figure 1. Um exemplo de labirinto**

Problema do labirinto

## 3. Branch and Bound

O método de Ramificar e limitar (em inglês, Branch and bound) é um algoritmo para encontrar soluções ótimas para vários problemas de otimização, especialmente em otimização combinatória. Consiste em uma enumeração sistemática de todos os candidatos a solução, através da qual grandes subconjuntos de candidatos infrutíferos são descartados em massa utilizando os limites superior e inferior da quantia otimizada.

O método foi proposto por A. H. Land e A. G. Doig em 1960 para programação discreta. É utilizado para vários problemas NP-completos como o problema do caixeiro viajante e o problema da mochila.

### 3.1. Atribuição de trabalho

| | **Job 1** | **Job 2** | **Job 3** |
|---|---|---|---|
| **A** | 9 | 3 | 4 |
| **B** | 7 | 8 | 4 |
| **C** | 10 | 5 | 2 |

## 4. Outras estratégias

Section titles must be in boldface, 13pt, flush left. There should be an extra 12 pt of space before each title. Section numbering is optional. The first paragraph of each section should not be indented, while the first lines of subsequent paragraphs should be indented by 1.27 cm.

### 4.1. Abordagem gulosa

Algoritmo guloso ou míope é técnica de projeto de algoritmos que tenta resolver o problema fazendo a escolha localmente ótima em cada fase com a esperança de encontrar um ótimo global.

Na solução de alguns problemas combinatórios a estratégia gulosa pode assegurar a obtenção de soluções ótimas, o que não é muito comum. No entanto, quando o problema a ser resolvido pertencer à classe NP-completo ou NP-difícil, a estratégia gulosa torna-se atrativa para a obtenção de solução aproximada em tempo polinomial.

### 4.2. Divisão e conquista

Divisão e Conquista (do inglês Divide and Conquer) em computação é uma técnica de projeto de algoritmos utilizada pela primeira vez por Anatolii Karatsuba em 1960 no algoritmo de Karatsuba.

Esta técnica consiste em dividir um problema maior recursivamente em problemas menores até que o problema possa ser resolvido diretamente. Então a solução do problema inicial é dada através da combinação dos resultados de todos os problemas menores computados. Vários problemas podem ser solucionados através desta técnica, como a ordenação de números através do algoritmo merge sort e a transformação discreta de Fourier através da transformada rápida de Fourier. Outro problema clássico que pode ser resolvido através desta técnica é a Torre de Hanoi.

#### 4.2.1. Pares de pontos próximos

### 4.3. Programação Dinâmica

Programação dinâmica é um método para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. Ela é aplicável a problemas nos quais a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada - de forma a evitar recálculo - de outros subproblemas que, sobrepostos, compõem o problema original.

O que um problema de otimização deve ter para que a programação dinâmica seja aplicável são duas principais características: subestrutura ótima e superposição de subproblemas. Um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém em seu interior soluções ótimas para subproblemas. A superposição de subproblemas acontece quando um algoritmo recursivo reexamina o mesmo problema muitas vezes.

#### 4.3.1. Fibonnaci

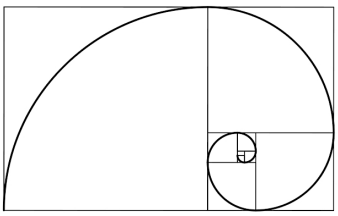

**Figure 2. Uma imagem ilustrativa da sequência Fibonnaci**

## 5. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

## References

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.