



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE  
CENTRO DE CIÊNCIAS EXATAS E DA TERRA  
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA  
DISCIPLINA DE LINGUAGENS FORMAIS E AUTÔMATOS



# Análise do uso de RegEx na biblioteca Glycowork

Andriel Vinicius de Medeiros Fernandes,  
Gabrielle de Vasconcelos Borja,  
Jeremias Pinheiro de Araújo Andrade,  
Lucas Vinicius Dantas de Medeiros,  
María Paz Marcato,  
Ramon Cândido Jales de Barros

Natal-RN  
Outubro, 2025

Nome completo do autor

## Análise de uso de RegEx na biblioteca Glycowork

Pesquisa elaborada e apresentada para a disciplina DIM0606 - Linguagens Formais e Autômatos, ofertada pelo Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte e ministrada no semestre 2025.2 pelo Prof. Dr. Valdigleis da Silva Costa, como requisito parcial para a obtenção de nota para a 1<sup>a</sup> unidade.

DIMAP – DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA  
CCET – CENTRO DE CIÊNCIAS EXATAS E DA TERRA  
UFRN – UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Natal-RN

Outubro, 2025

# Análise do uso de RegEx na biblioteca Glycowork

Autores: Andriel Vinicius de Medeiros Fernandes,  
Gabrielle de Vasconcelos Borja,  
Jeremias Pinheiro de Araújo Andrade,  
Lucas Vinicius Dantas de Medeiros,  
María Paz Marcato,  
Ramon Cândido Jales de Barros  
Professor: Valdigleis

## RESUMO

O resumo deve apresentar de forma concisa os pontos relevantes de um texto, fornecendo uma visão rápida e clara do conteúdo e das conclusões do trabalho. O texto, redigido na forma impessoal do verbo, é constituído de uma seqüência de frases concisas e objetivas e não de uma simples enumeração de tópicos, não ultrapassando 500 palavras, seguido, logo abaixo, das palavras representativas do conteúdo do trabalho, isto é, palavras-chave e/ou descritores. Por fim, deve-se evitar, na redação do resumo, o uso de parágrafos (em geral resumos são escritos em parágrafo único), bem como de fórmulas, equações, diagramas e símbolos, optando-se, quando necessário, pela transcrição na forma extensa, além de não incluir citações bibliográficas.

*Palavras-chave:* Palavra-chave 1, Palavra-chave 2, Palavra-chave 3.

# Lista de figuras

1	Teste de uma figura em formato .png. . . . .	p. 12
2	Implementação da função process_pattern . . . . .	p. 14
3	Aplicação da função process_pattern . . . . .	p. 15
4	Implementação da função specify_linkages . . . . .	p. 16
5	Aplicação da função specify_linkages . . . . .	p. 16
6	Implementação da função get_match . . . . .	p. 17
7	Aplicação da função specify_linkages . . . . .	p. 17

# Lista de tabelas

# Lista de abreviaturas e siglas

ReGex – Expressões Regulares

# Sumário

<b>1 Referencial Teórico</b>	p. 8
1.1 Glicanos: Complexidade e Importância Biológica . . . . .	p. 8
1.2 RegEx na Ciência da Computação . . . . .	p. 9
1.3 A Aplicação Inovadora de RegEx em Glicanos . . . . .	p. 9
<b>2 Capítulo 2</b>	p. 10
2.1 Estrutura e funções do módulo . . . . .	p. 10
2.2 Operação interna e papel das expressões regulares . . . . .	p. 12
2.3 Vantagens da RegEx no contexto do <code>glycowork</code> . . . . .	p. 13
<b>3 Funções chave e eventos de código</b>	p. 14
3.1 <code>preprocess_pattern</code> . . . . .	p. 14
3.1.1 Exemplo de uso . . . . .	p. 14
3.1.2 Comportamento de entrada . . . . .	p. 15
3.2 <code>specify_linkages</code> . . . . .	p. 15
3.2.1 Exemplo de uso . . . . .	p. 16
3.2.2 Comportamento de entrada . . . . .	p. 16
3.3 <code>get_match</code> . . . . .	p. 16
3.3.1 Exemplo de uso . . . . .	p. 17
3.3.2 Comportamento de entrada . . . . .	p. 17
<b>4 Considerações finais</b>	p. 19
4.1 Principais contribuições . . . . .	p. 19

4.2	Limitações . . . . .	p. 19
4.3	Trabalhos futuros . . . . .	p. 19
<b>Referências</b>		p. 20



# 1 Referencial Teórico

O trabalho feito na Universidade de Gotemburgo, na Suécia, tem como bases duas áreas distintas, a biologia e a ciência da computação. Na área biológica, o foco é sobre os glicanos, biopolímeros com inúmeras aplicações e funções biológicas. Já na computação, foi estudado o uso de expressões regulares (RegEx ), padrões formados por sequências de caracteres utilizados para busca, análise e manipulação de textos. Dessa forma, os pesquisadores aplicaram expressões regulares para formalizar uma estrutura de busca voltada à identificação de padrões específicos dos glicanos.

## 1.1 Glicanos: Complexidade e Importância Biológica

Para compreender melhor a abordagem biológica do estudo, é importante explorar os glicanos, suas características e funções, bem como outros conceitos relacionados abordados no trabalho.

**Definição 1.1.1** (Definição de Glicanos). *Glicanos são polissacarídeos estruturais, longas cadeias formadas por unidades de açúcar (monossacarídeos) ligadas entre si por ligações glicosídicas, presentes abundantemente na Terra como componentes importantes de estruturas como glicoproteínas, glicolipídeos e proteoglicanos, além de fazerem parte de paredes celulares de fungos e leveduras.*

Essa diversidade de funções ocorre porque, em sua composição, existem subestruturas chamadas de motivos de glicanos, uma sequência ou arranjo particular de açúcares que funciona como um sinal de reconhecimento molecular, ou seja, a parte que carrega o significado mais importante e que é reconhecida por outras moléculas. São eles que trazem a importância biológica dos glicanos e, por isso, é de extrema importância que eles sejam entendidos pela ciência.

Contudo, a grande diversidade e versatilidade traz à estrutura interesse contínuo de pesquisas de diversas áreas, como a imunologia, biotecnologia e parasitologia. Porém,

como existem inúmeros motivos de glicanos, catalogá-los e reconhecê-los é um desafio.

## 1.2 RegEx na Ciência da Computação

Na ciência da computação, RegEx são uma sequência de caracteres que define um padrão de busca. Elas representam uma linguagem formal que pode ser utilizada para identificar, extrair e manipular subconjuntos de texto com base em regras e padrões específicos.

Essa notação envolve uma combinação de cadeias de símbolos do alfabeto da linguagem. Por exemplo, o operador main (+) é utilizado para denotar união, o asterisco (\*) para o fecho estrela, indica zero ou mais ocorrências do caractere anterior, e o ponto (.) para concatenação. Essa sintaxe permite a criação de padrões que vão desde buscas simples, como encontrar uma palavra, até a validação de estruturas complexas em um texto.

## 1.3 A Aplicação Inovadora de RegEx em Glicanos

Diante do desafio na análise dos glicanos, o estudo propôs a aplicação de expressões regulares como ferramenta para buscar e extrair seus padrões estruturais de forma precisa e flexível. Essa técnica foi adaptada no trabalho para lidar com a estrutura ramificada e não linear dos glicanos, permitindo uma análise muito mais eficiente e escalável. O artigo demonstra que, embora as aplicações tradicionais de RegEx sejam conhecidas, sua versatilidade permite o uso em contextos menos convencionais, como a bioinformática.

## 2 Capítulo 2

Neste capítulo, descrevemos o funcionamento do módulo `glycowork.motif.regex`, suas funções principais, a atuação das expressões regulares e as vantagens do uso desse sistema.

O módulo `glycowork.motif.regex`, introduzido por Bennett e Bojar (2024) no pacote `glycowork`, representa um avanço significativo na análise computacional de glicanos ao incorporar o uso de expressões regulares (Regex) para identificação e extração de padrões estruturais complexos. Essa abordagem adapta a lógica das Regex convencionais, amplamente estudadas e utilizadas em ciência da computação para detecção de padrões em cadeias de texto, ao contexto estrutural e não linear dos glicanos, permitindo buscas mais precisas e flexíveis em sequências de carboidratos.

### 2.1 Estrutura e funções do módulo

O módulo foi desenvolvido para permitir que pesquisadores expressem padrões estruturais de glicanos em uma forma análoga às expressões regulares de texto. Ele se baseia na tradução de padrões definidos em formato de expressão regular para operações de isomorfismo de subgrafos dentro das estruturas moleculares de glicanos. Quando o usuário fornece um padrão, por exemplo, um motif glicosídico como `Neu5Ac $\alpha$ 2-3Gal $\beta$ 1-4(Fuc $\alpha$ 1-3)GlcNAc`, o sistema decompõe esse padrão em unidades menores, chamadas de módulos homogêneos, correspondentes a monossacarídeos e ligações individuais. Cada módulo é então processado para identificar as possíveis correspondências no grafo do glicano, permitindo localizar subestruturas equivalentes de forma independente da forma textual em que o glicano foi representado. Isso é essencial, já que o módulo aceita múltiplos formatos de entrada, como WURCS, GlycoCT, Oxford, IUPAC-condensed e outras notações padronizadas.

As expressões regulares glicosídicas seguem uma lógica análoga à das regex textuais,

com suporte a modificadores, quantificadores e operadores de busca contextual, como *lookahead* e *lookbehind*. Isso permite definir quantas vezes um determinado monossacarídeo pode aparecer, se uma ligação é opcional ou se uma ramificação deve ocorrer em uma posição específica. O sistema aceita curingas, como o caractere “.” ou o termo *Monosaccharide*, que funcionam como substitutos para qualquer unidade estrutural. Os ramos das moléculas são representados por parênteses, e ligações específicas podem ser explicitadas, como em *Mana6* ou *Galb3/4*, o que permite uma descrição sintática detalhada da estrutura. Após o processamento dos módulos e operadores, o sistema percorre o grafo da molécula para traçar um caminho contínuo que satisfaça todas as condições da expressão regular, retornando as correspondências encontradas. Por padrão, o comportamento é “guloso”, ou seja, o sistema tenta encontrar o maior padrão possível, embora a execução “preguiçosa”, que busca o menor padrão que satisfaça a condição, também seja suportada com o modificador “?”.

Entre as principais funções do módulo destacam-se:

- `get_match()`: Busca e extrai trechos da estrutura do glicano que correspondem a um padrão definido pelo usuário.
- `get_pvals_motifs()`: Avalia o enriquecimento estatístico de motifs dentro de diferentes contextos estruturais, permitindo identificar padrões recorrentes em conjuntos de glicanos.
- `get_differential_expression()`: Analisa a expressão diferencial de motifs glicosídicos entre grupos de amostras, facilitando a identificação de variações significativas em estudos comparativos.

Essas funções se integram diretamente às demais ferramentas do pacote *glycowork*, como a *GlycoDraw*, que permite visualizar graficamente os motifs identificados, e os módulos de anotação e destaque de redes de motifs. Essa integração favorece a análise automatizada e a exploração visual das estruturas reconhecidas pelas expressões regulares.

A seguir, apresenta-se um exemplo ilustrativo de inclusão de figura no contexto do módulo.

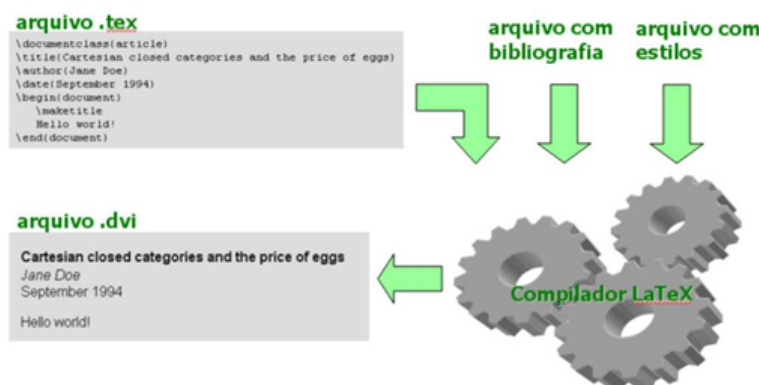


Figura 1: Teste de uma figura em formato .png.

## 2.2 Operação interna e papel das expressões regulares

O funcionamento interno do módulo envolve a tradução de padrões *RegEx* em operações de subgrafo sobre as estruturas moleculares representadas como grafos. Em termos gerais, o processo é dividido em etapas sucessivas que permitem converter descrições simbólicas de ligações e unidades monossacarídicas em estruturas compreensíveis computacionalmente.

Primeiramente, ocorre a segmentação do padrão, na qual a expressão *RegEx* fornecida é decomposta em módulos homogêneos, como unidades monossacarídicas ou ligações específicas. Por exemplo, o padrão "Neu5Ac $\alpha$ 2-3Gal $\beta$ 1-4(Fuc $\alpha$ 1-3)GlcNAc" é dividido em blocos como "Neu5Ac $\alpha$ 2-3", "Gal $\beta$ 1-4" e "Fuc $\alpha$ 1-3", representando diferentes segmentos estruturais que serão posteriormente analisados. Em seguida, ocorre a interpretação e expansão dos modificadores e quantificadores. Operadores como +, \*, ?, intervalos {m,n} e quantificadores preguiçosos (+?) são suportados, permitindo definir repetições, ocorrências opcionais ou faixas de aparecimento dos motivos. Essa flexibilidade amplia a capacidade do módulo de reconhecer variações estruturais dentro de uma mesma família de glicanos. Posteriormente, aplica-se o processo de *subgraph isomorphism*, em que cada segmento é buscado dentro do grafo que representa a estrutura glicosídica. Essa etapa utiliza algoritmos de isomorfismo de subgrafos para identificar correspondências exatas, mesmo em representações complexas que envolvem ramificações e múltiplas conexões entre unidades.

Outro componente essencial é a implementação das operações de *lookahead* e *look-behind*, herdadas das expressões regulares textuais. Elas permitem incluir ou excluir determinados contextos do padrão de busca sem incorporá-los ao resultado final, sendo cruciais para capturar motivos cuja ocorrência depende de um contexto estrutural específico.

Por fim, ocorre a construção do caminho de correspondência, na qual um algoritmo

iterativo reconstrói o trajeto contínuo dentro do grafo, unindo as correspondências parciais e validando os requisitos definidos pela expressão completa. Essa abordagem possibilita representar ligações específicas, como  $\alpha$ 1-3 ou  $\beta$ 1-6, bem como ambiguidade estrutural e ramificações expressas por meio de parênteses. Dessa forma, a notação *RegEx* é adaptada à topologia molecular, permitindo uma modelagem altamente expressiva e precisa das estruturas glicosídicas.

## 2.3 Vantagens da RegEx no contexto do glycowork

Como comentado, a inovação trazida pelo Glycan RegEx ao framework glycowork consiste na união do método tradicional de busca por motifs com isomorfismo de subgrafo e o uso de um sistema de expressões regulares específico para glicanos.

Sem o uso das expressões regulares, há a necessidade de uma base estática de motifs já conhecidos - como pode ser vista no arquivo `common_names.json` do próprio framework -, o uso de muitas combinações de padrões para detectar motifs complexos e específicos do glicano e a dificuldade de capturar contextos maiores, como um motif presente apenas em um ramo do glicano. Todos esses fatores culminam em um uso bastante rígido e, por vezes, difícil de representar de forma computacional.

Com a introdução do sistema Glycan RegEx é possível escrever padrões de buscas de motifs em um formato mais declarativo e eficaz, com uso de operadores clássicos que permitem negação, repetição e demais operações. Dessa forma, o algoritmo de busca de motifs tem a capacidade de gerar diversos subgrafos intermediários (ao contrário do mecanismo anterior), encontrando a combinação ideal de motifs de forma refinada e mais simples para o programador.

Esta camada adicional que o sistema traz pode gerar um *overhead* em estruturas mais simples do glicano quando comparado com a busca tradicional do glycowork por motifs. Entretanto, quando aplicado em glicanos mais complexos ou maiores, tal mecanismo se sobressai tanto em sua eficácia, podendo ser mais simples escrever uma única expressão regular que atenda aos critérios, quanto em sua eficiência.

## 3 Funções chave e eventos de código

Para ilustrar o funcionamento do framework glycowork para expressões regulares de glicanos, exploraremos algumas de suas funções chave e apresentaremos exemplos de código.

### 3.1 preprocess\_pattern

A função *preprocess\_pattern* é responsável por transformar uma expressão regular de glicano em uma lista de "pedaços" ou componentes. Isso é crucial porque as expressões de glicanos podem conter modificadores, quantificadores e estruturas condicionais que precisam ser interpretadas corretamente antes da correspondência de padrões.

```
def preprocess_pattern(pattern: str) -> List[str]:
    """Transform glyco-regular expression into chunks"""
```

Figura 2: Implementação da função process\_pattern

#### 3.1.1 Exemplo de uso

Considere a expressão regular de glicano  $ex-HexNAc-([Hex|Fuc])\{1,2\}-HexNAc$ . Esta expressão busca um padrão que começa com *Hex*, seguido por *HexNAc*, depois um ou dois *Hex* ou *Fuc*, e termina com *HexNAc*.

```

import re

def specify_linkages(pattern_component: str):
    if re.search(r"[d|\?]\(|\d$", pattern_component):
        pattern = re.compile(r"([ab\?])(\d+/\d+|\d|\?)\(\?1-\?\)")
        def replacer(match):
            letter, number = match.groups()
            return f'({letter}{1}-{number})'
        pattern_component = pattern.sub(replacer, pattern_component)
    return re.sub(r'(5Ac|5Gc|Kdn|Sia)\(a1', r'\1(a2', pattern_component)

# Exemplos de entrada
component1 = "Mana6"
component2 = "Galb3/4"

# Aplicação da função
specified_linkage1 = specify_linkages(component1)
specified_linkage2 = specify_linkages(component2)

print(f"Componente original 1: {component1} -> Ligação especificada: {specified_linkage1}")
print(f"Componente original 2: {component2} -> Ligação especificada: {specified_linkage2}")

```

Figura 3: Aplicação da função `process_pattern`

### 3.1.2 Comportamento de entrada

Para a entrada  $Hex-HexNAc-([Hex|Fuc])\{1,2\}-HexNAc$ , a função `preprocess_pattern` irá:

1. Substituir qualquer `.` por Monosaccharide;
2. Dividir a string em componentes com base em um padrão de expressão regular que identifica modificadores e quantificadores  $([Hex|Fuc], 1, 2)$ ;
3. Remover strings vazias e espaços em branco, além de hífen ( - ) das extremidades dos componentes.

O resultado esperado será uma lista de strings, onde cada string representa um "peçaço" da expressão regular de glicano, pronto para ser processado por outras funções do framework.

## 3.2 `specify_linkages`

A função `specify_linkages` converte notações abreviadas de ligações em sua forma completa. Isso é essencial para padronizar as representações de glicanos e garantir que as operações de grafo possam interpretar corretamente as ligações entre os monossacarídeos.



```
def specify_linkages(pattern_component: str) -> str:
    """Convert expression linkages from shorthand to full notation, such as
    Mana6 to Man(a1-6) or Galb3/4 to Gal(b1-3/4)"""
```

Figura 4: Implementação da função `specify_linkages`

### 3.2.1 Exemplo de uso

Considere um componente de padrão como *Mana6* ou *Galb3/4*.

```
import re

def specify_linkages(pattern_component: str):
    if re.search(r"[\d|\?]\([\d$]", pattern_component):
        pattern = re.compile(r"([ab\?])(\d+/\d+|\d|\?)(\?1-\?)"")
        def replacer(match):
            letter, number = match.groups()
            return f'({letter}{1}-{number})'
        pattern_component = pattern.sub(replacer, pattern_component)
    return re.sub(r'(5Ac|5Gc|Kdn|Sia)\(a1', r'\1(a2', pattern_component)

# Exemplos de entrada
component1 = "Mana6"
component2 = "Galb3/4"

# Aplicação da função
specified_linkage1 = specify_linkages(component1)
specified_linkage2 = specify_linkages(component2)

print(f"Componente original 1: {component1} -> Ligação especificada: {specified_linkage1}")
print(f"Componente original 2: {component2} -> Ligação especificada: {specified_linkage2}")
```

Figura 5: Aplicação da função `specify_linkages`

### 3.2.2 Comportamento de entrada

Para *Mana6*, a função identificaria a notação abreviada e a converteria para *Man(a1–6)*. Similarmente, *Galb3/4* seria convertida para *Gal(b1 – 3/4)*. Esta padronização é vital para a consistência na representação das ligações glicosídicas.

## 3.3 get\_match

A função `getmatch` é a principal interface para encontrar correspondências de uma expressão regular de glicano em uma estrutura de glicano. Ela utiliza as funções auxiliares como `preprocesspattern` e `specify_linkages` internamente para realizar a correspondência de padrões baseada em grafos.

```
def get_match(pattern: Union[str, List[str]],
              glycan: Union[str, nx.DiGraph],
              return_matches: bool = True
              ) -> Union[bool, List[str]]:
    """Find matches for glyco-regular expression in glycan"""
```

Figura 6: Implementação da função `get_match`

### 3.3.1 Exemplo de uso

Para um exemplo prático de `get_match`, precisaríamos de um ambiente `glycowork` completo, incluindo a capacidade de converter strings de glicanos em objetos `networkx.DiGraph`. No entanto, podemos ilustrar o conceito com um exemplo simplificado de como a função seria invocada e o tipo de resultado esperado.

```
# Este é um exemplo conceitual e requer o pacote glycowork instalado e
# configurado
# from glycowork.motif.regex import get_match
# from glycowork.glycan_data.loader import glycan_to_nxGraph

# Exemplo de expressão regular de glicano
pattern_to_find = "Hex-HexNAc"

# Exemplo de estrutura de glicano (representação simplificada para ilustração)
glycan structure = "GlcNAc(b1-2)Man(a1-3)[GlcNAc(b1-2)Man(a1-6)]Man(b1-
4)GlcNAc(b1-4)GlcNAc"

# Converter a string do glicano para um grafo (passo interno do glycowork)
# ggraph = glycan_to_nxGraph(glycan_structure)

# Aplicação conceitual da função
# matches = get_match(pattern_to_find, glycan_structure, return_matches=True)

# print(f"Padrão a ser encontrado: {pattern_to_find}")
# print(f"Estrutura do glicano: {glycan_structure}")
# print(f"Correspondências encontradas: {matches}")
```

Figura 7: Aplicação da função `specify_linkages`

### 3.3.2 Comportamento de entrada

Quando `get_match` é chamada com um `pattern` (expressão regular de glicano) e um `glycan` (estrutura de glicano), ela realiza os seguintes passos:

1. Pré-processamento do Padrão: Utiliza `preprocess_pattern` para quebrar a expressão regular em componentes;
2. Conversão do Glicano: Se o glicano for fornecido como string, ele é convertido para um objeto `networkx.DiGraph`, que é a representação interna usada para operações de grafo;

3. Correspondência de Padrões Baseada em Grafo: A função então executa um algoritmo de correspondência de subgrafo para encontrar todas as ocorrências do padrão dentro do grafo do glicano;
4. Formatação dos Resultados: Se *return\_matches* for True, a função retorna uma lista de strings, onde cada string representa uma subestrutura de glicano que corresponde ao padrão. Caso contrário, retorna um booleano indicando se alguma correspondência foi encontrada.

Este processo permite que o framework glycowork identifique padrões complexos em estruturas de glicanos, superando as limitações das expressões regulares textuais tradicionais ao considerar a topologia e as ligações químicas dos glicanos.

## 4 Considerações finais

As considerações finais formam a parte final (fechamento) do texto, sendo dito de forma resumida (1) o que foi desenvolvido no presente trabalho e quais os resultados do mesmo, (2) o que se pôde concluir após o desenvolvimento bem como as principais contribuições do trabalho, e (3) perspectivas para o desenvolvimento de trabalhos futuros, como listado nos exemplos de seção abaixo. O texto referente às considerações finais do autor deve salientar a extensão e os resultados da contribuição do trabalho e os argumentos utilizados estar baseados em dados comprovados e fundamentados nos resultados e na discussão do texto, contendo deduções lógicas correspondentes aos objetivos do trabalho, propostos inicialmente.

### 4.1 Principais contribuições

Texto.

### 4.2 Limitações

Texto.

### 4.3 Trabalhos futuros

Texto.

# Referências

- AGRA, A. Projeto de Diplomação, *Implementação de uma Proposta para Atualização de Bancos de Dados através de Visões*. Porto Alegre, RS, Brasil: [s.n.], jul. 2004.
- JASZKIEWICZ, A. *Genetic local search for multiple objective combinatorial optimization*. [S.l.], 1998.
- KNOWLES, J.; CORNE, D. Metaheuristics for multiobjective optimisation. In: \_\_\_\_\_. [S.l.]: Springer, 2004. (Lecture Notes in Economics and Mathematical Systems, v. 535), cap. Bounded Pareto Archiving: Theory and Practice, p. 39–64.
- KNOWLES, J. D.; CORNE, D. W.; FLEISCHER, M. Bounded archiving using the Lebesgue measure. In: *Proceedings of the IEEE Congress on Evolutionary Computation*. [S.l.]: IEEE Press, 2003. p. 2490–2497.
- LÓPEZ-IBÁÑEZ, M.; PAQUETE, L.; STÜTZLE, T. Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms*, v. 5, n. 1, p. 111–137, 2006.
- PAQUETE, L. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization Problems: Methods and Analysis*. Tese (Doutorado) — Technische Universität Darmstadt, 2005.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database system concepts*. 4th. ed. Boston: McGraw-Hill, 2002.
- TURAU, V. *DB2XML 1.4: Transforming relational databases into XML documents*. out. 2001. Out., 2001. Disponível em: <<http://www.informatik.fh-wiesbaden.de/~turau/DB2XML/index.html>>. Acesso em Abril 9, 2004.