# 2 REVISÃO BIBLIOGRÁFICA

## 2.1 Conceito de Revisão Bibliográfica

Revisão bibliográfica é um estudo e avaliação da literatura relacionado a um assunto específico (AVEYARD, 2014). Ou seja, é um estudo que apresenta o conhecimento atual existente para um tópico em particular. No termo em inglês existem várias formas de se referir à revisão bibliográfica: *systematic review*, *meta-analysis*, *rapid review*, *literature review*, *narrative review*, *research synthesis* e *structured review* (O'MALLEY, 2005).

Segundo Hart (1998) existem pelo menos 12 propósitos para uma revisão bibliográfica:

1. Distinguir o que já foi feito do que precisa ser feito.

2. Descobrir variáveis relevantes para o tópico.

3. Sintetizar e obter uma nova perspectiva.

4. O servidor de Alice recebe o pacote em seu endereço IP.

5. Identificar relações entre ideias e prática.

6. Estabelecer o contexto para o tópico ou problema de pesquisa.

7. Racionalizar a significância teórica ou prática do problema.

8. Melhorando e obtendo vocabulários comuns ao assunto.

9. Entender as origens e a estrutura do assunto.

10. Relacionar ideias e teoria com problemas e questões.

11. Identificar as principais metodologias e ferramentas de coleta de dados que têm sido usados.

12. Colocar a pesquisa num contexto histórico para mostrar familiaridade com o estado da arte da pesquisa.

Existem dois principais tipos de revisão bibliográfica: a intervencionista e a escolástica (HART, 1998). A intervencionista tem o propósito de usar todos as evidências disponíveis e confiáveis para tomar uma decisão. A escolástica tem por objetivo examinar os argumentos, procurar por contradições, desafiar proposições existentes e fazer inferências.

Segundo Aveyard (2014), uma revisão bibliográfica de qualidade, mesmo sendo apenas uma seção de um artigo maior, ela por si só é um trabalho de pesquisa completo e geralmente contém as seguintes estruturas:

- Problema de pesquisa bibliográfica ou contextualização
- Métodos de pesquisa usados
- Apresentação dos resultados
- Discussão dos resultados

Lingard (2018) afirma que a seção de revisão bibliográfica deve mostrar o que ainda não é conhecido dentro daquele tópico (o que Lingard chama de lacuna, ou gap) permitindo ao leitor entender porque tal pesquisa é necessária. Essa estratégia faz com que a revisão bibliográfica se torne mais um argumento convincente do que apenas uma lista de fatos.

Lingard (2018) identificou os quatro tipos mais comuns de *gap*:

- Um simples déficit no conhecimento — "*ninguém nunca analisou a relação entre habilidade no xadrez e desempenho em matemática*"
- Um déficit no conhecimento acadêmico, muitas vezes por causa de vieses filosóficos ou metodológicos — "o*s pesquisadores interpretaram x numa perspectiva cognitiva, mas ignoraram a perspectiva humanista" ou "até hoje nós analisamos a frequência de erros médicos cometidos por residentes, mas não exploramos as experiências subjetivas deles em tais erros*"
- Uma controvérsia — "*pesquisadores discordam na definição de profissionalismo dentro da medicina...*"
- Uma suposição comum, mas não comprovada — "*é muito comum na literatura a afirmação de que o cérebro possui 100 bilhões de neurônios, mas existem artigos publicados na literatura que mediu e confirmou esse número?*"

## 2.2 Web Scrapping

Web Scraping é a prática de coletar dados da Web que não seja acessando uma API. Isso é normalmente feito criando um programa que faz requisição para um servidor web, recebe os dados (na maioria das vezes na forma de documentos HTML) e extrai as informações necessárias. (MITCHELL, 2018).

Os dados na Internet são transferidos num formato bastante formal e estruturado, fácil de ser lido por computadores. Mas nas páginas web estes dados são exibidos numa forma desestruturada, em forma de textos que são fáceis de ler pelo ser humano, porém que são difíceis de serem processados por algoritmos. A prática de

Web Scraping permite que estes dados desestruturados sejam extraídos em dados estruturados (BOEING et al, 2016), daí a importância das técnicas de Web Scraping.

Mitchell (2018) explica de forma didática e simples o funcionamento básico da Internet, através de um exemplo:

Alice possui um servidor web. Bob usa um computador desktop que está tentando se conectar ao servidor de Alice. Quando uma máquina quer falar com outra máquina, algo como a seguinte troca ocorre:

1. O computador de Bob envia fluxo de dados de 1 e 0 bits, representados por tensões elétricas alta e baixas no fio de transmissão. Esses bits formam algumas informações, contendo um cabeçalho (header) e um corpo (body). O cabeçalho contém um destino imediato do endereço MAC do seu roteador local, com um destino final do endereço IP de Alice. O corpo contém sua solicitação para a aplicação do servidor web de Alice.

2. O roteador local de Bob recebe todos esses 1s e 0s e os interpreta como um pacote, a partir do endereço MAC do próprio Bob, destinado ao endereço IP de Alice. Seu roteador marca seu próprio endereço IP no pacote como o endereço IP de origem e o envia pela Internet.

3. O pacote de Bob percorre vários servidores intermediários, que direcionam seu pacote para o caminho correto, até o servidor de Alice.

4. O servidor de Alice recebe o pacote em seu endereço IP.

5. O servidor de Alice lê a porta de destino do pacote no cabeçalho e passa para a aplicação apropriada - a aplicação do servidor web. (A porta de destino do pacote é quase sempre a porta 80 para aplicativos da Web; a porta pode ser considerado como um número de apartamento para dados do pacote, enquanto o endereço IP é como o endereço da rua.)

6. O aplicativo do servidor da web recebe um fluxo de dados do processador do servidor. Esses dados dizem algo como o seguinte: - Esta é uma solicitação GET. - O seguinte arquivo é solicitado: index.html.

7. O servidor web localiza o arquivo HTML correto, agrupa-o em um novo pacote para enviar a Bob e o envia para o roteador local, para ser transportado de volta à máquina de Bob, por meio do mesmo processo.

Nos casos que é necessário coletar dados de múltiplas páginas, ou até múltiplos sites, será necessário o uso de um Web Crawler que são programas que “rasteja” (crawl, em inglês) ao longo da web (MITCHELL, 2018). Web Crawlers também são chamados de robôs, spiders, worms, walkers e wanderers (HEYDON et al, 1999).

Todos os motores de busca populares usam web crawlers que precisam escalar para grandes proporções da web (HEYDON et al, 1999). Porém com o avanço da Internet simples web crawlers centralizados se tornaram cada vez menos satisfatório e foi notado que é necessário paralelizar os processos de crawling usando múltiplos

servidores numa arquitetura distribuída para um web crawling eficiente (BOLDI et al, 2003).

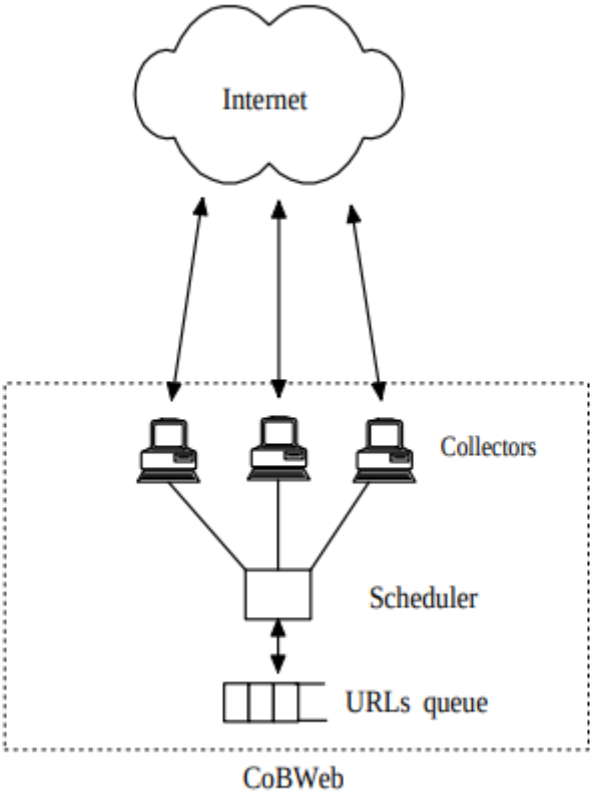
A Google é um exemplo de sistema que usa um web crawling distribuído com múltiplas máquinas para o processo de crawling. O crawling consiste em cinco componentes funcionais rodando em diferentes processos. O URL server process lê as URL de um arquivo e os redireciona para múltiplos processos crawler. Cada processo crawler roda numa diferente máquina, é de thread única, e usa I/O assíncrono para buscar dados de mais de 300 servidores web em paralelo. Os crawlers transmite as páginas baixadas para um único processo indexador que extrai links do HTML e os salva num arquivo diferente. O URL resolver process lê o arquivo com os links, e faz um processo chamado derelativization das URLs e salva as URL absolutas no arquivo que é lido pelo URL server. Tipicamente, três a quatro máquinas crawlers são usados de forma que o sistema inteiro precise de quatro a oito máquinas (HEYDON, 1999).

A UbiCrawler é um web crawler distribuído, implementado com a linguagem de programação Java, cuja arquitetura e funcionamento foi publicado num artigo (BOLDI et al, 2003). Os autores afirmam que os principais atributos do UbiCrawlers são:

- Independente de plataforma
- Distribuição total de cada task (não tem um ponto único de falha e nem coordenação centralizada)
- Tolerância a falhas: falhas permanentes e transientes são lidados de forma eficiente
- Escalabilidade

O CoBWeb é um web crawler para coletar dados especificamente da web brasileira (que tenha .br no domínio). As operações do CobWeb é orientado para eficiência, robustez e parcimônia no uso de recursos compartilhados (SILVA et al, 1999).

Figura 1 – Arquitetura do CoBWeb




Fonte – SILVA et al (1999)

Uma busca no site github.com pelo termo “web scraping” mostra que a linguagem de programação Python é a mais utilizada para tal tarefa (busca realizada em 19 de abril de 2019 no site **https://github.com**). O Python possui um módulo para buscar dados da web de forma fácil e prática, chamado de urllib.request (docs.python.org, acessado em 19 de abril de 2019).

Figura 2 - Exemplo de código para buscar dados a partir de uma URL

```
import urllib
params = urllib.urlencode({'spam': 1, 'eggs': 2, 'bacon': 0})
f = urllib.urlopen("http://www.musi-cal.com/cgi-bin/query?%s" % params)
print f.read()
```

Fonte - docs.python.org (acessado em 19 de abril de 2019)

A biblioteca BeautifulSoup é usada para fazer o parsing de documentos HTML de forma fácil na forma de objetos Python e o seu nome é em homenagem a um poema recitado por um personagem da estória Alice nos Países da Maravilha (MITCHELL, 2018).

Figura 3 - Exemplo de código BeautifulSoup em que a partir do HTML retornado é extraído um elemento H1

```
from urllib.request import urlopen
from bs4 import BeautifulSoup

html = urlopen('http://www.pythonscraping.com/pages/page1.html')
bs = BeautifulSoup(html.read(), 'html.parser')
print(bs.h1)
```

Fonte - Mitchell (2018).

Outra ferramenta existente é o framework Scrapy. No site oficial do Scrapy a ferramenta é definida como “um framework de código aberto e colaborativo para extrair dados que você precisa dos sites web de forma rápida, simples e extensível” (**https://scrapy.org**, acessado em 19 de abril de 2019).

Figura 4 - Exemplo de programa Scrapy que percorre três sites do Wikipédia exibindo o texto do primeiro elemento H1 que encontrar em cada página

```
import scrapy

class ArticleSpider(scrapy.Spider):
    name='article'

    def start_requests(self):
        urls = [
            'http://en.wikipedia.org/wiki/Python_'
            '%28programming_language%29',
            'https://en.wikipedia.org/wiki/Functional_programming',
            'https://en.wikipedia.org/wiki/Monty_Python']
        return [scrapy.Request(url=url, callback=self.parse)
            for url in urls]

    def parse(self, response):
        url = response.url
        title = response.css('h1::text').extract_first()
        print('URL is: {}'.format(url))
        print('Title is: {}'.format(title))
```

Fonte - Mitchell (2018).

O PySocks é um módulo do Python que consegue rotear o tráfego HTTP para servidores de proxy e funciona muito bem com o navegador Tor permitindo a atividade de web scraping de forma anônima.

Figura 5 - Exemplo de código em PySocks que busca os dados de uma URL passando pelo proxy definido na porta 9150, que é a porta padrão onde roda o serviço Tor.

```
import socks
import socket
from urllib.request import urlopen

socks.set_default_proxy(socks.SOCKS5, "localhost", 9150)
socket.socket = socks.socksocket
print(urlopen('http://icanhazip.com').read())
```

Fonte - Mitchell (2018).

Alguns exemplos de aplicações do Web Scraping:

- Boeing e Wadell (2016) usaram técnicas de Web Scraping para coletar dados do site de anúncios Craiglist para fazer uma análise sobre os aluguéis de imóveis nos Estados Unidos.
- Landers et al (2016) usaram programas desenvolvidos em Python para demonstrar como o web scraping pode ser usado para coletar dados do big data para uso em pesquisas de psicologia.
- Haddaway (2015) usou web scraping para pesquisar por literatura cinzenta, que são publicações não convencionais difíceis de encontrar em canais tradicionais de distribuição, com controle biográfico ineficaz pois não recebem numeração internacional, e não são depósito de objetos legais em muitos países sendo normalmente não incluídas em bibliografias e catálogos (BOTELHO, 2015).
- Arora et al (2013) usou web scraping para coletar dados de sites de divulgação de empresas pequenas e médias de grafeno para analisar como as especializações dessas empresas impacta o mercado geral de grafeno.

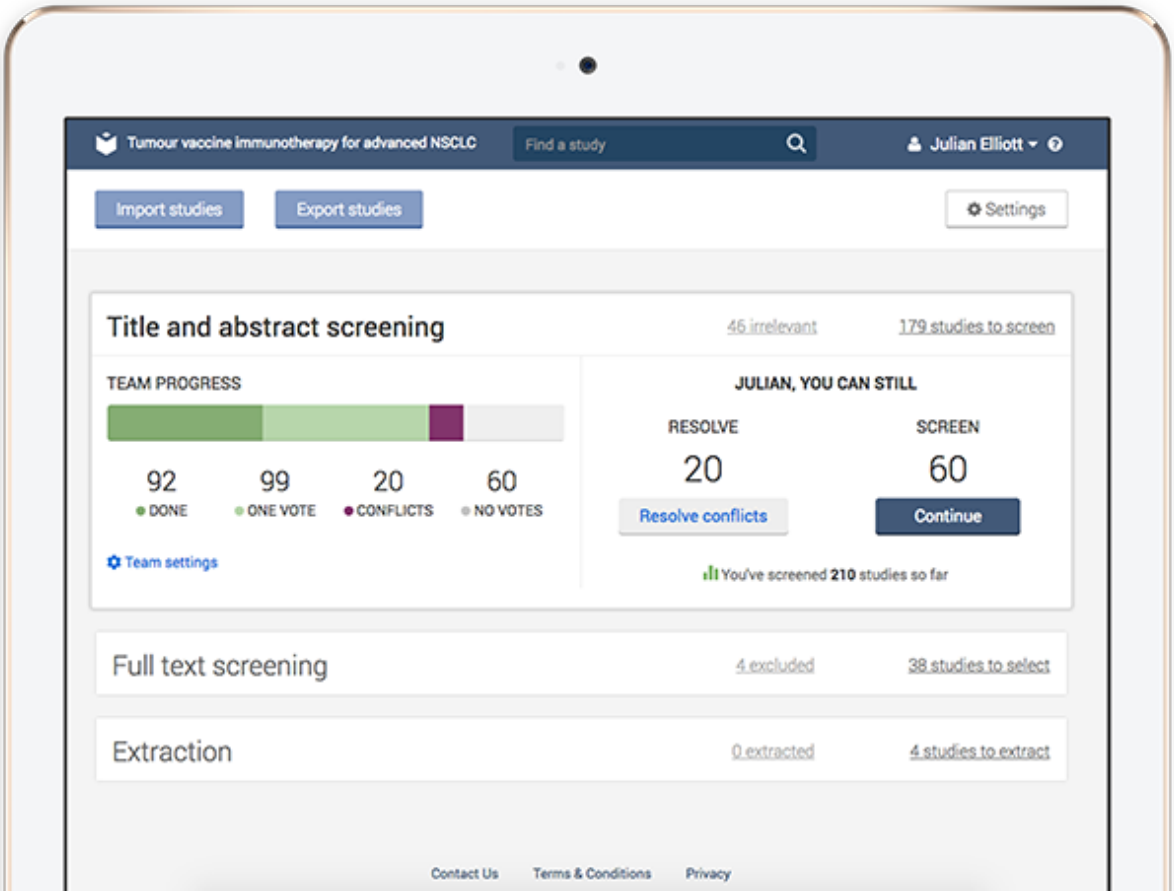
## 2.3 Estado da Arte

O **covidence** é um gerenciador de revisão sistemática, na forma de um sistema web, da empresa Cochrane. Existe o plano gratuito de demonstração com limitações e o plano pago mais barato é um de U$ 240,00. Segundo o site oficial do produto (**https://www.covidence.org**, acessado em 1 de maio de 2019) possui as seguintes funcionalidades principais:

- Importa citações de outros sistemas gerenciador de referências como o Mendeley, EndNote, Zotero e Refworks ou qualquer ferramenta que suporte o formato CSV, RIS ou PubMedXML.
- Fazer a “triagem” de artigos e resumos usando destacador de palavras-chaves.

- Anexar comentários em textos diretamente no PDF.
- Transferir PDFs armazenados num sistema gerenciador de referência para o covidence.
- Ir extraindo dados de vários artigos diferentes conforme for lendo para analisar em conjunto depois.
- Exportar o trabalho feito no covidence para variados formatos de documentos.

Figura 6. Exemplo de tela do covidence




Fonte - **https://www.covidence.org/home**

O **DistillerSR** é um sistema web que gerencia, rastreia e agiliza o processo de análise do artigo, extração de dados e reportar processos utilizados no sua revisão sistemática. O plano mais barato é o de U$ 15,00 mensal por usuário. Algumas das funcionalidades do DistillerSR são (**https://www.evidencepartners.com**, acessado em 1 de maio de 2019):

- Permite múltiplos usuários da equipe manipular o mesmo trabalho ao mesmo tempo evitando duplicar trabalho.

- Permite importar artigos das principais ferramentas de gerenciador de referências, banco de dados e até planilhas com acesso direto ao PubMed para busca e referência de artigos.
- Rastreia cada referência, cada mudança em cada célula dos dados ao longo do desenvolvimento do trabalho de revisão sistemática facilitando a auditoria de quem da equipe alterou o que.

## REFERÊNCIAS BIBLIOGRÁFICAS


AVEYARD, Helen. **Doing a Literature Review in Health and Social Care**. 3. ed. Maidenhead: McGraw-Hill, 2014.

ARKSEY, Hilary; O'MALLEY, Lisa. **Scoping studies: towards a methodological framework**. International Journal of Social Research Methodology, 2005

HART, Chris. **Doing a literature review**. Thousand Oaks: Sage Publications Ltd, 2018.

LINGARD, L. **Writing an effective literature review**. Springer, fev 2018. Disponível em: <https://doi.org/10.1007/s40037-017-0401-x>. Acesso em 19 abr. 2019.

MITCHELL, Ryan. **Web Scraping with Python**. Sebastopol: O'Reilly, 2018.

BOEING, Geoff, and PAUL Waddell. **New Insights into Rental Housing Markets across the United States: Web Scraping and Analyzing Craigslist Rental Listings**. Journal of Planning Education and Research, 2017, v. 37, no. 4, pp. 457–476.

HEYDON, A. & NAJORK, M. **Mercator: A scalable, extensible Web crawler**. World Wide Web, 1999, v. 2, n. 4, pp 219-229.

BOLDI P et al. **Ubicrawler: A scalable fully distributed web crawler**. The 8th Australian World Wide Web Conference, 2002.

SILVA et al. **CoBWeb-a crawler for the Brazilian Web**. 6th International Symposium on String Processing and Information Retrieval, 1999.

ARORA et al. E**ntry strategies in an emerging technology: a pilot web-based study of graphene firms**. Scientometrics, 2013, v. 95, n. 3, pp 1189-1207.

LANDERS, R. N et al. **A primer on theory-driven web scraping: Automatic extraction of big data from the Internet for use in psychological research**. Psychological Methods, 2016, v. 21, n. 4, pp 475-492.

HADDAWAY, Neal. **The Use of Web-scraping Software in Searching for Grey Literature**. TDJ, 2015, v. 11, n. 3, pp 186-190.