UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THIAGO ALEXANDRE NAKAO FRANÇA

PROPOSTA DE MECANISMOS PARA O PROBLEMA COMBINATÓRIO DE COMPRAS EM LOTE DE DIFERENTES E-COMMERCES - UM ESTUDO DE CASO DO DESENVOLVIMENTO DE SHOPBOTS PARA MAGIC: THE GATHERING

MONOGRAFIA

CAMPO MOURÃO
2019

THIAGO ALEXANDRE NAKAO FRANÇA

PROPOSTA DE MECANISMOS PARA O PROBLEMA COMBINATÓRIO DE COMPRAS EM LOTE DE DIFERENTES E-COMMERCES - UM ESTUDO DE CASO DO DESENVOLVIMENTO DE SHOPBOTS PARA MAGIC: THE GATHERING

Proposta de Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr Rodrigo Campiolo

Coorientador: Profª. Drª Lilian Caroline Xavier Candido

CAMPO MOURÃO

2019

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **21:20** do dia **25 de novembro de 2019** foi realizada na sala **E100** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Thiago Alexandre Nakao Franca** com o título **Proposta de mecanismos para o problema combinatório de compras em lote de diferentes e-commerces - Um estudo de caso do desenvolvimento de shopbots para Magic: The Gathering**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Rodrigo Campiolo** (orientador(a)), **Profa. Dra. Lilian Caroline Xavier Candido, Prof. Dr. Luiz Arthur Feitosa dos Santos e Prof. Dr. Lucio Geronimo Valentin**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso 2 e atribuiu, em consenso, a nota _____ (_____). Esse resultado foi comunicado ao (à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao (à) acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

_____

_____

Campo Mourão, **25 de novembro de 2019**

_____	_____
Profa. Dra. Lilian Caroline Xavier Candido	**Prof. Dr. Luiz Arthur Feitosa dos Santos**
Membro 1	Membro 2
_____	_____
Prof. Dr. Lucio Geronimo Valentin	**Prof. Dr. Rodrigo Campiolo**
Membro 3	Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Resumo

Nakao, Thiago. Proposta de mecanismos para o problema combinatório de compras em lote de diferentes e-commerces - Um estudo de caso do desenvolvimento de shopbots para Magic: The Gathering. 2019. 55. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2019.

No comércio eletrônico é comum que serviços de *shopbots* cataloguem preços de diversos fornecedores. No entanto, são raros os que devolvem a configuração de compra ótima (menor preço) para diversos produtos solicitados pelo usuário. No caso do jogo de cartas *Magic:The Gathering*, existem diversos fornecedores de cartas e usuários interessados em adquirir um conjunto específico de cartas, tornando assim interessante uma aplicação que encontre configurações de compra de custo mínimo para diversos produtos. Neste trabalho foram propostos e avaliados mecanismos que efetuam a pesquisa de uma lista de produtos em um conjunto de *e-commerces*, visando encontrar a configuração de compra em que o preço é mínimo. Para tal, um estudo de caso foi realizado com o jogo de cartas *Magic: The Gathering*. A coleta e o armazenamento dos preços foram realizados por processos concorrentes. Os preços foram armazenados em uma *cache* para a redução do tempo de resposta da aplicação. A configuração de compra de custo mínimo foi implementada usando algoritmos de otimização combinatória. Foram implementadas três soluções utilizando programação linear inteira. Como resultados, os mecanismos desenvolvidos são capazes de encontrar soluções ótimas com garantias de tempo de resposta. No estudo de caso, cenários com 100 produtos distintos e 118 fornecedores convergiram em cerca de 25 segundos.

Palavras-chave: *web crawlers*, programação linear inteira, *caching*.

Abstract

Nakao, Thiago. Mechanisms for combinatorial optimization problem of batch purchases in many e-commerces - A case study of shopbots development for Magic: the Gathering. 2019. 55. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2019.

Shopbots services are common in Internet to compare prices from distinct providers. However, few of them find an optimal purchase configuration to a product list informed by users. In the card game Magic: The Gathering, there are many card retailers and users interested in acquiring a specific set of cards, so a tool to fetch purchase configurations with minimal cost would be interesting. In this work, it is proposed and evaluated mechanisms to perform price comparison for a product list in an e-commerces group, and to find purchase configuration with minimal cost. A case study was conducted with the Magic: The Gathering to achieve proposed goals. Product prices were collected and stored in a database, and they were used by cache mechanisms to mitigate response time from application. Three integer linear programing solutions were designed and implemented to get purchase configuration with minimal cost. The results demonstrated that developed techniques were able to find optimal solutions with response time guarantees. Empirical tests in the case study with 100 distinct products and 118 providers converged in about 25 seconds.

Keywords: web crawlers, integer linear programming, caching.

Lista de figuras

Lista de tabelas

Siglas

API: *Application Programming Interface*
CSS: *Cascading Style Sheets*
CSV: *Comma-Separated Values*
GRAB: *Greedy Addition of Bundles*
HTML: *HyperText Markup Language*
HTTP: *Hypertext Transfer Protocol*
SGBD: Sistema de Gerenciamento de Base de Dados

Sumário

Capítulo 1

Introdução

A quantidade e diversidade dos comércios eletrônicos (*e-commerces*) na Internet é imensa, esse fato é evidenciado em um estudo realizado pela Ebit (2017). O estudo mostra que o faturamento dos *e-commerces* em 2017 no Brasil foi de 47,7 bilhões de reais, o que representa aumento de $7,5\%$ em relação a 2016, quando o setor registrou faturamento de $44,4$ bilhões de reais. Além disso, conforme novos negócios emergem a quantidade de *e-commerces* tende a aumentar cada vez mais. Em 2018, um novo estudo realizado pela Ebit (2018) constatou que o comércio eletrônico brasileiro manteve o crescimento e registrou o faturamento de 53,2 bilhões de reais, alta nominal de 12% comparado a 2017.

Considerando esse cenário, verifica-se o surgimento de serviços que catalogam os produtos de diversos *e-commerces*, de forma a facilitar a escolha de compra do consumidor. Tais serviços são comumente denominados buscadores de preços ou *shopbots*. Muitas aplicações desse gênero já existem, como os sítios do Buscapé (BUSCAPE, 2019), *Camel Camel Camel* (CAMELCAMELCAMEL, 2019), *Yahoo Shopping* (YAHOO, 2019) e o *Google Shopping* (GOOGLE, 2019a).

Essas aplicações indexam uma vasta quantidade de produtos, obtêm respostas rápidas de consultas e devolvem os preços classificados de diferentes formas para o usuário. No entanto, a maioria possibilita somente buscas contendo um único produto. Além disso, muitos buscadores de preços não exibem todos os *e-commerces* que atuam sobre um ramo específico.

Há categorias de produtos nas quais é comum que usuários tenham interesse em comprar vários itens, podendo até mesmo ser uma quantidade que ultrapasse dezenas de itens. É interessante para o usuário que compra na Internet gastar o menor valor possível, ainda que a compra ocorra em mais que uma loja. Neste contexto é que se enquadra o presente trabalho, o qual é detalhado a seguir.

1.1. Contexto

Existem *e-commerces* que atuam com a venda de produtos de diversas ramificações, por exemplo, negócios voltados para a comercialização de perfumes e cosméticos na Internet, como os sítios do Boticário (BOTICARIO, 2019), Época Cosméticos (COSMETICOS, 2019a) e Nikkey Cosméticos (COSMETICOS, 2019b). Da mesma forma, há a comercialização de uma vasta gama de produtos, desde acessórios esportivos, roupas, relógios, jogos, computadores e muito mais.

Dentre todas essas ramificações, *Magic: The Gathering* é uma ramificação em que cada compra corriqueiramente inclui uma grande lista de produtos distintos, podendo facilmente ultrapassar 60 itens por compra. *Magic: The Gathering* é um jogo de cartas físico e digital muito popular em diversos países. Para comprar cartas, os jogadores têm acesso a serviços de lojas independentes, que vendem cartas avulsas, baralhos fechados e outros tipos de produtos relacionados.

As cartas do jogo são comercializadas por centenas do lojas no mundo e, no Brasil, já são mais de 90 *e-commerces* que atuam nesse ramo. A compra de muitos produtos por vez nesse mercado é comum, pois o baralho de *Magic: The Gathering* tipicamente contém pelo menos 60 cartas (WONG, 2018). Além disso, o preço de cada carta varia entre as lojas que as comercializam, logo não é incomum que a diferença de preço de uma única carta ultrapasse o valor do frete para uma loja concorrente.

No Brasil, o sítio da Ligamagic (LIGAMAGIC, 2019) oferece um serviço que efetua buscas de preços das cartas de *Magic: The Gathering* em diversas lojas. Entretanto, as buscas por conjunto de cartas não devolvem a configuração de compra com custo mínimo.

Embora pareça um caso isolado, a busca pelo menor preço de um conjunto de produtos é interessante para diversos ramos, como em listas de compras de mercado, nas quais a compra de menor custo pode ocorrer em diversos mercados locais, bem como a compra de materiais e objetos para construção de edifícios, onde produtos como piso porcelanato, piso madeira e piso vinílico podem ser comprados em lojas de materiais de construção diferentes.

1.2. Problema de pesquisa

Em *shopbots* populares como Buscapé, *Google Shopping* e *Yahoo Shopping* são executadas consultas que contêm um único produto por vez. Além disso, é comum que existam *e-commerces* do ramo que não sejam exibidos por certos *shopbots*, por exemplo, a Ligamagic não exibe os portfólios de todos os *e-commerces* brasileiros de *Magic: The Gathering*.

Na Seção 1.1, foram introduzidas categorias de produtos nas quais a compra de diversos itens simultâneos é recorrente. No entanto, a execução dessa funcionalidade envolve dois problemas consideráveis, que são:

1. A coleta dos preços dos produtos para todas as lojas.
2. Descobrir em quais lojas os produtos devem ser comprados.

A coleta é um problema devido à grande quantidade de produtos e de lojas. O preço de cada carta é obtido por uma requisição *Hypertext Transfer Protocol* (HTTP), logo para cada *e-commerce* seria realizada uma requisição por produto, o que leva a conclusão que a quantidade de requisições HTTP seria elevada.

Mesmo com o acesso aos preços de cada carta em todas as lojas, indicar em qual loja o usuário deve comprar cada carta não é uma tarefa trivial, ainda que o critério de seleção para cada produto seja exclusivamente a escolha da loja que apresentar o menor preço para o produto. Mais detalhes sobre a coleta e busca da melhor configuração das compras são discutidas nas duas subseções seguintes.

1.2.1. A coleta dos preços

A coleta de preços é uma etapa fundamental para aplicações que comparam os custos de produtos em fornecedores distintos. Esse problema é intensificado quando não existem formas facilitadas de acesso ao portfólio dos *e-commerces*, como a disponibilização da *Application Programming Interface* (API) para o acesso de múltiplos preços já filtrados, ou o acesso direto à base de dados.

Considerando que a maioria das lojas que comercializam as cartas de *Magic: The Gathering* no Brasil não disponibilizam a API para acessar os preços das cartas, seria necessário o uso de requisições HTTP para cada carta a ser obtida. Isso ocorre porque o preço de cada carta é geralmente disponibilizado em uma página Web específica.

No Brasil, há mais de 90 *e-commerces* que vendem as cartas do jogo. Supondo que um jogador deseje comprar um baralho de 100 cartas, para uma única compra seriam geradas 9.000 requisições HTTP. Ainda que sejam usadas técnicas de concorrência e paralelismo, são muitas requisições a serem realizadas por um único usuário, ainda mais quando se espera uma resposta rápida.

1.2.2. A melhor configuração de compra

A determinação da configuração de compra com custo mínimo quando existem diferentes produtos e fornecedores não é uma tarefa adequada para um algoritmo de busca por força bruta (algoritmo que testa todas as configurações existentes). É possível que a compra de custo mínimo ocorra em mais do que um *e-commerce* caso a diferença do preço dos produtos supere a diferença do valor do frete. Isso implica em uma grande quantidade de iterações para calcular o custo da compra em cada possível configuração de compra.

Considere um conjunto de produtos indexados por $I = \{1, \ldots, n\}$ e um conjunto de *e-commerces* indexados por $J = \{1, \ldots, m\}$. Determinar a melhor configuração de compra

consiste em selecionar a loja para a compra de cada produto. Para tal, podem constar uma, duas, três até m lojas na configuração resultante. A quantidade $C_{m,j}$ de configurações de compra contendo j lojas é dada pela equação 1.1, na qual $C_{m,j}$ é o total de configurações em que são utilizados j *e-commerces* dentre os m disponíveis.

$$C_{m,j} = \frac{m!}{j! \cdot (m-j)!} \tag{1.1}$$

O total de configurações possíveis para m *e-commerces* é composto pela soma das quantidades das configurações com j *e-commerces*, para todo $j \in J$, conforme apresentado na equação 1.2, em que C é o total de configurações possíveis e $C_{m,j}$ é calculado conforme a equação 1.1.

$$C = \sum_{j=1}^{m} C_{m,j} = 2^{m-1} \tag{1.2}$$

Intuitivamente surge uma solução simples para o problema: calcular o custo da compra em cada uma das possíveis configurações de *e-commerces* e, então, selecionar a configuração cujo custo seja mínimo. Logo, considerando as 2^{m-1} configurações possíveis de lojas, para cada uma delas deve-se percorrer todos os n produtos em todas as lojas presentes naquela configuração. Nessa solução pode-se descrever a quantidade mínima de iterações para gerar todas configurações, com seu respectivo custo, na equação 1.3, em que Q é a quantidade de iterações necessárias.

$$Q = \sum_{j=1}^{m} C_{m,j} \cdot (j \cdot n) \tag{1.3}$$

Em outras palavras, em todas as configurações possíveis seria necessário percorrer a lista de produtos uma vez para cada *e-commerce* que compõe a configuração.

No problema que deseja-se resolver, existem pelo menos 90 *e-commerces* para buscar preços, também é comum que as listas de produtos tenham entre 60 e 100 produtos. Apenas para gerar as configurações dos *e-commerces* seriam necessárias mais que 2^{90-1} operações, uma quantidade muito alta e impraticável para aplicações reais.

Existe outra abordagem que intuitivamente parece resolver o problema, bastaria escolher as lojas que contêm os produtos de menor preço. No entanto, nem sempre essa medida encontraria o valor mínimo real, devido ao frete acrescentado para cada loja.

1.3. Objetivo

Neste trabalho, são propostos e avaliados mecanismos que efetuam a pesquisa de preços de uma lista de produtos em um conjunto de *e-commerces*, visando encontrar o subconjunto de *e-commerces* em que a lista de produtos tenha o custo mínimo. Para tal, é realizado

um estudo de caso nos *e-commerces* voltados à venda de cartas para o jogo *Magic: The Gathering*.

Como objetivos específicos, têm-se:

1. Propor um mecanismo para coletar os preços de produtos em vários *e-commerces*, visando a redução do tempo de resposta.
2. Comparar o desempenho de algoritmos que sejam capazes de encontrar o conjunto de *e-commerces* no qual o custo da compra seja minimizado.
3. Desenvolver um *shopbot* para os *e-commerces* de *Magic: The Gathering*.

1.4. Contribuições

As principais contribuições são:

1. Avaliação de diferentes algoritmos para resolução do problema combinatório de compras em lote de diferentes *e-commerces*.
2. Modelos propostos de programação linear inteira para cenários de compra de diversos produtos em diferentes fornecedores.
3. Software para buscar os *e-commerces* de *Magic: The Gathering* nos quais compras têm o custo monetário mínimo.

1.5. Estrutura

O Capítulo 2 contém conceitos fundamentais para a compreensão desta pesquisa, em especial, o jogo de cartas *Magic: The Gathering*, conceitos e técnicas para otimização de *Web crawlers* e algoritmos de otimização combinatória. O Capítulo 3 apresenta e discute os trabalhos relacionados. O capítulo 4 descreve os materiais e o método de pesquisa adotados para alcançar o objetivo deste trabalho. O Capítulo 5 mostra e analisa os resultados dos experimentos executados com o protótipo do *shopbot*. Por fim, o Capítulo 6 apresenta as considerações finais a respeito da pesquisa e os trabalhos futuros.

Capítulo 2

Conceitos

Este capítulo aborda os tópicos fundamentais para a compreensão deste trabalho: o jogo de cartas *Magic: The Gathering*, comércio eletrônico na Internet (*e-commerces*), buscadores de preços, *web crawlers*, programação linear inteira e programação dinâmica.

2.1. Magic: The Gathering

Magic: The Gathering é um jogo de cartas colecionável criado pelo matemático Richard Garfield, lançado em 1993 pela empresa *Wizards of the Coast*. Em 2015 o jogo tinha cerca de vinte milhões de jogadores e pelo menos 1 bilhão de cartas impressas (WIZARDS, 2019c).

De acordo com Wong (2018), cada partida de *Magic: The Gathering* representa uma batalha de magos que conjuram magias, usam artefatos e invocam criaturas usando suas respectivas cartas. Na maioria dos formatos, os jogadores começam com 20 pontos de vida, cada jogador que chegar a 0 pontos de vida perde o jogo, embora existam formas alternativas de obter a vitória.

Pela Wizards (2019a), existem diversos formatos de jogo, que são modos alternativos de jogar com as cartas, variando as regras, totais de pontos de vida, quantidade de jogadores e cartas permitidas.

Cada jogador utiliza um baralho que geralmente contém 60 cartas, que são os recursos para batalhar contra os seus oponentes. Embora a maioria dos formatos do jogo definam que os baralhos tenham pelo menos 60 cartas, existe um formato muito popular chamado *Commander* no qual a quantidade de cartas padrão é 100. Os jogadores quase sempre utilizam baralhos diferentes de seus oponentes em cada partida, a escolha das cartas que compõe cada baralho é um dos principais aspectos do jogo.

A Figura 2.1 apresenta a aparência convencional de uma carta de *Magic: The Gathering* e as suas principais informações: nome, custo, tipo, edição, habilidades, poder e

Figura 2.1. Uma carta de *Magic: The Gathering*

resistência.

As cartas são lançadas periodicamente em edições, sendo comum que uma nova coleção seja lançada a cada quatro meses. As coleções tradicionais costumam ter de 200 a 350 cartas, mas existem muitas coleções especiais com quantidades distintas. Dados das coleções lançadas do jogo são disponibilizados pela *Wizards of the Coast* (WIZARDS, 2019b).

Magic: The Gathering possui versões digitais, como o *Magic: The Gathering Online* e o *Magic Arena*. A *Wizards of the Coast* (WIZARDS, 2019c) promove campeonatos mundiais para diversos formatos do jogo físico e digital, com recompensas para os vencedores, o que acaba ajudando a promover o jogo.

2.2. Comércio eletrônico

De acordo com Turban et al. (2009), comércio eletrônico, do inglês *e-commerce*, é um termo usado para definir qualquer atividade de compra ou venda de produtos, contanto que estejam sendo comercializados em serviços na Internet.

No Brasil existem *e-commerces* de diversas ramificações, como de perfumes, cosméticos, tênis, joias, computadores, livros e muito mais. Como os *e-commerces* comuns atuam a distância, o uso do serviço dos Correios (CORREIOS, 2019) e de transportadoras são as formas mais comuns para o envio dos produtos.

Os e-commerces de *Magic: The Gathering* em sua maioria são de sítios de compras

de cartas e acessórios, no entanto vale ressaltar que existem certas lojas que trabalham com produtos de outras categorias, como histórias de quadrinhos e jogos de tabuleiro.

No Brasil, mais de 100 *e-commerces* comercializam as cartas do jogo, a vasta maioria deles contrata serviços de uma empresa chamada Ligamagic (LIGAMAGIC, 2019), que vende para os lojistas todo o serviço de comércio online, contemplando todas as cartas do jogo em suas respectivas edições, bem como questões de pagamento e inclusão no buscador de preços da Ligamagic, por uma taxa mensal.

2.3. *Shopbots*

Segundo Zhang e Jing (2011), *shopbots* ou buscadores de preço são ferramentas de busca que geralmente filtram e comparam produtos de diversos fornecedores. Usualmente a comparação utiliza métricas como preço, avaliações ou demais características específicas dos produtos e fornecedores.

Em geral, os buscadores de preços não vendem diretamente os produtos exibidos, apenas agregam listas de produtos de diversos fornecedores. Existem buscadores de preços que atuam sobre produtos de *e-commerces* brasileiros, como o Buscapé, *Google Shopping*, Ligamagic e o Zaply (ZAPLY, 2019).

A Ligamagic é o principal buscador de preços voltado para cartas de *Magic: The Gathering* no Brasil, tendo mais de 90 *e-commerces* indexados. O serviço permite que qualquer usuário pesquise por uma carta individual, tendo acesso a uma relação de lojas e seus preços.

No mês abril de 2019 a Ligamagic disponibilizou o serviço de busca por grupos de cartas, no entanto não há garantia de que a configuração de compra retornada pela aplicação seja a de custo mínimo.

2.4. Web Crawlers

De acordo com Liu e Menczer (2011), *Web crawlers*, também conhecidos como *spiders* ou robôs, são programas que fazem o *download* automático de páginas Web. Um *crawler* pode visitar muitos sítios para coletar informações que podem ser analisadas e armazenadas em uma localização central.

Muitos *shopbots* são *crawlers*, tais como os serviços do Buscapé (BUSCAPE, 2019), Google Shopping (GOOGLE, 2019a) e Trivago (TRIVAGO, 2019). Também se enquadram como *crawlers* serviços de buscas de páginas, como o Google (GOOGLE, 2019b) e o Duckduckgo (DUCKDUCKGO, 2019).

Existem pesquisas relacionadas ao desempenho dos *web crawlers*, como o trabalho de Cho e Garcia-Molina (2000a) que estuda métricas de atualização de cópias locais de dados de servidores externos, bem como o trabalho de Shkapenyuk e Suel (2002) que aborda o

projeto e implementação de um *web crawler* distribuído de alto desempenho. Ambos os trabalhos elucidam que a construção de um *web crawler* de qualidade não é trivial, pois questões como políticas de atualização dos dados, arquitetura do sistema e recursos alocados têm um impacto direto sobre o desempenho desse tipo de aplicação.

Nesse tipo de serviço, é comum a utilização de mecanismos de *cache* ou até mesmo de servidores *proxies* para armazenamento dos preços dos produtos. Coulouris et al. (2011) define a cache da seguinte forma:

> "Uma cache consiste em realizar um armazenamento de objetos recentemente usados em um local mais próximo que a origem real dos objetos remotos em si. Quando um novo objeto é recebido, ele é adicionado a cache, substituindo, se houver necessidade, alguns objetos já existentes. Quando um processo cliente requisita um objeto, o serviço de cache primeiro verifica se possui armazenado uma cópia atualizada desse objeto, caso esteja disponível, o mesmo é entregue ao processo cliente. Se o objeto não estiver armazenado, ou se a cópia não está atualizada, o mesmo é acessado diretamente em sua origem".

Neste trabalho, o mecanismo de *cache* foi utilizado para armazenar os preços dos produtos dos *e-commerces*.

2.5. Problemas combinatórios e técnicas de resolução

Segundo Raidl e Puchinger (2008), nas áreas de pesquisa operacional, matemática aplicada e ciência da computação teórica, a otimização combinatória consiste em encontrar um objeto ótimo entre um conjunto finito de objetos. Em muitos desses problemas soluções por busca exaustiva não são aplicáveis. Técnicas de otimização combinatória operam em problemas nos quais o conjunto solução factível é discreto ou pode ser reduzido a discreto, ou seja, cujo objetivo seja encontrar a melhor solução.

Problemas de otimização combinatória frequentemente aparecem em diversos campos relevantes. Como a determinação ótima de agendas para trabalhos que devem ser executados em uma linha de produção, projeto de redes de comunicação e determinação de rotas eficientes para veículos. Todos esses problemas envolvem a busca de valores para variáveis discretas atreladas a uma função objetivo sujeita a restrições.

2.5.1. Técnicas para a resolução de problemas combinatórios

A maioria dos problemas combinatórios são difíceis de resolver. Isso se dá pelo fato que muitos desses problemas serem NP-completos (GAREY; JOHNSON, 1990). Diversas técnicas de resolução desses problemas foram propostas desde 1970, que podem ser classificadas em duas categorias principais: (i) algoritmos exatos e (ii) algoritmos heurísticos.

Algoritmos exatos garantem encontrar a solução ótima, no entanto o tempo de execução cresce dramaticamente conforme a quantidade de dados de entrada, comumente essa abordagem é adequada para problemas de tamanho pequeno ou médio. Algoritmos heurísticos geralmente são utilizados para casos em que a quantidade de dados da entrada é muito grande, muitas vezes trocam a garantia de convergência da solução ótima por tempo de execução. Esses algoritmos são projetados para obter bons resultados, mas não necessariamente encontram a solução ótima.

Ao considerar abordagens exatas, as seguintes técnicas têm um sucesso significativo: (i) programação linear inteira e (ii) programação dinâmica. Definições dessas técnicas podem ser encontradas nos trabalhos de Grotschel (1993) e Nemhauser e Wolsey (1988a). No campo das abordagens heurísticas, destacam-se: (i) algoritmos genéticos, (ii) algoritmos de descida do gradiente e (iii) algoritmo das formigas, detalhes dessas técnicas são descritas nos trabalhos de Aarts e Lenstra (1997), Beasley e Chu (1996) e Dorigo e Caro (1999). Nas subseções seguintes são detalhadas as abordagens exatas mencionadas, as quais foram utilizadas neste trabalho.

2.5.2. Programação linear inteira

Um problema de programação linear inteira é um problema envolvendo variáveis inteiras, uma função objetivo dependente linearmente das variáveis e um conjunto de restrições expressas como inequalidades.

$$Z_{ILP} = min\{c \cdot x | A \cdot x \geq b, x \geq 0, x \in Z^n\} \tag{2.1}$$

Considere a equação 2.1 em que x é o vetor n-dimensional de variáveis de uma coluna e $c \in R^n$ um vetor n-dimensional. O produto cx é a função objetivo que deve ser minizada. A matriz $A \in R^{m \cdot n}$ e o vetor de colunas m-dimensional $b \in R^m$ juntos definem m restrições de inequalidade.

Problemas de maximização podem ser convertidos a problemas de minimização simplesmente ao inverter o sinal de c. Restrições do tipo “menor que” são trazidos a forma “maior que” ou “igual” pela troca do sinal dos coeficientes, adicionalmente restrições com igualdades podem ser traduzidas para restrições com pares de desigualdades. Logo, podem ser consideradas todos os tipos de restrições fazendo uso das transformações apropriadas.

Os trabalhos de otimização inteira Nemhauser e Wolsey (1988b) e Wolsey (1998) são recomendados para uma abordagem mais detalhada relativa à programação linear inteira.

2.5.3. Programação dinâmica

Programação dinâmica é um método de otimização matemática e um método de programação de computadores. O método divide os problemas em subproblemas (recursivamente) e, é aplicável quando os subproblemas não são independentes, ou seja, que possuem características comuns, as quais podem ser reaproveitadas (CORMEN et al., 2009).

A programação dinâmica tipicamente é aplicada em problemas de otimização, virtualmente qualquer problema de otimização pode ser modelado fazendo uso de programação dinâmica. No entanto, existem soluções recursivas em programação dinâmica que não são eficientes. Em contrapartida, o modelo de programação dinâmica pode ser usado para projetar algoritmos rápidos em casos que o número de estados pode ser controlado (GROTSCHEL, 1993).

2.6. Considerações finais

O estudo de caso desta pesquisa abrange os *e-commerces* de *Magic: The Gathering*, é um caso em que existe a necessidade da compra de vários produtos em diferentes fornecedores, também existem desafios na coleta dos preços devido ao número elevado de requisições para tantos produtos e lojas. Logo, mecanismos de *cache* e algoritmos para resolver problemas de otimização combinatória foram estudados e descritos neste capítulo. O capítulo seguinte apresenta trabalhos que utilizaram esses conceitos para fins semelhantes, que serviram de apoio e referência para essa pesquisa.

Capítulo 3

Trabalhos Relacionados

Esse capítulo apresenta trabalhos que lidaram com problemas de sincronização dos dados e problemas combinatórios de natureza semelhante ao apresentado nesta monografia.

3.1. Synchronizing a database to Improve Freshness

Cho e Garcia-Molina (2000a) comparam métricas e políticas de sincronização visando manter atualizadas as cópias locais de fontes de dados autônomas. O estudo usa dados coletados de 270 sítios Web.

Os autores propõem um *framework* formal que define duas métricas nominadas *freshness* e *age*, que são o número de elementos atualizados e a data de atualização da cópia de uma fonte de dados autônoma. O *framework* faz uso do processo de Poisson para descobrir como os elementos do mundo real irão mudar, pois o processo de Poisson é comumente utilizado para modelar uma sequência de eventos que acontecem aleatoriamente num intervalo fixo no tempo.

O trabalho descreve e compara três políticas de sincronização, nas quais a sincronização de cada página ocorre após um intervalo de tempo predeterminado, variando a magnitude do intervalo para os elementos em cada política, como segue:

1. Política de alocação uniforme: Todas as páginas são sincronizadas no mesmo intervalo de tempo. Por exemplo, se o tempo de atualização definido for 10 minutos, cada página será sincronizada a cada 10 minutos, indiferente do número de atualizações diárias da página.
2. Política de alocação proporcional: Páginas são sincronizadas em intervalos de tempo diferentes, de forma que uma página que tem seu conteúdo alterado mais frequentemente é sincronizada mais vezes, logo quanto mais uma página muda, menor será o seu intervalo de sincronização.

3. Política de alocação ótima: É uma política que fica entre as políticas uniforme e proporcional, sendo aplicável quando existe o conhecimento da frequência de mudança das páginas.

Cho e Garcia-Molina (2000a) concluem que a política de alocação uniforme mantém cópias de bases autônomas mais atualizadas do que a política de alocação proporcional, e também demonstram que a política de alocação ótima é melhor do que ambas as anteriores, mas é preciso conhecer a frequência de mudança das páginas.

Nesta monografia, é importante a definição da política de atualização da cache (cópia local dos dados dos preços dos *e-commerces*) da mesma forma que no trabalho de Cho e Garcia-Molina (2000a). No entanto, Cho e Garcia-Molina (2000a) focam somente ao processo de sincronização, enquanto abrangemos o problema da sincronização e o processamento dos dados armazenados na *cache*.

3.2. Design of a shopbot and recommender system for bundle purchases

A pesquisa de Garfinkel et al. (2006) provê um modelo para computar um plano de compra que busca vários itens oferecidos por diversos vendedores, sendo aplicável quando esses itens estão presentes em promoções por lote de produtos, nos quais comumente o preço dos itens é menor que os seus preços individuais.

Os lotes de produtos e os preços foram coletados por meio de um processo manual de visita extensiva de sítios de venda, no qual diariamente foram feitos acessos a esses sítios e foram registrados seus preços. Os dados coletados foram utilizados pelo algoritmo proposto pelo trabalho.

No trabalho é apresentado um algoritmo chamado *Greedy Addition of Bundles* (GRAB), uma variação do algoritmo de Fisher e Wolsey (1982), que foi implementado e comparado a soluções de programação linear inteira. Os autores defendem que, para os mesmos dados, o GRAB tem um tempo de execução menor que o de pacotes de programação linear inteira da biblioteca CPLEX da linguagem C.

A proposta de Garfinkel et al. (2006) se assemelha a esta monografia por avaliar o desempenho de programação linear inteira para um problema de análise combinatória em diversos itens e fornecedores. No entanto, o processo de coleta foi realizada manualmente, a pesquisa não é aplicada a um ramo específico e o seu foco está em promoções por lote de produtos, diferenciando-se deste trabalho por esses motivos.

3.3. Shopbot 2.0: Integrating recommendations and promotions with comparison shopping

Garfinkel et al. (2008) propõem uma abordagem que integra promoções e serviços de recomendação a serviços de comparação dos *shopbots* atuais. A solução, denominada *shopbot* 2.0, utiliza programação linear inteira para criar um novo buscador de preços.

Em sistemas de recomendações populares como o da Amazon (AMAZON, 2019), quando um cliente seleciona um item, também são exibidos outros N itens relacionados ao escolhido, que estão ranqueados por uma medida específica (como quantidade de compradores que comparam ambos os produtos). Garfinkel et al. (2008) descrevem que nem sempre o item do topo da lista de relacionados é a oferta mais atrativa ao usuário por conta de possíveis promoções. Por exemplo, suponha uma promoção em que a compra acima de 25 dólares terá frete gratuito. Se um usuário comprar um produto e se o sistema de recomendação exibir a primeira opção da lista, resultando em valor abaixo de 25 dólares, o comprador pode não usufruir do frete gratuito, sendo que talvez, o segundo item da lista de recomendação resultasse em um valor acima de 25 dólares, mas que compensasse mais a compra, aumentando a chance da venda ocorrer.

Os autores descrevem que descontos e promoções podem ser de vários tipos, como frete gratuito, porcentagem fixa de desconto para um pacote ou "dois por um". Logo, propõem um modelo de programação linear inteira para resolver esse problema, cuja função de custo maximiza os descontos da lista de preços em adição das promoções aplicáveis. No modelo, são utilizadas restrições para controlar a quantidade de "itens de bônus", cupons de desconto, número de itens necessários para ter acesso ao "item gratuito", quantidade mínima necessária para ganhar um cupom de desconto. O modelo é aplicável para os produtos da Amazon e é descrito em detalhes por Garfinkel et al. (2008).

A abordagem de Garfinkel et al. (2008) foca no uso de programação linear inteira para a resolução de um problema combinatório envolvendo comércio eletrônico, sendo assim diretamente relacionada ao tema deste trabalho (que também resolve um problema combinatório para *e-commerces* utilizando programação linear inteira). No trabalho de Garfinkel et al. (2008), foram abordados modelos de maximização de "economia" e descontos, enquanto abordamos a minimização do custo de compras. Além disso, eles propõem um modelo que atende as necessidades específicas da Amazon.

3.4. Considerações finais

Os trabalhos de Garfinkel et al. (2006, 2008) utilizaram programação linear inteira em problemas de minimização do custo de compras por grupo. No trabalho de Cho e Garcia-Molina (2000a) foi realizada a comparação de métricas aplicáveis a cópias locais de bases

remotas. Ambos os mecanismos (programação linear inteira e cópias de bases remotas) foram utilizados para a resolução do problema proposto neste trabalho. O Capítulo 4 apresenta os materiais e métodos de pesquisa utilizados nesta monografia.

Capítulo 4

Materiais e Métodos

Neste trabalho, objetivou-se propor mecanismos para a coleta de preços de diversos produtos em diferentes fornecedores, comparar algoritmos que encontram a melhor configuração de compra de produtos em vários *e-commerces* e implementar um *shopbot* em um cenário específico (*Magic: The Gathering*). Neste capítulo, são apresentados os materiais e o método de pesquisa para alcançar esses objetivos.

O método de pesquisa é composto por um conjunto de etapas, que são: (i) resolução de problema de minimização de custo, (ii) especificação de mecanismo para coleta de preços e (iii) desenvolvimento do protótipo do *shopbot* e (iv) avaliação dos mecanismos do protótipo. As etapas contêm sub-etapas, que são partições menores das tarefas a executar (Figura 4.1).

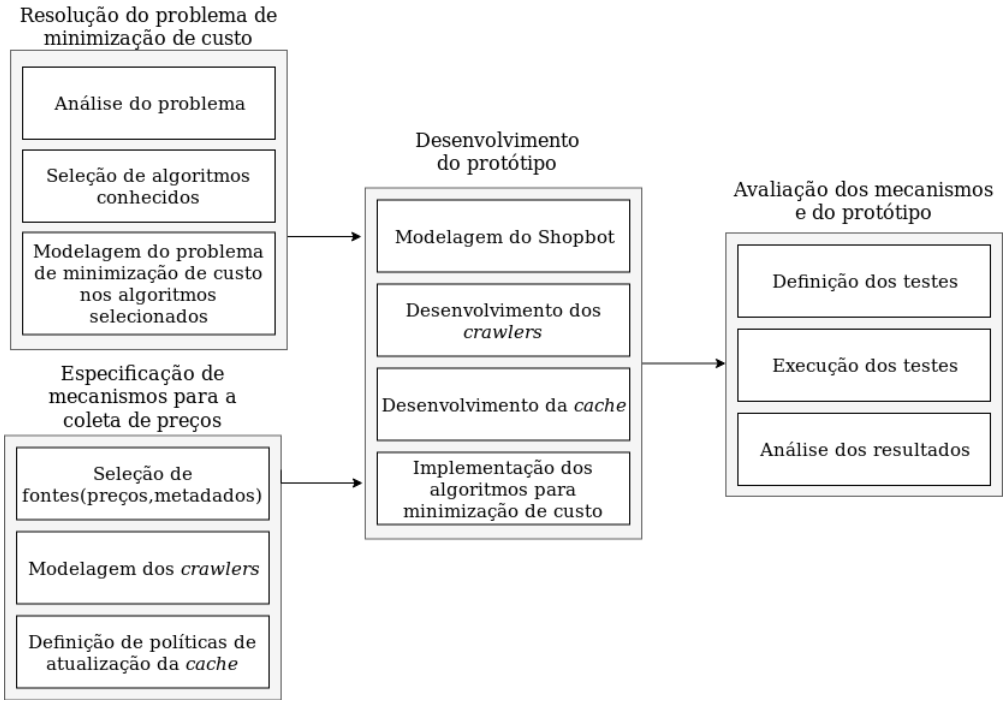

Figura 4.1. Diagrama das etapas da pesquisa

As seções deste capítulo detalham os procedimentos de cada uma das etapas e sub-etapas apresentadas na Figura 4.1.

Quanto aos materiais, o protótipo do *Shopbot* foi desenvolvido na linguagem de programação Python na versão 3.6. O Sistema de Gerenciamento de Base de Dados (SGBD) PostgreSQL (POSTGRESQL, 2019) foi usado para armazenar os dados e metadados dos produtos e fornecedores da aplicação. As principais bibliotecas utilizadas neste trabalho na linguagem Python são: (i) pulp v.1.6.9: provê funções para o modelo de programação linear inteira; (ii) scipy v.1.2.1: provê funções para o modelo de programação quadrática; e (iii) requests v.2.18.4: provê funções para as solicitações HTTP aos *e-commerces*.

4.1. Resolução do problema de minimização de custo

Esta etapa descreve a análise do problema, a seleção de algoritmos conhecidos e a modelagem de algoritmos capazes de encontrar a configuração adequada de lojas para a compra de custo mínimo de diversos produtos.

4.1.1. Análise do problema

No Capítulo 1 foi demonstrada a quantidade mínima de execuções necessárias para um algoritmo de resolução combinatória por força bruta, declarada na equação 1.1. Nesta seção, o problema é analisado a partir de um exemplo.

Considere um cenário hipotético no qual um cliente deseja comprar 10 produtos, denominados P1, P2, P3, P4, P5, P6, P7, P8, P9 e P10, tendo a sua disposição 4 *e-commerces* que comercializam os 10 produtos, denominados E1, E2, E3, E4. A Tabela 4.1 apresenta os preços dos produtos de cada *e-commerce* e a Tabela 4.2 apresenta a quantidade de configurações possíveis para 4 *e-commerces*.

A compra de custo mínimo pode ocorrer em um, dois, três ou quatro *e-commerces*. Tomando-se as configurações que contêm apenas um *e-commerce* (E1, E2, E3 ou E4), haverá uma verificação de preço para cada produto em toda loja, ou seja, 10 verificações de preço por loja. Para as configurações de dois *e-commerces* (E1E2, E1E3, E1E4, E2E3, E2E4, E3E4), a lista de produtos de cada loja dos pares deverá ser percorrida. Portanto, são necessárias 20 verificações de preço para cada par de lojas.

Podemos analisar as quantidades de checagens de preços necessárias para gerar as combinações de um ou mais *e-commerces* na Tabela 4.2. Para todo j, o número de checagens é dado pela quantidade de configurações de compra que contém j lojas, multiplicado por j e pela quantidade de produtos. Para exemplificar, considere o caso em que $j = 2$. Por existirem 6 configurações com 2 lojas e 10 produtos, haverão ao menos $6 \cdot 2 \cdot 10$ checagens de preço para gerar as combinações de duas lojas. Logo, para gerar todas as configurações, o mesmo procedimento deve ser aplicado para $j = 1, 3$ e 4, conforme a Tabela 4.2.

Tabela 4.1. Tabela de preços dos produtos hipotéticos

Produto	Preço E1 (R$)	Preço E2 (R$)	Preço E3 (R$)	Preço E4 (R$)
P1	10,00	15,00	7,00	10,00
P2	5,00	20,00	15,00	9,00
P3	12,00	16,00	12,00	16,00
P4	7,00	20,00	13,00	7,00
P5	9,00	14,50	11,70	13,20
P6	13,00	23,00	9,00	10,00
P7	7,00	10,00	14,00	15,00
P8	19,00	12,00	8,00	15,00
P9	15,00	21,00	7,00	6,00
P10	13,00	18,00	11,00	10,00
Frete	11,00	14,00	12,00	13,00

Tabela 4.2. Quantidade de configurações de compra para 4 *e-commerces*

Número de lojas (j)	**1**	**2**	**3**	**4**
Configurações de compra com j lojas	4	6	4	1
Quantidade mínima de checagens de preço	40	120	120	40

Como no estudo de caso proposto existem mais de 90 *e-commerces* para consultar, e corriqueiramente as listas de produtos possuem entre 60 e 100 produtos, o custo de convergência dessa solução requisitaria no mínimo 2^{90-1} operações, o que é impraticável.

4.1.2. Seleção de algoritmos conhecidos

Nas etapas iniciais da pesquisa, foi desenvolvido um algoritmo de comparação de preços por força bruta que limitava a quantidade de *e-commerces* no resultado. No entanto, essa solução se mostrou inadequada devido ao longo tempo de convergência do algoritmo, mesmo limitando o resultado à configurações até três lojas, as soluções levavam aproximadamente 15 minutos para convergir quando submetidas a listas de 60 cartas. Limitar o algoritmo também implicou na falta de garantia que a configuração devolvida apresentava o custo mínimo.

A solução por força bruta mostrou-se ineficiente e, após a análise dos trabalhos de Garfinkel et al. (2006, 2008), foi verificado a possibilidade da resolução do problema por meio do uso de métodos de pesquisa operacional, como programação linear inteira.

Por se tratar de um problema combinatório em que tipicamente o número de entradas é pequeno, as soluções combinatórias exatas são viáveis, por esse motivo foram selecionados algoritmos de programação linear inteira.

Adicionalmente, durante a execução do algoritmo por força bruta as listas de preços

de cada loja são consultadas diversas vezes, em vários níveis da árvore de execução. Por essa razão é possível reduzir a quantidade de execuções fazendo uso de uma otimização do algoritmo de força bruta com uma técnica de programação dinâmica.

4.1.3. Modelagem do problema de minimização de custo nos algoritmos selecionados

Para a formulação dos modelos, considere um conjunto de modelos de produtos indexados por $I = \{1, ..., n\}$ e um conjunto de fornecedores indexados por $J = \{1, ..., m\}$. Para todo modelo de produto $i \in I$ existe a quantidade desejada de compra q_i. Para todo fornecedor $j \in J$, o frete fixo f_j é considerado se o fornecedor j é designado para a compra de algum produto. Para todo modelo de produto i associado a um fornecedor j existe um custo fixo c_{ij} de se comprar o produto i no fornecedor j, também há o estoque e_{ij} disponível para todo produto i no fornecedor j. A seguir, é apresentado o Modelo Linear 1, que resolve problemas combinatórios com a quantidade de cada item de entrada igual a 1, e os Modelos Lineares 2 e 3, que resolvem problemas com mais unidades de cada item.

Modelo linear 1

Sejam as variáveis x_{ij} e y_j definidas por:

$$x_{ij} = \begin{cases} 1, & \text{se o produto } i \text{ for comprado na loja } j \\ 0, & \text{caso contrário} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se o frete da loja } j \text{ for usado} \\ 0, & \text{caso contrário} \end{cases}$$

O problema consiste em minimizar o custo total de uma compra para todo $q_i = 1$, o custo é composto pela soma dos preços c_{ij} das cartas selecionadas e dos fretes f_j das lojas utilizadas, assim a função objetivo a ser minimizada é expressa na equação 4.1:

$$F = \sum_{j \in J} f_j \cdot y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} \cdot x_{ij} \tag{4.1}$$

A função objetivo está sujeita às restrições:

$$\sum_{j \in J} x_{ij} = 1, \quad \forall\ i \in I \tag{4.2}$$

$$\sum_{i \in I} x_{ij} \leq n \cdot y_j, \quad \forall\ j \in J \tag{4.3}$$

$$y_j, x_{ij} \in \{0, 1\}, \ \forall \ i \in I, j \in J \tag{4.4}$$

A restrição 4.2 garante que um produto é comprado em somente uma loja. A restrição 4.3 ativa a variável y_j para cada *e-commerce* utilizado, fazendo assim o frete y_j ser considerado na função objetivo. A restrição 4.4 delimita o intervalo de valores possíveis para y_j e $x_{i,j}$, que podem ser apenas os números inteiros 0 ou 1.

Por fim, reforçamos que o objetivo desse modelo é minimizar a função 4.1 sujeita as restrições 4.2, 4.3 e 4.4, pois a minimização do custo dessa função implicará na resolução do problema de busca da melhor configuração de compra. Nesse modelo todo $q_i = 1$, ou seja, não são permitidos produtos repetidos como entrada.

Modelo linear 2

Considere um conjunto de instâncias de produtos $K = \{1, ..., q_1, q_1 + 1, ..., q_1 + q_2, ..., q_{n-1} + 1, ..., \sum_{i \in I} q_i\}$, que é composta pela união de i conjuntos $K_i = \{q_{i-1} + 1, ..., \sum_{l=1}^{i} q_l\}$, todo conjunto K_i contém q_i instâncias de produto i. Para toda instância de produto $k \in K$ associada a um fornecedor $j \in J$ existe um custo fixo c_{kj} de se comprar a instância de produto k no fornecedor j. Sejam as variáveis x_{kj} e y_j definidas por:

$$x_{kj} = \begin{cases} 1, & \text{se a instância de produto k for comprado na loja j} \\ 0, & \text{caso contrário} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se o frete da loja j for usado} \\ 0, & \text{caso contrário} \end{cases}$$

O problema consiste em minimizar o custo total de uma compra, que é composto pela soma dos preços c_{kj} das cartas selecionadas e dos fretes f_j das lojas utilizadas, assim a função objetivo a ser minimizada é expressa na equação 4.5:

$$F = \sum_{j \in J} f_j \cdot y_j + \sum_{k \in K} \sum_{j \in J} c_{kj} \cdot x_{kj} \tag{4.5}$$

A função objetivo está sujeita às restrições:

$$\sum_{j \in J} x_{kj} = 1, \quad \forall \ k \in K \tag{4.6}$$

$$\sum_{k \in K} x_{kj} \leq y_j \cdot \sum_{i \in I} q_i, \quad \forall \ j \in J \tag{4.7}$$

$$\sum_{k \in Ki} x_{kj} \leq e_{ij}, \quad \forall\ j \in J,\ \forall\ i \in I \tag{4.8}$$

$$y_j, x_{kj} \in \{0, 1\},\ \forall\ k \in K, j \in J \tag{4.9}$$

A restrição 4.6 garante que uma instância de produto é comprada em somente uma loja. A restrição 4.7 ativa a variável y_j para cada *e-commerce* utilizado, fazendo assim o frete y_j ser considerado na função objetivo. A restrição 4.8 garante que a quantidade de produtos do modelo i comprados na loja j não exceda o estoque de cartas e_{ij}. A restrição 4.9 delimita o intervalo de valores possíveis para y_j e $x_{k,j}$, que podem ser apenas os números inteiros 0 ou 1.

Por fim, reforçamos que o objetivo desse modelo é minimizar a função 4.5 sujeita as restrições 4.6, 4.7, 4.8 e 4.9, pois a minimização do custo dessa função implicará na resolução do problema de busca da melhor configuração de compra.

Modelo linear 3

Sejam as variáveis x_{ij} e y_j definidas por:

$$x_{ij} \in Z, \text{é a quantidade adquirida do item i na loja j}$$

$$y_j = \begin{cases} 1, & \text{se o frete da loja j for usado} \\ 0, & \text{caso contrário} \end{cases}$$

O problema consiste em minimizar o custo total de uma compra, que é composto pela soma dos preços c_{ij} das cartas selecionadas e dos fretes f_j das lojas utilizadas, assim a função objetivo a ser minimizada é expressa na equação 4.10:

$$F = \sum_{j \in J} f_j \cdot y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} \cdot x_{ij} \tag{4.10}$$

A função objetivo está sujeita às restrições:

$$\sum_{j \in J} x_{ij} = q_i, \quad \forall\ i \in I \tag{4.11}$$

$$x_{ij} \leq e_{ij} \quad \forall\ i \in I,\ \forall\ j \in J \tag{4.12}$$

$$\sum_{j \in J} x_{ij} \leq y \cdot \sum_{j \in J} e_{ij} \quad \forall\ i \in I \tag{4.13}$$

$$y_j \in \{0, 1\},\ \forall\ j \in J \tag{4.14}$$

$$x_{ij} \geq 0, x_{ij} \in Z,\ \forall\ i \in I,\ \forall\ j \in J \tag{4.15}$$

A restrição 4.11 garante que a quantidade de produtos comprados não supere ao total desejado. A restrição 4.12 garante que a quantidade usada de cada produto x_{ij} seja menor ou igual a quantidade disponível e_{ij}. A restrição 4.13 garante que o total de itens comprados seja menor que o total disponível. A restrição 4.14 delimita o intervalo de valores possíveis para y_j e a restrição 4.15 dita o intervalo de valores para $x_{i,j}$.

Por fim, reforçamos que o objetivo desse modelo é minimizar a função 4.10 sujeita as restrições 4.11, 4.12, 4.14 e 4.13, pois a minimização do custo dessa função implicará na resolução do problema de busca da melhor configuração de compra.

4.2. Especificação de mecanismos para a coleta de preços

Esta etapa aborda a especificação dos mecanismos que são utilizados para a obtenção dos preços de diversos produtos quando existem muitos fornecedores. Para tal, é necessário definir a fonte dos preços dos produtos em questão, bem como se os metadados destes são coletados para a validação prévia. Também é preciso modelar os coletores e definir as políticas de atualização da *cache* local.

4.2.1. Seleção de fontes

O *shopbot* precisa coletar dois tipos de informação: (i) o preço de cada carta nos *e-commerces*, e (ii) os metadados das cartas, que servem para validar as informações de cada produto antes de transmiti-las adiante na aplicação.

No caso do *Magic: The Gathering*, os preços das cartas podem ser obtidos diretamente nos sítios dos *e-commerces* que as comercializam, e também é possível realizar consultas em outros *shopbots* que catalogam preços das cartas para diversas lojas, sendo o maior deles a Ligamagic.

Seleção de fontes para o download dos metadados das cartas

Na seleção da fonte dos metadados é importante considerar que novos registros podem surgir, nem todas as fontes atualizam suas bases no tempo desejado. Por exemplo, pode ocorrer de novas cartas serem lançadas em um mês e que alguns serviços apenas disponibilizem os dados da nova coleção no próximo mês. Nesse intervalo muitos usuários podem vir a perder

o interesse na ferramenta. Para o *shopbot* desenvolvido neste trabalho, foram consideradas as fontes: (i) Wizards of the Coast card set archive [1], (ii) *Mtgjson* [2] e (iii) *Scryfall* [3].

A própria *Wizards of the Coast* disponibiliza informações das cartas do jogo, no entanto para coleções de lançamento, geralmente disponibiliza as cartas somente na data de lançamento da edição. Pode ocorrer de certas cartas serem exibidas ao público antes do lançamento por terceiros.

O sítio Mtgjson, embora disponibilize os dados das cartas de forma facilitada e prática, não provê as imagens das cartas e as informações para todos os idiomas lançados em todas as cartas, tornando essa opção inviável.

A fonte escolhida foi o serviço Scryfall, pois disponibiliza nomes, habilidades, edições, imagens em diversos idiomas e permite que robôs acessem os seus dados, contanto que haja um intervalo razoável entre as requisições. Empiricamente descobrimos que a aplicação aceita que robôs a acessem contanto que o intervalo entre as requisições seja maior ou igual que 10 segundos. Também é uma fonte que geralmente realiza a atualização dos dados das cartas na data de lançamento.

Seleção de fontes para o download dos preços das cartas

No comércio eletrônico, os preços dos produtos podem ser disponibilizados pelos proprietários dos *e-commerces* ou por outros serviços que mapeiam os preços dos *e-commerces*. Tipicamente *shopbots* agrupam os preços de produtos de diversas fontes, sendo fontes interessantes de consultas de preço por essa razão.

A principal vantagem de consultar preços em outros *shopbots* é a quantidade reduzida de requisições a executar, pois toda requisição é realizada a uma única fonte ao invés de uma requisição por fornecedor para cada produto específico. No entanto, essa medida também gera dependência de um único serviço externo, que pode mudar a forma de disposição das informações ou até mesmo fechar inesperadamente.

No caso do *shopbot* desenvolvido neste trabalho, o acesso é feito ao serviço da Ligamagic e a lojas independentes. A Ligamagic foi escolhida pelo fato da maioria dos *e-commerces* nacionais serem indexados pela mesma, ou seja, basta uma única consulta de preço a uma carta na Ligamagic para obter o seu preço em quase todos os *e-commerces* nacionais. Apesar da Ligamagic dominar o mercado, é interessante propor formas de acesso direto aos *e-commerces* para não depender somente de um único fornecedor de informações. Logo, é válido prover o acesso direto aos preços dos produtos desses *e-commerces* por meio de *web crawlers* específicos para seus sítios.

[1] https://magic.wizards.com/pt-br/products/card-set-archive
[2] https://mtgjson.com/
[3] https://scryfall.com/

4.2.2. Definição de políticas de atualização da cache

O uso do mecanismo de cache viabiliza a redução da quantidade de requisições HTTP que são disparadas a cada busca, para tal propõe-se o uso de *caching* (cópia local de dados remotos), aliado a consulta de preços em outros *shopbots* que já catalogam os preços para diversos *e-commerces*.

Cho e Garcia-Molina (2000a) realizaram um trabalho que comparou métricas e políticas de atualização em cópias locais de sistemas remotos. Eles descrevem que a política de alocação uniforme (na qual todos os elementos são sincronizados seguindo o mesmo intervalo de tempo) tem resultados melhores que a política de alocação proporcional. Por essa razão, é utilizada a política de atualização uniforme para a atualização dos dados remotos, usando um intervalo fixo de 24 horas entre a atualização de cada produto. Portanto, os preços das cartas em cada loja é armazenado em uma base de dados. Além disso, sempre que uma consulta a um dado desatualizado ocorrer a *cache* deverá ser atualizada.

4.2.3. Modelagem dos crawlers

Como no caso do *Magic: The Gathering* existem muitas cartas e lojas para efetuar atualizações na *cache*, o processo de atualização teria que realizar uma requisição para cada carta do jogo em todos os *e-commerces* existentes, o que certamente causaria uma latência impactante na aplicação.

No entanto, todas as cartas do jogo são indexadas pelo *shopbot* da Ligamagic, logo para N cartas a serem buscadas seriam necessárias apenas N requisições ao serviço da Ligamagic para atualizar a base de dados de preços local, ao invés de N requisições para cada *e-commerce* existente. No entanto, nem todos os *e-commerces* estão associados à Ligamagic, por exemplo o *e-commerce* UG Cardshop (UG, 2019), para casos como esse *web crawlers* individuais foram desenvolvidos para atualizar a *cache*. A seguir, são apresentados os modelos desenvolvidos para o *web crawler* da Ligamagic e dos *e-commerces* individuais.

Crawler para a Ligamagic

A Ligamagic se mantém como o maior *shopbot* voltado para o mercado de *Magic: The Gathering* do Brasil, sendo que quase todos os *e-commerces* nacionais são clientes dela. No sítio da Ligamagic, a funcionalidade de busca de preços devolve os preços de todos os *e-commerces* pesquisados em uma única requisição e, por esse motivo, é interessante o desenvolvimento de um *web crawler* para a Ligamagic (Figura 4.2).

A Figura 4.2 ilustra o fluxo de execução do *web crawler* para a Ligamagic por meio de um diagrama de atividade. Primeiramente, são obtidos os nomes das cartas a pesquisar. Para cada carta, é criado um processo que fará a requisição HTTP para a Ligamagic. Todo processo retorna uma relação de preços de todos os *e-commerces* para aquela carta. É possível

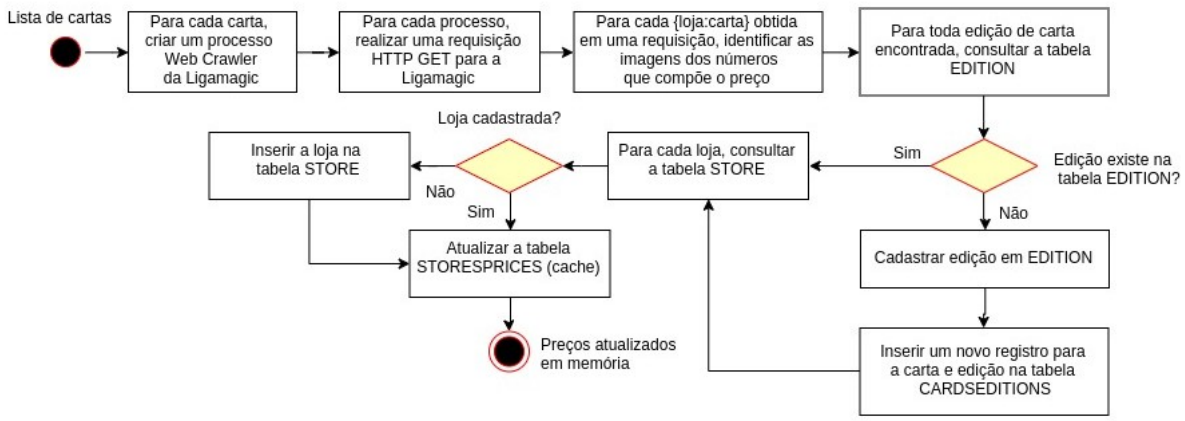

Figura 4.2. Diagrama de atividade do Crawler da Ligamagic

que as requisições retornem edições ou lojas ainda não cadastradas na base de dados, que devem ser cadastradas imediatamente na base de dados. Ao final, os dados de coleta são juntados e a *cache* (STORESPRICES) é atualizada. Durante a consulta é possível que apareçam edições e lojas que não estavam previamente cadastradas na base de dados, nesses casos as suas informações são coletadas e armazenadas.

A princípio assumimos que seria trivial realizar a coleta de preços, apenas por as consultas serem efetuadas diretamente na Ligamagic. No entanto, a informação dos preços das cartas não é disponibilizada em forma de texto pela aplicação, mas sim ofuscada, no formato de imagens.

Ao realizar uma requisição à Ligamagic, é retornada uma figura gerada que contém diversos dígitos. Partes dessa figura são utilizadas em diversos lugares no texto *HyperText Markup Language* (HTML). A Figura 4.3 é uma imagem gerada e retornada pela Ligamagic.

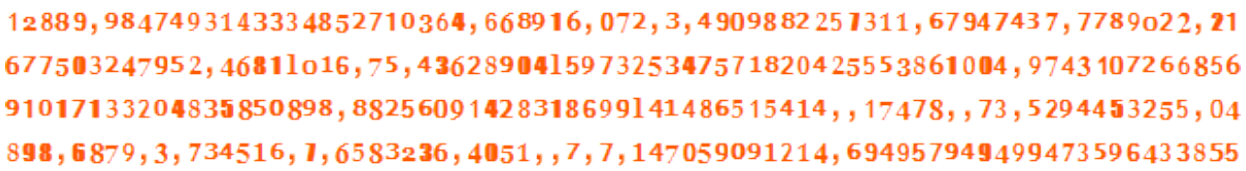

Figura 4.3. Imagem de números retornados pela Ligamagic

No local em que os preços deveriam estar no HTML há uma referência a uma classe *Cascading Style Sheets* (CSS), que indica a posição da figura e o tamanho adequado dos números a serem exibidos. Assim foi possível descobrir os preços de todas as cartas para os *e-commerces* que a comercializavam.

Adicionalmente, embora as figuras geradas pela Ligamagic sejam distintas (em toda consulta é gerada uma imagem em que os números estão em posições diferentes), as posições referenciadas pelo CSS sempre apontam para recortes de imagens iguais, por exemplo, para toda imagem gerada (como o da Figura 4.3), a figura que representa o número 1 exibido ao usuário é a mesma.

Como sempre são utilizadas as imagens dos mesmos números, as quais são armazenadas para posteriormente as comparar as sub-imagens da consulta do *crawler*. Logo, ao tentar "desvendar" qual número um pedaço da Figura 4.3 representa, bastaria comparar os seus bytes com os das imagens gravadas. Caso sejam os mesmos bytes de uma delas, tem-se o número que aquele recorte representa.

Apesar da disposição dos preços da Ligamagic ser diferente por usar imagens de números e os mapear usando classes CSS, essas classes são públicas e são retornadas para qualquer requisição *GET* ao serviço.

Crawler para lojas individuais

Apesar de muitos *shopbots* de *Magic:The Gathering* estarem na Ligamagic, alguns comércios não pagam pelo serviço, como o caso da Lojabat[4] e UG cardshop[5]. Para casos como esses é necessário efetuar as requisições de forma direta.

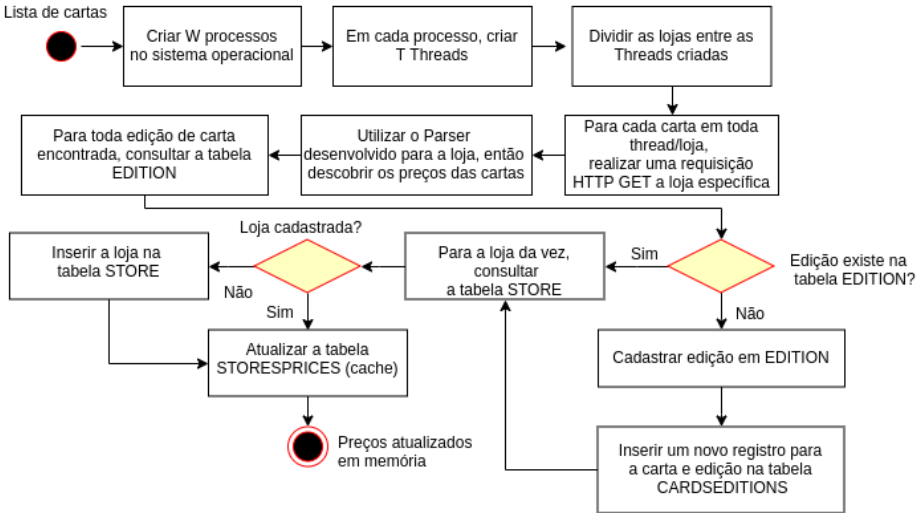

Figura 4.4. Diagrama de atividade das lojas independentes

Considere duas variáveis parametrizáveis w e t para a quantidade de processos e de *threads* por processo. A Figura 4.4 ilustra o diagrama de atividade da coleta de preços para lojas desvinculadas a Ligamagic. A divisão dos processos se dá pelas lojas ao invés das cartas (como no *crawler* da Ligamagic). São criados w processos principais e, em cada processo, são geradas t *threads*. Cada *thread* é responsável por consultar os preços das cartas de um conjunto específico de lojas, em todas elas é realizada uma requisição para cada carta. Quando uma *thread* terminar de executar as requisições, ela atualizará os preços na base de dados. É possível que durante a consulta apareçam edições e lojas que não estavam previamente cadastradas na base de dados, nesses casos as suas informações são coletadas e

[4] https://lojabat.com/
[5] https://www.ugcardshop.com.br

armazenadas. Adicionalmente, para cada loja deve existir um *crawler* específico para tratar as respostas de cada sítio.

***Crawler* para o download dos metadados das cartas.**

Como o serviço provido pelo sítio Scryfall[6] foi selecionado como fonte dos metadados, foi preciso definir o intervalo entre as requisições realizadas ao serviço. É permitido que robôs acessem os dados disponibilizados, contanto que os acessos ocorram em intervalos razoáveis de tempo. Para realizar as requisições é utilizada a biblioteca *requests* da linguagem Python, utilizando um intervalo de 10 segundos entre cada requisição. Esse valor foi escolhido empiricamente devido o arquivo de recomendação robots.txt[7] do Scryfall não especificar um intervalo de acesso para robôs.

4.3. Desenvolvimento do protótipo

Essa etapa aborda a arquitetura do *shopbot* desenvolvido, e também contém pontos referentes ao desenvolvimento dos *web crawlers*, da *cache* e da implementação dos algoritmos para minimização de custo.

4.3.1. Modelagem do shopbot

Tendo definido a estratégia para obtenção dos preços das cartas e os métodos de processamento dos preços, resta a montagem do *shopbot* que retorna a lista de *e-commerces* em que uma compra deve ocorrer. A Figura 4.5 ilustra o processo de execução do *shopbot* por meio de um diagrama de atividade.

Toda vez que uma consulta for executada, ocorrerá uma verificação da existência das cartas, para as disponíveis são utilizados os nomes das cartas em inglês. Depois o programa executa a consulta na *cache* (POSTGRESQL, 2019). Caso todos os itens estejam disponíveis e válidos, eles são obtidos da *cache* e convertidos para uma estrutura de dados adequada às bibliotecas de programação linear inteira. Caso os itens não estejam na *cache* ou estejam depreciados, eles são passados para os *crawlers* da *Ligamagic* e para os *crawlers* para lojas específicas, que executam as requisições e atualizam a *cache*, também retornando os preços das cartas em todos os *e-commerces* para que sejam convertidos para as estruturas das bibliotecas matemáticas.

Concomitantemente, há um processo que periodicamente atualiza a *cache*, fazendo uso de uma fila circular entre as cartas, partindo das coleções mais novas até as coleções mais antigas do jogo.

[6] https://scryfall.com/
[7] https://scryfall.com/robots.txt

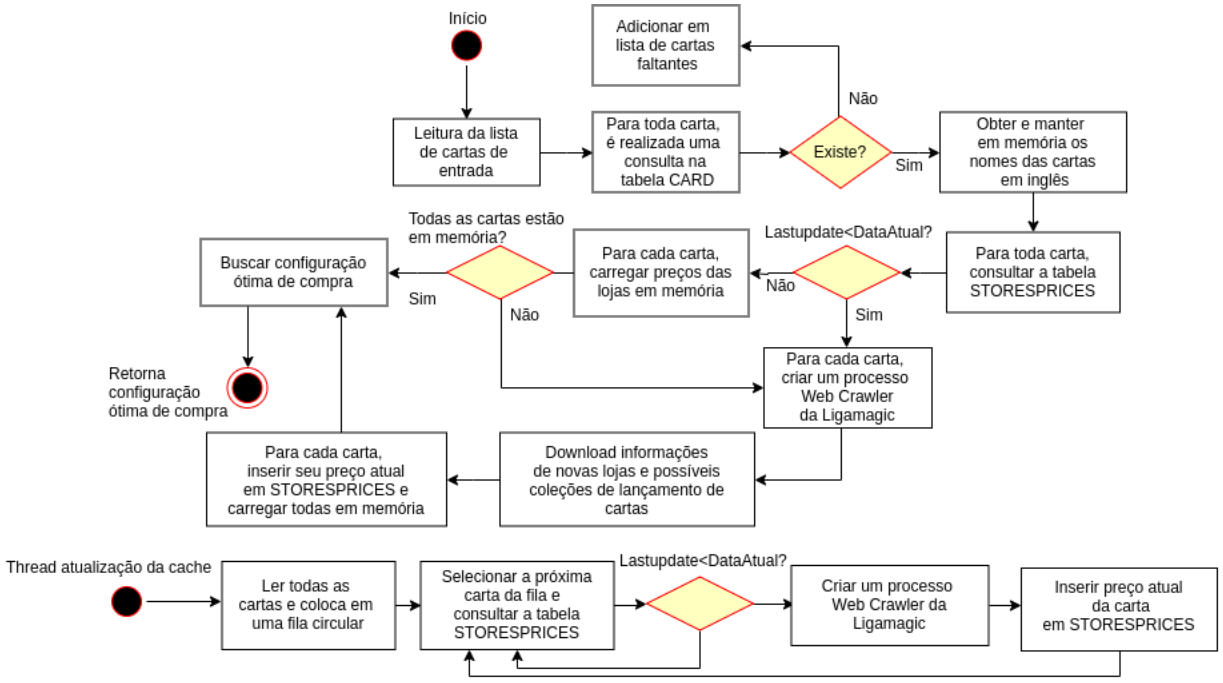

Figura 4.5. Diagrama de atividade do *shopbot*

4.3.2. Implementação dos algoritmos para minimização de custo

Uma vez que os preços estejam coletados, é possível encontrar a combinação de *e-commerces* em que o custo da compra é minimizado. Para a solução de programação linear inteira é utilizada a biblioteca pulp[8] da linguagem Python.

As estruturas de dados de entrada devem ser convertidas para o formato que a biblioteca espera, seguindo as regras de declaração da função objetivo, variáveis no sistema de restrições e os preços das cartas em suas respectivas lojas. O Código 4.1 apresenta um exemplo de definição de um problema de programação linear inteira utilizando o modelo linear 1, na biblioteca *pulp*, em que são declaradas as variáveis, a função objetivo e as restrições.

```
1 x11 = LpVariable("x11", lowBound=0, cat="Integer")
2 x12 = LpVariable("x12", lowBound=0, cat="Integer")
3 x21 = LpVariable("x21", lowBound=0, cat="Integer")
4 x22 = LpVariable("x22", lowBound=0, cat="Integer")
5 y1 = LpVariable("frete-shipping|1", lowBound=0, cat="Integer")
6 y2 = LpVariable("frete-shipping|2", lowBound=0, cat="Integer")
7 problem = LpProblem("Description", LpMinimize)
8 problem += x11 * c11
9 problem += x12 * c12
10 problem += x12 * c21
11 problem += x22 * c22
```

[8] https://pythonhosted.org/PuLP/

```
12 problem += y1 * f1
13 problem += y2 * f2
14 problem += x11 + x21 == 1
15 problem += x12 + 22 == 1
16 problem += x11 + x21 <= 2 * y1
17 problem += x12 + x22 <= 2 * y2
18 problem.solve()
```

Código-fonte 4.1. Exemplo de uso do Pulp com o Modelo Linear 1

No Código 4.1, as variáveis $x11$ a $x22$ são instâncias de variáveis x_{ij} dos três modelos lineares, as constantes cij (preços dos produtos) e fij (frete das lojas) são passadas como parâmetros na definição do problema. As linhas de 1 a 6 declaram as variáveis inteiras disponibilizadas pela biblioteca, a linha 7 cria a estrutura para a função objetivo, as linhas 8 a 13 definem a função objetivo. As linhas 14 a 17 definem as restrições, e por fim, a linha 18 invoca a função de resolução do problema.

4.3.3. Desenvolvimento dos crawlers

A aplicação utiliza três *web crawlers* distintos: um coletor para os metadados das cartas, um coletor para os preços das cartas na Ligamagic e um coletor para os preços em lojas individuais. Todos os três *crawlers* realizam requisições HTTP GET através da biblioteca *requests* da linguagem Python.

4.3.4. Desenvolvimento da cache

É adotada a política de atualização uniforme, considerando o trabalho de Cho e Garcia-Molina (2000b), que demonstrou a sua superioridade em relação à política de atualização proporcional. Além disso, sempre que ocorrerem consultas utilizando cartas desatualizadas a *cache* é atualizada. Os dados são considerados atualizados se a última data de atualização for igual à data corrente. É utilizada uma base de dados para armazenar os preços e quantidades das cartas de cada *e-commerce* e, periodicamente os preços das cartas são atualizados de forma uniforme.

4.4. Avaliação dos mecanismos do protótipo

Esta etapa descreve a definição dos testes que foram executados sobre o protótipo desenvolvido, também apresenta os critérios avaliativos aplicados aos resultados das execuções dos testes.

4.4.1. Definição dos testes

Os *web crawlers* da Ligamagic e individuais são testados utilizando os mesmos produtos como entrada. Foi avaliado o tempo de execução das requisições para 30, 60 e 100 cartas em ambos os *crawlers* e para tal foram utilizada as listas A (TAPPEDOUT.NET, 2019a) e B (TAPPEDOUT.NET, 2019b).

Com relação às soluções utilizando programação linear inteira, são executados testes para 30, 60 e 100 cartas de entrada, variando as quantidades adquiridas de cada carta entre 1, 2 e 4 unidades. Os dados de entrada serão provenientes das listas A (TAPPEDOUT.NET, 2019a) e B (TAPPEDOUT.NET, 2019b). Portanto, para cada modelo/quantidade de cartas/unidades haverão duas execuções, sendo uma para cada lista. Esses testes foram realizados sobre baralhos reais do jogo *Magic: The Gathering*.

Além dos testes utilizando baralhos reais, foram executados os três modelos lineares em conjuntos de entrada gerados aleatoriamente. Para isso, foram gerados 100 conjuntos de 30 cartas, 10 conjuntos de 60 cartas e 10 conjuntos de 100 cartas, variando as quantidades de cada carta entre 1, 2 e 4 unidades para os conjuntos de 30 cartas, e de 1 e 2 unidades para os conjuntos de 60 e 100 cartas.

Adicionalmente, foi realizado um experimento comparativo entre o modelo linear 3, a solução combinatória por força bruta (limitada a combinações de até duas lojas) e entre o buscador de preços por grupo disponibilizado pela Ligamagic. A lista utilizada foi a random0-30, uma lista gerada aleatoriamente com 30 cartas para a execução de testes nos três modelos lineares, disponível na página do *Github* do projeto (NAKAO, 2019).

4.4.2. Avaliação dos testes

Existem dois fatores fundamentais para a avaliação dos mecanismos propostos e do protótipo desenvolvido, que são: (i) retornar a configuração de compra de custo mínimo nos modelos combinatórios e (ii) executar os processos de coleta de informação e convergência combinatória com garantias de tempo de resposta. Adicionalmente, também são comparados os tempos de execução dos modelos propostos.

Dentre todos os testes executados, o tempo de execução da coleta e convergência combinacional foram avaliados por meio de medidas estatísticas de média e desvio padrão entre as execuções que se enquadram nas mesmas categorias.

Os testes são considerados de mesma categoria se a quantidade de cartas de entrada são iguais, ainda que as listas de entrada sejam diferentes. Pode haver uma variação significante no desvio padrão nos modelos lineares, pois devido a características específicas de cada lista como preço e quantidade de lojas, o tempo de convergência deve variar.

4.5. Considerações finais

Apesar do protótipo e das fontes de produtos selecionadas fazerem parte de uma ramificação específica de *e-commerces* (*Magic: The Gathering*), os algoritmos selecionados e os modelos propostos na etapa de resolução do problema de minimização de custo são aplicáveis a diversos ramos comerciais, contanto que hajam produtos e fornecedores envolvidos. Os mecanismos propostos consideram as diferentes fontes de coleta, se é viável ou não o armazenamento dos metadados dos produtos, e a importância da utilização de uma *cache* local para os preços dos produtos. Todos esses fatores são aplicáveis para o desenvolvimento de *shopbots* em geral. O Capítulo 5 apresenta os resultados da aplicação dos métodos apresentados neste capítulo.

Capítulo 5

Resultados e Discussões

Este capítulo apresenta os resultados e discussões referentes aos *web crawlers* e a coleta dos preços e a implementação da *cache*. Também são apresentados e discutidos os resultados dos testes definidos na Seção 4.4. O código fonte, a cópia da base de dados e a saída dos testes executados podem ser encontrados em <https://github.com/nakaosensei/Shopbot-Project> (NAKAO, 2019).

5.1. Base de dados do shopbot

O modelo relacional do protótipo especifica 5 tabelas, conforme ilustrado na Figura 5.1. As tabelas *CARD* e *CARDSEDITIONS* são referentes aos metadados de todas as cartas do jogo e as edições de cada carta. As tabelas *STORE* e *EDITIONS* são referentes a um cadastro simples dos *e-commerces* e edições isoladamente, enquanto a tabela *STOREPRICES* é utilizada pelo mecanismo de *cache* para o armazenamento dos preços das cartas em cada loja.

As tabelas do protótipo são utilizadas como fonte para avaliação do protótipo e servem como estrutura de armazenamento do mecanismo de *cache* de preços das cartas.

Para a tabela *STORE*, armazena-se o nome, país, *link* do sítio, caminho da imagem salva no sistema de arquivos e um id da Ligamagic. Esse último campo é referente a um identificador específico dos sítios vinculados à Ligamagic. É utilizado para consultas de lojas associadas ao serviço.

Na tabela *STOREPRICES*, cada registro é referente ao preço de uma carta em uma loja. Os campos definidos são o nome, país, carta, edição da carta, notação se a carta é brilhante, preço, quantidade, última data de atualização e um campo chamado *lifecounter*, que pode vir a ser usado na administração da *cache* no futuro.

A tabela *CARD* contém o nome em inglês e em português, tipo da carta em inglês e

Figura 5.1. Modelo relacional do *shopbot*

em português, poder, resistência, custo de mana convertido, cores, uma variável para denotar a deleção e uma variável multi-valorada que contém todos os símbolos das edições em que a carta foi impressa. A tabela *CARDSEDITIONS* registra as edições em que as cartas foram impressas, de forma que cada tupla na tabela representa uma edição de uma carta, nela também são registrados os caminhos em que as imagens em português e em inglês da imagem daquela carta em sua edição específica.

Por fim, a tabela *EDITION* armazena os símbolos, nome em inglês e nome em português das cartas. A Ligamagic usa alguns símbolos distintos no seu mapeamento de certas coleções, por essa razão existe um atributo para símbolos específico na Ligamagic.

5.2. Web crawler para o download dos metadados das cartas

Como já discutido na seção 4.2.3, antes de realizar uma requisição a um *e-commerce* para obter uma relação de preços de um grupo de produtos, é interessante verificar se cada produto da entrada realmente existe para evitar tráfego desnecessário, além da possibilidade de expansão de funcionalidades da ferramenta no futuro. Portanto, foi desenvolvido um *web crawler* para

realizar o *download* de informações de todas as cartas já lançadas no jogo, sendo um *script* que pode ser facilmente invocado para atualizar os dados em novos lançamentos.

Foi realizado o *download* de todas as cartas lançadas no jogo, desde a primeira edição até a coleção corrente (*War of the spark*). Os dados das cartas e das coleções foram armazenados nas tabelas *CARD* e *CARDSEDITIONS*, que foram introduzidas na Seção 5.1. Foram adquiridos 19.838 registros para a tabela *CARD* e 40.913 para *CARDSEDITIONS*. As imagens de impressão de cada carta para toda edição também foram obtidas. Um detalhe importante é que houve um intervalo de 10 segundos entre as requisições HTTP, pois dessa forma o servidor do Scryfall não foi inundado com requisições para uma pesquisa que não é afiliada da organização deles. Entretanto, essa medida resultou em uma coleta gradativa, levando aproximadamente uma semana para coletar os metadados das cartas de todas as edições já lançadas. Para novas edições, estima-se que a coleta dos dados das novas cartas dure aproximadamente de uma a três horas, dependendo da quantidade de cartas das edições lançadas.

5.3. Web crawlers para *e-commerces* individuais e para a Ligamagic

Os protótipos de *web crawlers* propostos nessa pesquisa foram implementados e testados, ambos realizaram consultas e armazenamento de cartas provenientes de listas de entrada reais.

No desenvolvimento do *web crawler* para lojas específicas, observou-se que existe um padrão no HTML da maioria dos *e-commerces* associados a Ligamagic, então foi desenvolvido um componente *crawler* genérico que serve para mais de 20 *e-commerces* da Ligamagic. Esse componente existe para fins de comparação de tempo de aquisição do *crawler* para *e-commerces* específicos com o *crawler* feito para a Ligamagic. Em ambos os *crawlers* foram realizados três testes de coleta com 30, 60 e 100 cartas distintas em duas listas reais.

A Tabela 5.1 apresenta as médias e os desvios padrões dos tempos de execução do *web crawler* para lojas específicas e da Ligamagic, aplicado nas listas Tappedout.net (2019a) – Lista A e Tappedout.net (2019b) – Lista B. Os resultados que deram origem aos dados da tabela são apresentados no Apêndice 7.1.

Os tempos médios de execução do *web crawler* de lojas específicas para 20 *e-commerces* são elevados, quando submetido a listas de 60 cartas o *crawler* levou mais de 300 segundos na maioria das execuções, isso ocorre devido à grande quantidade de requisições *HTTP GET* (número de cartas de entrada únicas multiplicada pela quantidade de lojas). O uso de mecanismos de concorrência e paralelismo agilizaram o processo (aproximadamente 6 vezes mais rápido), no entanto, para alcançar melhores resultados utilizando esse modelo, pode-se escalar o serviço de forma horizontal ou vertical. Por exemplo, utilizar um servidor

próprio para coletar os preços de cada *e-commerce* alvo, de forma que todos esses servidores armazenem e busquem informações em uma base de dados unificada.

Tabela 5.1. Tempo médio de execução dos *Web crawlers* em segundos

Crawler	Lista	Md 30	Dp 30	Md 60	Dp 60	Md 100	Dp 100
Ligamagic	Lista A	9,99	0,26	17,9	0,29	29,0	0,71
Ligamagic	Lista B	10,48	0,27	18,68	0,24	29,68	0,3
Específicas	Lista A	158,0	0,75	321,39	0,42	555,16	10,23
Específicas	Lista B	152,02	1,61	301,66	1,81	523,26	3,73

Os tempos médios de execução do *web crawler* da Ligamagic são significativamente menores que os tempos do *web crawler* das lojas específicas. Para 100 cartas, suas coletas levaram em torno de 30 segundos para convergir, enquanto as execuções do *crawler* para lojas específicas levaram cerca de 500 segundos. Essa diferença é justificada pela quantidade reduzida de requisições *HTTP GET* que precisam ser disparadas, sendo disparada uma requisição por carta única. A diferença é suficientemente elevada a ponto de compensar a necessidade de tratamento das informações ofuscadas de preço e quantidade.

5.4. Resolução da minimização combinatória utilizando os modelo lineares

O problema de encontrar a configuração de compra cujo custo seja mínimo é um dos principais problemas a resolver neste trabalho. Para tal, os modelos de programação linear inteira foram implementados fazendo uso da biblioteca pulp do Python. As estruturas de dados da coleta são convertidas para o formato das funções da biblioteca.

Para os testes utilizando baralhos reais, foram realizadas duas execuções com 30, 60 e 100 cartas nos 3 modelos lineares propostos, para cada modelo/quantidade de cartas distintas/unidades foram realizadas duas execuções, totalizando 42 execuções. As cartas de entrada são as listas A (TAPPEDOUT.NET, 2019a) e B (TAPPEDOUT.NET, 2019b), variando a quantidade de cada carta entre 1, 2 e 4. Para cada execução, foram considerados os *e-commerces* que continham aquelas cartas. No Apêndice 7, as Tabelas 7.3 e 7.4 apresentam, o modelo, a lista, a quantidade de cartas distintas, unidades de cada carta, quantidade de lojas total, tamanho do conjunto solução (número de lojas do conjunto solução), tempo de execução e o preço. O tempo resultante é medido em segundos e o preço é dado em reais.

A Tabela 5.2 apresenta a quantidade de cartas distintas usadas, número de unidades de cada carta, as médias e desvios padrões dos modelos 1, 2 e 3, levando em consideração os resultados obtidos a partir dos dados reais obtidos das listas A e B.

Além dos testes executados utilizando os dados reais, foram executados testes

Tabela 5.2. Média dos tempos de execução dos modelos em segundos - dados reais

Quant. de cartas	Núm. de unidades	Modelo 1		Modelo 2		Modelo 3	
		Média	DP	Média	DP	Média	DP
30	1	4,76	0,81	5,54	0,41	5,14	1,06
30	2	-	-	16,69	0,37	5,18	0,44
30	4	-	-	400,48	346,4	7,46	2,2
60	1	12,82	7,33	14,28	8,36	17,96	12,48
60	2	-	-	138,78	53,93	11,16	0,27
60	4	-	-	1039,5	454,5	12,81	4,12
100	1	15,72	5,2	16,18	6,36	13,09	1,24
100	2	-	-	534,5	398,5	60,04	31,22
100	4	-	-	12596,0	11257,0	92,62	71,9

utilizando conjuntos de cartas gerados automaticamente. Tendo sido criados 100 conjuntos de 30 cartas, 10 conjuntos de 60 cartas e 10 conjuntos de 100 cartas. As cartas atribuídas a cada conjunto foram selecionadas aleatoriamente, e não existem cartas repetidas dentro dos conjuntos.

A Tabela 5.3 apresenta a quantidade de cartas distintas usadas, número de unidades de cada carta, as médias e desvios padrões dos modelos 1, 2 e 3, levando em consideração os resultados obtidos a partir dos modelos gerados automaticamente. Os resultados de cada execução dos problemas gerados aleatóriamente estão disponibilizados na página do *github* do protótipo (NAKAO, 2019) no formato *Comma-Separated Values* (CSV).

Tabela 5.3. Médida dos tempos de execução dos modelos em segundos - dados gerados

Quant. de cartas	Núm. de unidades	Modelo 1		Modelo 2		Modelo 3	
		Média	DP	Média	DP	Média	DP
30	1	5,14	3,68	5,13	3,83	4,85	3,29
30	2	-	-	36,81	34,7	6,26	5,01
30	4	-	-	509,85	504,55	5,87	4,87
60	1	9,95	8,03	8,79	6,89	8,41	6,8
60	2	-	-	687,45	1610,25	31,75	51,57
100	1	29,94	22,19	28,92	17,2	24,49	15,95
100	2	-	-	1156,8	1468,34	73,65	71,83

Nos testes executados em que havia apenas um único exemplar de cada carta os 3 modelos convergiram em intervalos inferiores a 1 minuto, ou seja, de fato apresentaram garantias nos tempos de resposta.

A Figura 5.2 exibe o gráfico de barras contendo a média e desvio padrão dos tempos

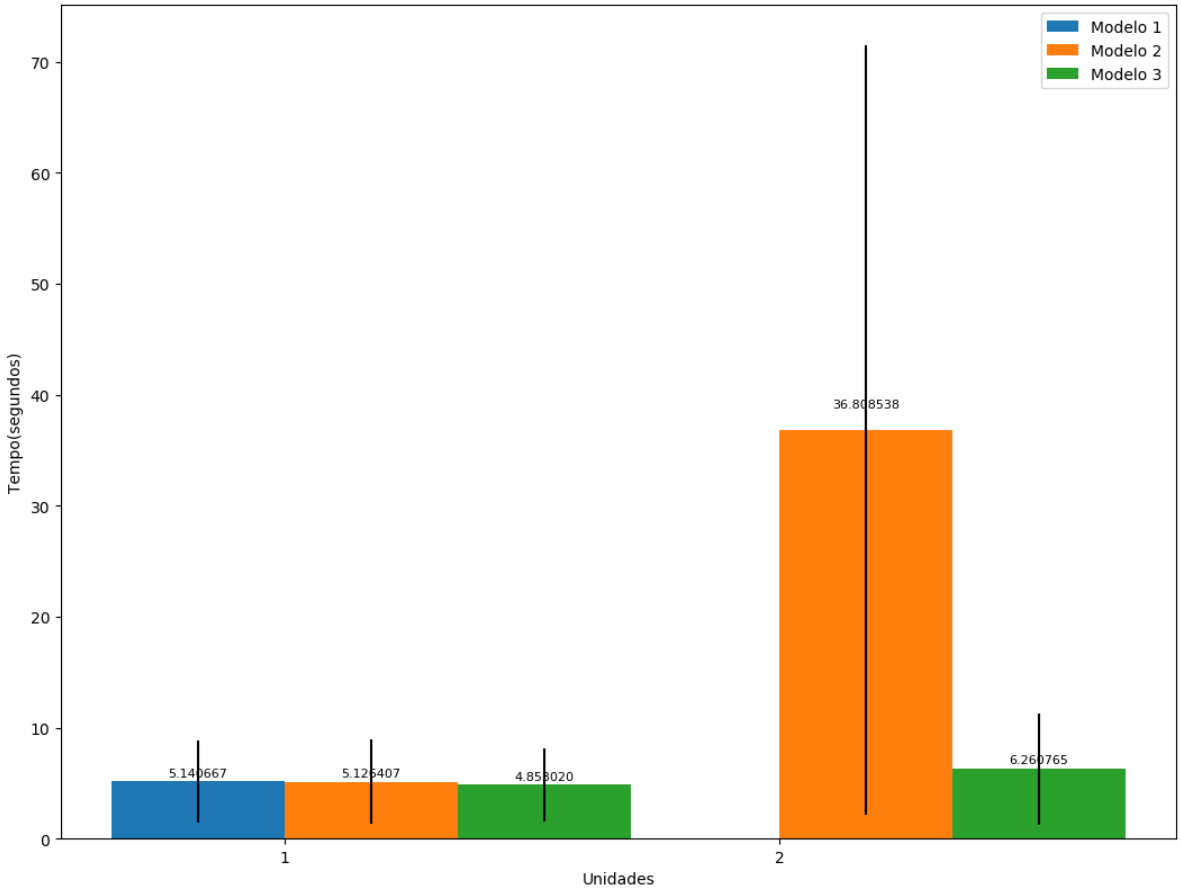

Figura 5.2. Média dos tempos de execução para 30 cartas

de execução dos modelos 1, 2 e 3 para 30 cartas, com uma e duas unidades das listas geradas. O desvio padrão é representado pela linha que cruza as barras. Quando o número de unidades das cartas é incrementado, os tempos de execução do modelo 2 aumentam significativamente (de 5,12 aumentou para 36,8). O modelo 3 também apresentou um aumento no tempo necessário para convergir, no entanto os tempos de resposta do modelo 3 não foram tão elevados (de 4,85 aumentou para 6,26). Esse comportamento se repetiu em todos os cenários que foram testados.

Esperava-se que o modelo linear 1 teria o melhor desempenho dos três modelos para todos os testes, por ele ser restrito a encontrar configurações de compra quando não existem itens repetidos. No entanto, o modelo linear 3 obteve tempos de execução menores que o modelo linear 1 em muitas das execuções.

O modelo linear 2 é muito semelhante ao modelo linear 1, a principal diferença é que no modelo linear 2 é permitido que a quantidade desejada de cada item seja maior que 1, nesses casos são geradas variáveis diferentes para as instâncias dos itens. Além disso, o modelo linear 2 inclui a restrição 4.8, que faz a verificação de disponibilidade de estoque para cada carta e loja, o que eleva o custo computacional da solução. Nesse modelo, a quantidade de variáveis aumenta conforme a quantidade de unidades de cada item, multiplicando o número de variáveis únicas pela quantidade de unidades desejadas. São geradas muitas restrições

para acompanhar esse crescimento no número de variáveis, fazendo com que os tempos de execução do modelo linear 2 sejam drasticamente superiores ao modelo 3 quando o número de unidades é maior que um, como no caso de 30 cartas e quatro unidades em que o tempo de convergência do modelo 2 é aproximadamente 100 vezes maior que o do modelo 3.

Em todas as execuções, para as mesmas entradas, os modelos convergiram para o mesmo resultado, ou seja, devolveram a mesma configuração de compra resultante, todas com o mesmo custo.

O modelo linear 3 obteve a maioria dos melhores tempos de execução, sendo superado em poucos testes para quando o número de unidades dos itens era igual a um. A quantidade de variáveis do modelo não aumenta de acordo com as unidades de cada item e, as restrições aumentam em uma escala pouco significativa. Apesar do modelo 3 conter muitas restrições, a maioria delas são restrições do tipo 4.12, cujo custo computacional de solução pela biblioteca `pulp` é baixo. Justificando assim o tempo de resposta da solução.

O desvio padrão foi elevado em todas as execuções dos testes. Uma das causas desse fenômeno é a divergência nos preços dos dados de entrada dos conjuntos, ou seja, o tempo de execução dos modelos varia de acordo com os escalares dos dados de entrada fornecido. Além disso, a quantidade de lojas que disponibilizam os produtos podem variar em função da lista de produtos utilizada, ambos os fatores justificam as diferenças dos resultados entre os testes dos conjuntos reais para os gerados aleatoriamente, a diferença dos dados e também a quantidade de amostras utilizadas nos testes.

Adicionalmente, um experimento comparativo foi realizado usando a lista random0-30 entre o modelo linear 3, a solução por força bruta e o comprador por grupos da Ligamagic (19/11/2019 às 14:00). Após executar as três abordagens, observou-se que o modelo linear 3 e o algoritmo por força bruta indicaram a compra nas lojas Magic Leste e Multiverso Cards, no valor aproximado de 55,17 reais, enquanto a solução da Ligamagic indicou a compra na loja Cards Outlet, com o valor de 63,45 reais. Os resultados dessas execuções estão disponíveis na página do *Github* do projeto.

5.5. Cache

O desenvolvimento da *cache* envolveu a definição das políticas de atualização, a seleção estrutura de gravação dos dados e a implementação da forma de gravação e coleta dos dados no disco.

Quanto à política de atualização, os dados são considerados atualizados se a data de atualização for igual à corrente. A realização de uma nova consulta deve ocorrer se dados estiverem desatualizados.

A *cache* é representada pela tabela do banco de dados *STORESPRICES*. A base escolhida foi o *PostgreSQL* (POSTGRESQL, 2019), por ser um banco de dados robusto, que

lida com questões de atomicidade, relacionamentos e dá suporte para inserções massivas de dados via métodos especiais.

Na tabela *STORESPRICES*, cada tupla armazena o preço de um produto em uma loja específica, isso implica que durante uma consulta ao *Shopbot* deve ocorrer uma inserção para cada item de todo *e-commerce* que o tenha em estoque, por exemplo, em um cenário com 100 produtos e 100 *e-commerces*, seriam necessárias 10000 inserções na base de dados.

A biblioteca *psycopg2* disponibiliza o método *execute_values* (GREGORIO, 2019), que é um método projetado para lidar com entrada massiva de dados em um banco de dados *PostgreSQL*. Dessa forma, foram realizadas inserções massivas em um curto intervalo de tempo. No entanto, operações na base dados com a cláusula *WHERE* ainda demoram para executar, por essa razão, toda vez que uma consulta a um fornecedor externo ocorrer, os registros antigos contendo os preços dos produtos buscados serão deletados, para então serem reinseridos.

5.6. Considerações finais

Neste capítulo foram apresentados os resultados dos mecanismos propostos nesta pesquisa. Utilizamos paralelismo e concorrência nas abordagens propostas para a coleta de dados e, também concluímos que a consulta a fontes detentoras das informações de muitos fornecedoras é uma estratégia viável e possível de executar com somente uma máquina. Quanto à busca da melhor configuração de compra, dos três modelos desenvolvidos o modelo linear 3 obteve resultados melhores na grande maioria das execuções, por exemplo, para 30 cartas e quatro unidades o modelo 2 foi levou 100 vezes o tempo do modelo 3 para convergir. Sendo assim, o modelo linear 3 é adequado para aplicações em sistemas que visam minimizar a compra de diversos produtos em diferentes fornecedores. O uso de um mecanismo de *caching* reduz significativamente o tempo de resposta da aplicação, pois para consultas de cartas válidas na *cache*, reduz-se o tempo de coleta nas fontes externas para a apresentação dos resultados.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho, foram implementados e avaliados mecanismos para efetuar a pesquisa de preços em uma lista de produtos em um conjunto de *e-commerces* visando encontrar o subconjunto de *e-commerces* em que a lista de produtos tenha o custo mínimo. Por meio do estudo de caso dirigido aos *e-commerces* de *Magic: The Gathering*, foram encontradas configurações ótimas de compra com garantias de tempo de resposta.

No desenvolvimento dos *web crawlers*, o uso de outros buscadores de preços como fonte de dados aliado a técnicas de paralelismo otimizou tempo de resposta da aplicação. No entanto, a legalidade do uso dessa estratégia depende da fonte escolhida, já que o arquivo *robots.txt* sempre deve ser respeitado. Além disso, mudanças na forma de exposição dos fornecedores impactam diretamente a capacidade de coleta da aplicação.

Os modelos de otimização combinatória propostos convergem para a solução ótima, contanto que o problema seja factível. Dentre os três modelos lineares criados, o modelo 3 é o mais adequado em todas as situações testadas. Esse modelo não é aplicável a somente cartas de *Magic: The Gathering*, pois pode resolver qualquer problema que se encaixe nas mesmas condições.

Os mecanismos de *caching* reduziram a quantidade de acessos aos dados remotos. Para sua implementação, questões referentes aos intervalos e formas de gravação no disco devem ser consideradas. Em cenários em que muitas inserções em lote precisam ocorrer, o SGBD *PostgreSQL* (POSTGRESQL, 2019) é uma alternativa interessante, pois possui suporte para esse tipo de operação.

Para as consultas-alvo do estudo de caso, os tempos de execução atendem aos usuários interessados na compra dos produtos, e retornam as configurações de compra cujo custo é mínimo. No entanto, por se tratar de um problema de otimização combinatória, ao aumentar o conjunto de entrada o tempo de resposta da aplicação pode crescer consideravelmente. Além disso, o problema da coleta persiste caso coletores de preços externos não possam ser

utilizados ou não existam.

Em trabalhos futuros, arquiteturas distribuídas podem ser especificadas e avaliadas para coletar informações em muitos *e-commerces* diretamente, escalando o serviço de forma horizontal ou vertical. Além disso, podem haver soluções melhores para o problema combinatório proposto, seja por meio de outras modelagens lineares ou outros métodos, por exemplo, com o uso de otimizações heurísticas. Adicionalmente, o modelo linear 3 pode ser testado em outros contextos de busca com muitos itens em diversos fornecedores.

Capítulo 7

Apêndice - Tabelas de Resultados

Este apêndice apresenta os resultados das execuções utilizando os dados reais nos *web crawlers* e nos modelos lineares. Foram realizados testes usando as listas A (TAPPEDOUT.NET, 2019a) e B (TAPPEDOUT.NET, 2019b). A Seção 7.1 apresenta o resultado das coletas nos *web crawlers*. A Seção 7.2 mostra os resultados do uso dos três modelos lineares propostos nas listas A e B. Os resultados das execuções dos modelos nas listas de compra geradas aleatoriamente estão disponíveis na página do *Github* do projeto(NAKAO, 2019) no formato CSV.

7.1. Testes nos *Web Crawlers*

Tabela 7.1. Testes no *Crawler* da Ligamagic em segundos

Lista	30 cartas	60 cartas	100 cartas
`Lista A`	9,72	18,32	29,20
`Lista A`	10,35	17,7	29,75
`Lista A`	9,9	17,69	28,04
`Lista B`	10,6	18,44	30,1
`Lista B`	10,11	18,6	29,53
`Lista B`	10,74	19,01	29,4

7.2. Testes nas listas A e B

Nesta seção estão expostos os resultados dos testes executados com as listas A (TAPPEDOUT.NET, 2019a) e B (TAPPEDOUT.NET, 2019b).

Tabela 7.2. Testes no *Crawler* das lojas específicas em segundos

Lista	30 cartas	60 cartas	100 cartas
Lista A	157,17	321,82	542,28
Lista A	157,84	321,52	567,31
Lista A	158,98	320,82	555,89
Lista B	150,76	299,88	528
Lista B	151	304,14	522,90
Lista B	154,29	300,96	518,89

Tabela 7.3. Execuções dos modelos lineares nas listas A e B em segundos - Parte 1

Modelo	Lista	Cartas	Unids	Lojas	Tam	Tempo	Preço
MODELO 1	LISTA A	30	1	119	3	5,57	198,1
MODELO 1	LISTA B	30	1	120	5	3,95	503,62
MODELO 1	LISTA A	60	1	120	6	20,15	358,74
MODELO 1	LISTA B	60	1	120	7	5,48	827,02
MODELO 1	LISTA A	100	1	120	8	20,93	539,75
MODELO 1	LISTA B	100	1	120	10	10,52	1121,03
MODELO 2	LISTA B	30	1	120	5	5,14	503,62
MODELO 2	LISTA A	30	1	119	3	5,95	198,1
MODELO 2	LISTA B	30	2	120	8	16,32	1023,34
MODELO 2	LISTA A	30	2	119	7	17,06	373,21
MODELO 2	LISTA A	30	4	119	11	746,89	809,82
MODELO 2	LISTA B	30	4	120	12	54,08	2045,05
MODELO 2	LISTA A	60	1	120	6	22,65	358,74
MODELO 2	LISTA B	60	1	120	7	5,92	827,02
MODELO 2	LISTA A	60	2	121	9	192,71	650,18
MODELO 2	LISTA B	60	2	120	11	84,85	1665,38
MODELO 2	LISTA B	60	4	120	18	585,0	3343,91
MODELO 2	LISTA A	60	4	120	15	1494,0	1362,22
MODELO 2	LISTA B	100	1	120	10	9,81	1121,03
MODELO 2	LISTA A	100	1	120	8	22,54	539,75
MODELO 2	LISTA A	100	2	120	13	933,0	1046,5
MODELO 2	LISTA B	100	2	120	14	136,0	2233,83
MODELO 2	LISTA A	100	4	120	21	23853,0	2069,69
MODELO 2	LISTA B	100	4	120	25	1339,0	4477,04
MODELO 3	LISTA A	30	1	119	3	6,2	198,1

Tabela 7.4. Execuções dos modelos lineares nas listas A e B em segundos - Parte 2

Modelo	Lista	Cartas	Unids	Lojas	Tam	Tempo	Preço
MODELO 3	LISTA B	30	1	120	5	4,08	503,63
MODELO 3	LISTA B	30	2	120	8	5,63	1023,34
MODELO 3	LISTA A	30	2	119	7	4,74	373,21
MODELO 3	LISTA A	30	4	119	11	9,66	746,89
MODELO 3	LISTA B	30	4	120	12	5,25	2045,05
MODELO 3	LISTA A	60	1	120	6	30,44	358,75
MODELO 3	LISTA B	60	1	120	7	5,49	827,02
MODELO 3	LISTA A	60	2	120	9	10,88	681,5
MODELO 3	LISTA B	60	2	120	11	11,43	1665,38
MODELO 3	LISTA A	60	4	120	15	16,93	1363,2
MODELO 3	LISTA B	60	4	120	18	8,69	3343,91
MODELO 3	LISTA B	100	1	120	10	11,85	1121,03
MODELO 3	LISTA A	100	1	120	8	14,33	539,75
MODELO 3	LISTA A	100	2	120	13	91,26	1046,69
MODELO 3	LISTA B	100	2	120	14	28,83	2233,83
MODELO 3	LISTA B	100	4	120	25	20,71	4477,04
MODELO 3	LISTA A	100	4	120	21	164,52	2069,7

Referências

AARTS, E.H.L.; LENSTRA, J.K. *Local search in combinatorial optimization*. [S.l.]: Wiley-Interscience, 1997. (Wiley-Interscience series in discrete mathematics and optimization). ISBN 0471-94822-5.

AMAZON. *Amazon Web Page*. 2019. <https://www.amazon.com.br>. Acessado: 2019-06-15.

BEASLEY, J.E; CHU, P.C. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, v. 94, n. 2, p. 392 – 404, 1996. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/037722179500159X>.

BOTICARIO. *Boticario Web Page*. 2019. <https://www.boticario.com.br>. Acessado: 2019-06-15.

BUSCAPE. *Buscape Web Page*. 2019. <https://www.buscape.com.br>. Acessado: 2019-06-15.

CAMELCAMELCAMEL. *CamelCamelCamel Web Page*. 2019. <https://camelcamelcamel.com>. Acessado: 2019-06-15.

CHO, Junghoo; GARCIA-MOLINA, Hector. Synchronizing a database to improve freshness. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2000. (SIGMOD '00), p. 117–128. ISBN 1-58113-217-4. Disponível em: <http://doi.acm.org/10.1145/342009.335391>.

CHO, Junghoo; GARCIA-MOLINA, Hector. Synchronizing a database to improve freshness. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 29, n. 2, p. 117–128, maio 2000. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/335191.335391>.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. *Introduction to Algorithms, Third Edition*. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262033844, 9780262033848.

CORREIOS. *Página dos correios*. 2019. <http://www.correios.com.br>. Acessado: 2019-05-03.

COSMETICOS, Epoca. *Epoca Cosmeticos Web Page*. 2019. <https://www.epocacosmeticos.com.br>. Acessado: 2019-06-15.

COSMETICOS, Nikkey. *Nikkey Cosmeticos Web Page*. 2019. <https://www.nikkeycosmeticos.com.br>. Acessado: 2019-06-15.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; BLAIR, Gordon. *Distributed Systems: Concepts and Design*. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011, 9780132143011.

DORIGO, Marco; CARO, Gianni Di. Ant colony optimization: A new meta-heuristic. In: . [S.l.: s.n.], 1999. v. 2, p. 1477 Vol. 2. ISBN 0-7803-5536-9.

DUCKDUCKGO. *DuckDuckGo Web Page*. 2019. <https://duckduckgo.com>. Acessado: 2019-06-15.

EBIT. *Webshoppers 37 edição*. 2017. <https://www.ebit.com.br/webshoppers>. Acessado: 2019-05-03.

EBIT. *Webshoppers 38 edição*. 2018. <https://www.ebit.com.br/webshoppers>. Acessado: 2019-08-03.

FISHER, Marshall L.; WOLSEY, Laurence. On the greedy heuristic for continuous covering and packing problems. *Siam Journal on Algebraic and Discrete Methods*, v. 3, 12 1982.

GAREY, Michael R.; JOHNSON, David S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

GARFINKEL, Robert; GOPAL, Ram; PATHAK, Bhavik; YIN, Fang. Shopbot 2.0: Integrating recommendations and promotions with comparison shopping. *Decision Support Systems*, v. 46, n. 1, p. 61 – 69, 2008. ISSN 0167-9236. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167923608000869>.

GARFINKEL, Robert; GOPAL, Ram; TRIPATHI, Arvind; YIN, Fang. Design of a shopbot and recommender system for bundle purchases. *Decision Support Systems*, v. 42, n. 3, p. 1974 – 1986, 2006. ISSN 0167-9236. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167923606000741>.

GOOGLE. *Google Shopping Web Page*. 2019. <https://www.google.com/shopping?hl=pt-BR>. Acessado: 2019-06-15.

GOOGLE. *Google Web Page*. 2019. <https://www.google.com>. Acessado: 2019-06-15.

GREGORIO, Daniele Varrazzo Federico Di. *Psycopg2 extras web page*. 2019. <http://initd.org/psycopg/docs/extras.html>. Acessado: 2019-06-15.

GROTSCHEL, Konrad-Zuse-Zentrum M. *Geometric Algorithms and Combinatorial Optimization*. [S.l.: s.n.], 1993.

LIGAMAGIC. *Ligamagic Web Page*. 2019. <https://ligamagic.com>. Acessado: 2019-06-15.

LIU, Bing; MENCZER, Filippo. Web crawling. In: _____. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 311–362. ISBN 978-3-642-19460-3. Disponível em: <https://doi.org/10.1007/978-3-642-19460-3_8>.

NAKAO, Thiago Alexandre. *Shopbot github page*. 2019. <https://github.com/nakaosensei/Shopbot-Project>. Acessado: 2019-06-10.

NEMHAUSER, George L.; WOLSEY, Laurence A. *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience, 1988. ISBN 0-471-82819-X.

NEMHAUSER, George L.; WOLSEY, Laurence A. *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience, 1988. ISBN 0-471-82819-X.

POSTGRESQL. *PostgreSQL web page*. 2019. <https://www.postgresql.org/>. Acessado: 2019-06-10.

RAIDL, Günther; PUCHINGER, Jakob. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: _____. [S.l.: s.n.], 2008. v. 114, p. 31–62.

Shkapenyuk, V.; Suel, T. Design and implementation of a high-performance distributed web crawler. In: *Proceedings 18th International Conference on Data Engineering*. [S.l.: s.n.], 2002. p. 357–368. ISSN 1063-6382.

TAPPEDOUT.NET. *A Commander deck sample*. 2019. <http://tappedout.net/mtg-decks/17-06-19-the-ur-dragon/?cb=1560782444>. Acessado: 2019-06-10.

TAPPEDOUT.NET. *A Commander deck sample*. 2019. <http://tappedout.net/mtg-decks/my-five-colors-wait-for-it-victor/>. Acessado: 2019-06-10.

TRIVAGO. *Trivago Web Page*. 2019. <https://www.trivago.com.br>. Acessado: 2019-05-03.

TURBAN, Efraim; LEE, Jae K.; KING, David; LIANG, Ting Peng; TURBAN, Deborrah. *Electronic Commerce 2010*. 6th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136100368, 9780136100362.

UG. *UG card shop web page*. 2019. <https://www.ugcardshop.com.br/>. Acessado: 2019-06-12.

WIZARDS. *Magic Formats*. 2019. <https://magic.wizards.com/en/game-info/gameplay/rules-and-formats/formats>. Acessado: 2019-05-03.

WIZARDS. *Magic Sets*. 2019. <https://magic.wizards.com/en/products/card-set-archive>. Acessado: 2019-05-03.

WIZARDS. *Wizards of the Coast story*. 2019. <https://company.wizards.com/content/company>. Acessado: 2019-05-03.

WOLSEY, LA. Wiley-interscience series in discrete mathematics and optimization. *Integer Programming*, 1998.

WONG, Jimmy. *New to Magic?* 2018. <https://magic.wizards.com/en/new-to-magic>. Acessado: 2019-05-03.

YAHOO. *Yahoo Shopping Web Page*. 2019. <https://shopping.yahoo.com>. Acessado: 2019-06-15.

ZAPLY. *Zaply Web Page*. 2019. <https://zaply.com.br>. Acessado: 2019-05-03.

ZHANG, Jie; JING, Bing. The impacts of shopbots on online consumer search. In: . [S.l.: s.n.], 2011. p. 1 – 10.