

MEC-SETEC

INSTITUTO FEDERAL DE MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

**USO DE METAHEURÍSTICA EM UMA
FERRAMENTA DE COTAÇÃO PARA COMPRAS
DE CARTAS DE *MAGIC: THE GATHERING***

Luiz Eduardo Pereira

Orientador: Prof. Dr. Bruno Ferreira

Coorientador: Prof. Me. Diego Mello da Silva

Formiga - MG

2019

Luiz Eduardo Pereira

**USO DE METAHEURÍSTICA EM UMA
FERRAMENTA DE COTAÇÃO PARA COMPRAS
DE CARTAS DE *MAGIC: THE GATHERING***

Trabalho de Conclusão de Curso apresentado
ao Instituto Federal Minas Gerais - Campus
Formiga, como requisito parcial para obten-
ção do título de Bacharel em Ciência da Com-
putação.

Orientador: Prof. Dr. Bruno Ferreira

Coorientador: Prof. Me. Diego Mello da Silva

Formiga - MG

2019

Pereira, Luiz Eduardo
P436u Uso de Metaheurística em uma Ferramenta de Cotação para
compras de Magic: The Gathering / Luiz Eduardo Pereira. --
Formiga : IFMG, 2019.
73p. : il.

Orientador: Prof. Dr. Bruno Ferreira
Co-orientador: Prof. Msc. Diego Mello da Silva
Trabalho de Conclusão de Curso – Instituto Federal de Educação,
Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Simulated Annealing. 2. Scrapy. 3 . MongoDB. 4. Vue.
5. Flask. I. Ferreira, Bruno. II. Silva, Diego Mello da. III. Título.

CDD 004

LUIZ EDUARDO PEREIRA

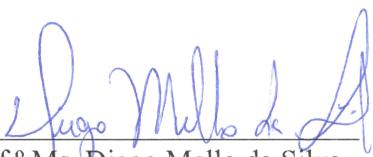
**USO DE METAHEURÍSTICA EM UMA FERRAMENTA DE
COTAÇÃO PARA COMPRAS DE CARTAS DE *MAGIC: THE
GATHERING***

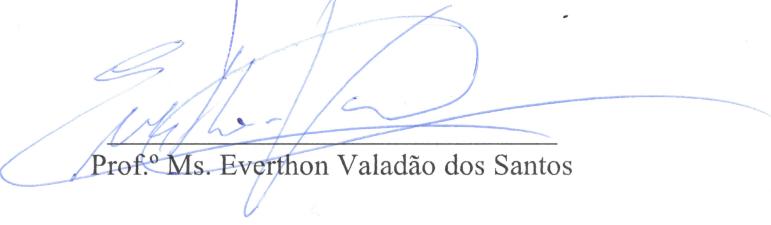
Trabalho de Conclusão de Curso apresentado ao
Instituto Federal de Minas Gerais - Campus
Formiga, como Requisito parcial para obtenção do
título de Bacharel em Ciência da Computação.

Aprovado em: 29 de Novembro de 2019.

BANCA EXAMINADORA


Prof.º Dr. Bruno Ferreira


Prof.º Ms. Diego Mello da Silva


Prof.º Ms. Everthon Valadão dos Santos


Prof.º Ms. Wallace de Almeida Rodrigues

AGRADECIMENTOS

Agradeço aos meus pais pela força e incentivo. Foi somente através deles que foi possível tornar o meu sonho em realidade.

Agradeço aos professores Bruno Ferreira e Diego Mello por passar seus conhecimentos para me ajudar a construir este trabalho, além de todos os ensinamentos ao longo do curso.

Agradeço aos meus amigos e familiares pelo apoio ao longo da vida. Em especial, minha amiga Lavínia, que nunca me deixou desanimar, elogiando quando merecia, criticando quando precisava.

*Haverá um momento em que teremos que
escolher entre o que é fácil e o que é certo.
(Alvo Dumbledore)*

RESUMO

Este trabalho apresenta o desenvolvimento de uma SPA (*Single Page Application*) para cotação de preços para compras de cartas de *Magic: The Gathering*. Este sistema foi nomeado como *ScryX*. O objetivo é que os usuários utilizem o sistema para enviar listas de cartas e, como resultado, recebam sugestões de compras que visam encontrar uma combinação de lojas que minimizam o preço das compras. Esta aplicação divide-se em três principais módulos: SPA, responsável pela interação com o usuário, recebendo dados e exibindo resultados de cotações de preços; *Web Crawler*, responsável pela busca de informações de cartas em centenas de lojas localizadas no Brasil; *Simulated Annealing*, a metaheurística responsável pela escolha das lojas para a solução. Foi utilizada a linguagem de programação Python, juntamente com os frameworks Vue, Flask e Scrapy. Para a persistência dos dados, utilizou-se o banco de dados MongoDB. A ferramenta desenvolvida foi testada em um conjunto de instâncias contendo *decks* que variam de 75 a 100 cartas cada. Os resultados obtidos foram comparados com a ferramenta de cotação disponível no site LigaMagic, e obteve de 8% a 21% de economia no valor da compra, mostrando ser uma ferramenta com potencial de uso para compradores de *Magic*.

Palavras-chave: *Simulated Annealing*, Scrapy, MongoDB, Vue, Flask.

ABSTRACT

This work presents the development of a SPA (Single Page Application) for price quotation to purchase cards from Magic: The Gathering. This system is called ScryX. The goal is for users to use the system to send card lists, and as a result receive shopping suggestions aimed at finding a combination of stores that minimize the price of purchases. This application is divided in three main modules: SPA, responsible for interacting with the user, receiving data and displaying price quotation results; Web Crawler, responsible for searching information in hundreds of stores located in Brazil; Simulated Annealing, the metaheuristic responsible for choosing the stores for the solution. The Python programming language was used in conjunction with Vue, Flask and Scrapy frameworks. For data persistence, the MongoDB database was used. The developed tool was tested on a set of instances containing decks ranging from 75 to 100 cards each. The results were compared with the quotation tool available on LigaMagic website, and obtained 8% to 21% savings on the purchase price, showing to be a tool with potential use for Magic buyers.

Keywords: Simulated Annealing. Scrapy. MongoDB. Vue. Flask.

LISTA DE ILUSTRAÇÕES

Figura 1 – Fronteira de Pareto.	31
Figura 2 – Fases do Processo Unificado.	38
Figura 3 – Diagrama de caso de uso.	39
Figura 4 – Módulos da aplicação.	40
Figura 5 – GUI <i>Login</i> .	41
Figura 6 – GUI Cadastro.	42
Figura 7 – GUI Perfil - Usuário.	42
Figura 8 – GUI Perfil - Administrador.	43
Figura 9 – GUI Cotar.	43
Figura 10 – GUI Histórico.	44
Figura 11 – GUI Histórico - expandido.	45
Figura 12 – Coleções do banco de dados.	48
Figura 13 – Algumas das lojas que possuem a carta “Niv-Mizzet, Parun”.	49
Figura 14 – Carta “Brontodonte Destruidor”.	49
Figura 15 – Funcionamento do <i>swap</i> .	52
Figura 16 – <i>Roulette wheel</i> .	53
Figura 17 – Uniforme.	53
Figura 18 – Quantidade.	53
Figura 19 – Fase 1 - Realocação.	56
Figura 20 – Resultado da realocação.	57
Figura 21 – Fase 2 - Eliminação.	57
Figura 22 – Resultado da eliminação.	58
Figura 23 – Atraxa.	60
Figura 24 – Niv-Mizzet.	60
Figura 25 – Atraxa.	61
Figura 26 – Niv-Mizzet.	61
Figura 27 – Ferramenta de cotação da LigaMagic.	62
Figura 28 – Atraxa.	63
Figura 29 – Niv-Mizzet.	63
Figura 30 – Storm.	63
Figura 31 – UB-Mill.	63
Figura 32 – Esper.	64
Figura 33 – Mono-Blue.	64

LISTA DE TABELAS

Tabela 1 – Configuração do computador pessoal.	35
Tabela 2 – Estrutura <i>content_table</i> .	51
Tabela 3 – Estrutura <i>result_table</i> .	52
Tabela 4 – Instâncias.	59
Tabela 5 – Análise de resultados.	64
Tabela 6 – <i>Deck Atraxa - Commander</i> .	71
Tabela 7 – <i>Deck Niv-Mizzet - Commander</i> .	72
Tabela 8 – <i>Deck Storm - Modern</i> .	72
Tabela 9 – <i>Deck UB-Mill - Modern</i> .	73
Tabela 10 – <i>Deck Esper - Standard</i> .	73
Tabela 11 – <i>Deck Mono-Blue - Standard</i> .	73

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
AJAX	<i>Asynchronous JavaScript and XML</i>
API	<i>Application Programming Interface</i>
CORS	<i>Cross Origin Resource Sharing</i>
CRUD	<i>Create, Read, Update and Delete</i>
CSS	<i>Cascading Style Sheets</i>
GUI	<i>Graphical User Interface</i>
JSON	<i>JavaScript Object Notation</i>
JWT	<i>JSON Web Tokens</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
NoSQL	<i>Not Only SQL</i>
PU	Processo Unificado
REST	<i>Representational State Transfer</i>
SGBD	Sistemas de Gestão de Base de Dados
SMTP	<i>Simple Mail Transfer Protocol</i>
SPA	<i>Single Page Application</i>
SQL	<i>Structured Query Language</i>
TCG	<i>Trading Card Game</i>
UML	<i>Unified Modeling Language</i>
XML	<i>Extensible Markup Language</i>
XPath	<i>XML Path Language</i>

LISTA DE SÍMBOLOS

α Fator de decaimento da temperatura

Δ Variação da função objetivo

\emptyset Conjunto vazio

λ Peso do objetivo

Λ Vetor de pesos dos objetivos

SUMÁRIO

1	INTRODUÇÃO	23
1.1	Justificativa	23
1.2	Objetivo	24
1.3	Estrutura do trabalho	24
2	FUNDAMENTAÇÃO TEÓRICA	27
2.1	Magic: The Gathering	27
2.1.1	Standard	27
2.1.2	Modern	27
2.1.3	Commander	27
2.2	Aplicações Web	28
2.2.1	SPA	28
2.2.2	REST	28
2.2.3	JWT	29
2.3	Banco de Dados	29
2.3.1	NoSQL	29
2.4	Web Crawler	30
2.5	Otimização Combinatória	30
2.5.1	Otimização Multiobjetivo	31
2.5.2	Fronteira de Pareto	31
2.6	Simulated Annealing	32
2.6.1	Temperatura	32
2.6.2	Função Objetivo	33
2.6.3	Critério de Aceitação	33
2.6.4	Cadeias de Markov	33
2.6.5	Simulated Annealing Multiobjetivo	34
3	MATERIAIS E MÉTODOS	35
3.1	Materiais	35
3.1.1	MongoDB	35
3.1.2	Python e suas bibliotecas	36
3.1.3	Vue	37
3.1.4	MTGJSON	37
3.2	Metodologia	37
4	PROJETO E DESENVOLVIMENTO	39

4.1	Modelagem	39
4.1.1	Casos de uso	39
4.1.2	Módulos	40
4.2	Interface	41
4.2.1	GUI Login	41
4.2.2	GUI Cadastro	42
4.2.3	GUI Perfil	42
4.2.4	GUI Cotar	43
4.2.5	GUI Histórico	44
4.3	Cliente	45
4.4	Servidor	46
4.5	Scrapy	48
4.6	Metaheurística	50
4.6.1	Estrutura de Dados	50
4.6.2	Solução Inicial	51
4.6.3	Geração de Vizinhança	52
4.6.4	Avaliação de Resultados	54
4.6.5	Produtor	54
4.6.6	Consumidor	55
4.7	Pós Otimização	56
5	RESULTADOS	59
5.1	Instâncias	59
5.2	Validação Simulated Annealing	60
5.3	Validação Pós Otimização	60
5.4	Resultados Finais	61
6	CONSIDERAÇÕES FINAIS	65
	REFERÊNCIAS	67
	APÊNDICE A – INSTÂNCIAS	71

1 INTRODUÇÃO

Magic: The Gathering, ou apenas *Magic*, é um *Trading Card Game* criado pela Wizards of the Coast em 1993. Em *Magic*, o jogador entra no papel de um *planeswalker*, um ser poderoso que utiliza *mana* para evocar criaturas, lançar feitiços e encantamentos para derrotar seu oponente. Usualmente, cada jogador inicia com 20 pontos de vida e um *deck*¹ de 60 cartas. O objetivo do jogador é reduzir a vida do oponente a zero para vencer ([OBELISK, 2017](#)).

O jogo fez tanto sucesso, que em 1994, surgiu o *Magic: The Gathering World Championship*, torneio realizado para jogadores que foram qualificados ao redor do mundo. Hoje, *Magic* possui mais de 12 milhões de jogadores espalhados pelo mundo na versão *tabletop*² e na versão digital ([WIZARDS, 2019](#)).

Atualmente *Magic* conta com mais de 20.000 cartas diferentes. Para que os jogadores tenham acesso a tantas opções, existem lojas especializadas no jogo espalhadas por todo o mundo. Essas lojas são um ponto de encontro para que os jogadores possam jogar, comprar, trocar e vender cartas.

No Brasil, a LigaMagic³ é o portal que disponibiliza várias ferramentas para os jogadores de *Magic*, como fóruns, notícias sobre o jogo e preços de cartas em lojas registradas. A LigaMagic conta com mais de 1000 lojas brasileiras em seu catálogo, permitindo aos seus usuários fazer o controle de cartas que desejam comprar e comparar valores entre lojas. ([LIGAMAGIC, 2019](#)).

Este trabalho tem como proposta o desenvolvimento de um protótipo de uma ferramenta *Web* de cotação de preços para *Magic: The Gathering* que auxilie na escolha de lojas. Esse sistema indica possíveis combinações de lojas, procurando encontrar o menor preço e a menor quantidade de lojas, visando economizar no frete de entrega dos produtos.

1.1 Justificativa

Em várias ocasiões do ano são lançadas novas coleções ao jogo, com o objetivo de estender o número de cartas, mecânicas e regras para diversificar a jogabilidade. Após o lançamento, não serão produzidas novas unidades de uma coleção. Por causa disso, as lojas não conseguem manter estoque de todas as cartas do jogo.

Este problema, acrescido de que nem toda cidade possui uma loja de *Magic*, tende

¹ Conjunto de cartas que formam o baralho.

² Jogos normalmente jogados sobre a mesa, como jogos de cartas, jogos de tabuleiros e jogos de dados.

³ URL da LigaMagic: <https://www.ligamagic.com.br>

a fazer com que os jogadores precisem procurar em lojas que não se localizam em suas cidades para encontrar as cartas que necessitam. Entretanto, comprar em lojas virtuais possui um ponto negativo, pois é necessário pagar um valor referente a entrega do produto para a transportadora.

A LigaMagic disponibiliza preços de cartas de todas as lojas virtuais registradas em sua plataforma. Mas, quando esse trabalho foi proposto, era dever do usuário encontrar a combinação de lojas que satisfizesse seu pedido. Levando em conta que um *deck* tem no mínimo 60 cartas, a chance do usuário conseguir encontrar a combinação que leva ao menor preço e a menor quantidade de lojas é pequena.

1.2 Objetivo

Este trabalho tem como objetivo o desenvolvimento de um protótipo de um sistema *Web* que realiza cotações de preço para cartas de *Magic: The Gathering*, buscando através de metaheurística, minimizar o valor final da compra, sugerindo combinações de lojas para o usuário. São objetivos secundários e mais específicos os seguintes tópicos:

- Implementar uma *Single Page Application*;
- Utilizar *web crawler* para extrair dados a partir do site LigaMagic que serão necessários para a cotação de preço;
- Gerar opções de compra utilizando a metaheurística *simulated annealing* e técnicas de pós-otimização, visando minimizar o preço final;
- Notificar o usuário via e-mail com os resultados da cotação;

1.3 Estrutura do trabalho

Este trabalho é composto por seis capítulos, sendo este o primeiro, onde fez-se as apresentações das ideias iniciais do projeto, com os objetivos e a justificativa.

No [Capítulo 2](#), é apresentado a fundamentação teórica sobre *Magic*, aplicações *web*, banco de dados, *web crawler*, otimização combinatória e a metaheurística *Simulated Annealing*.

O [Capítulo 3](#) apresenta os materiais utilizados para desenvolver a aplicação, assim como a metodologia.

No [Capítulo 4](#), apresenta a modelagem do sistema, a interface, a extração de dados, a metaheurística e a pós otimização.

O [Capítulo 5](#) apresenta os resultados encontrados neste trabalho.

Por fim, no [Capítulo 6](#) são apresentadas as considerações finais e discussões sobre este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os principais conceitos necessários ao desenvolvimento e compreensão deste trabalho, tais como as aplicações *web*, bancos de dados, *web crawlers* e metaheurísticas.

2.1 Magic: The Gathering

Magic utiliza um sistema de formatos para criar uma gama de estilos ao jogo. Cada formato possui suas próprias regras, alterando a jogabilidade em alguns aspectos, como expansões permitidas, número de cartas em um *deck*, pontos de vida, entre outros ([WIZARDS, 2019](#)). Nas próximas seções serão apresentados os três formatos utilizados para validar os resultados deste trabalho, note no entanto, que *Magic* possui vários outros formatos além dos aqui mencionados.

2.1.1 Standard

Também conhecido como formato padrão, o *standard* é um formato construído que consiste em utilizar apenas as expansões mais recentes do jogo. Anualmente, as quatro coleções mais antigas no formato param de ter validade e passam a valer as próximas coleções que forem introduzidas no jogo. Cada jogador inicia com 20 pontos de vida e um *deck* com no mínimo 60 cartas, mais 15 cartas de *sideboard*¹. Com exceção aos terrenos básicos, pode haver somente no máximo 4 cópias de uma mesma carta.

2.1.2 Modern

Formato construído que utiliza todas as expansões lançadas após a *Eighth Edition* de 2003. Possui as mesmas regras que o formato *standard*, com exceção de que nenhuma expansão já presente irá perder a validade no formato.

2.1.3 Commander

Commander é um formato construído voltado ao *multiplayer*. Todas as expansões do jogo são permitidas, começando pela *Limited Edition Alpha* de 1993. Esse formato possui uma criatura lendária para ser o comandante e mais 99 cartas sendo que, com exceção aos terrenos básicos, só é permitido ter uma cópia de cada carta. Cada jogador possui 40 pontos de vida e o último jogador a permanecer vivo vence o jogo.

¹ Cartas adicionais para fazer alterações durante uma partida no modo melhor de três.

2.2 Aplicações Web

Aplicação Web é um sistema desenvolvido para ser interpretado em um navegador de Internet, conectando-se através de um protocolo de comunicação a um servidor HTTP ([ARENDA, 2013](#)).

As aplicações *web* dividem-se em dois componentes: cliente e servidor ([DUARTE, 2015](#)). No lado do cliente são desenvolvidas páginas através da utilização de linguagens de programação como HTML, CSS e JavaScript. No servidor é desenvolvida a lógica de negócio, usando linguagens como PHP, NodeJs e .NET. A comunicação é feita utilizando o protocolo HTTP, que funciona da seguinte maneira: o cliente envia uma requisição HTTP para o servidor, que após ser processada, retorna uma nova página HTML com os resultados.

Para tornar a comunicação entre o cliente e servidor assíncrona, foi desenvolvido o AJAX. AJAX é um conjunto de técnicas de desenvolvimento voltado para a *web* que permite que aplicações trabalhem de modo assíncrono, processando qualquer requisição ao servidor em segundo plano. Com AJAX, o cliente pode enviar e receber dados, alterando dinamicamente conteúdos da página sem a necessidade de renderizar a página toda. Ele faz isso através de uma camada adicional do lado do cliente chamada motor AJAX, que pode realizar processamento sem a necessidade do servidor ([OLIVEIRA, 2017](#)).

2.2.1 SPA

Com AJAX, foi possível desenvolver o conceito de SPA, uma página *web* única que atualiza sua interface dinamicamente a medida que seus usuários interagem, sem a necessidade de recarregar todo o conteúdo. Esse tipo de aplicação foi ganhando espaço por prover uma experiência mais similar de aplicações *desktop* e por oferecer uma economia no uso da rede depois que a página tiver sido carregada, visto que a comunicação entre cliente e servidor agora envolve em sua maioria dados, e não uma interface completa ([OLIVEIRA, 2017](#)).

2.2.2 REST

Segundo ([FERREIRA, 2019](#)), REST é um modelo a ser utilizado para se projetar arquiteturas de *software* distribuído, baseadas em comunicação via rede.

Dados e funcionalidades da aplicação são considerados recursos. Um dos princípios do REST é que todo recurso deve possuir uma identificação única, sendo que um recurso é uma abstração sobre determinado tipo de informação que a aplicação gerencia. A partir de um recurso, é possível realizar ações de criar, retornar, alterar e excluir sobre ele. Estas ações são nomeadas de POST, GET, PUT e DELETE, respectivamente.

2.2.3 JWT

O JSON *Web Token* é um padrão aberto que define uma maneira compacta e independente de transmitir com segurança informações entre partes utilizando um objeto JSON. Essas informações podem ser verificadas e são confiáveis, porque são assinadas digitalmente ([JWT, 2019](#)).

Atualmente o JWT é mais utilizado em autorização. Após o *login* do usuário, cada solicitação subsequente incluirá o JWT, permitindo que o usuário acesse rotas, serviços e recursos permitidos com esse *token*.

2.3 Banco de Dados

Por volta de 1960 surgiram os primeiros SGBDs. Suas principais características são controle de concorrência, segurança, recuperação de falhas, gerenciamento dos mecanismos de armazenamento de dados e controle das restrições de integridade do banco de dados ([LÓSCIO; OLIVEIRA; PONTES, 2011](#)). SGBDs seguem a propriedade conhecida como ACID em suas transações.

[Date \(2004\)](#) afirma que desde então, surgiram diferentes modelos de banco de dados, sendo atualmente o banco de dados relacional uma das principais tecnologias de armazenamento. Este tipo de SGBD armazena os dados estruturados por meio de tabelas e relacionamentos entre essas tabelas. Para [Lóscio, Oliveira e Pontes \(2011\)](#), a simplicidade do modelo relacional contribuiu para sua grande disseminação e adoção. Porém, surgiu a necessidade de manipulação de outros formatos de dados, como imagem, som e vídeo, fazendo com que novas propostas de banco de dados aparecessem.

2.3.1 NoSQL

NoSQL surgiu com a chegada das aplicações *Web*, onde tornou-se essencial que as aplicações que fazem o gerenciamento desses dados tolerem falhas, possuam disponibilidade e escalabilidade, bem como a necessidade de manipulação de dados não estruturados ou semiestruturados ([SCAVUZZO; NITTO; CERI, 2014](#)). NoSQL vem do termo “não apenas SQL”. Esse termo referencia os SGBDs que não utilizam o modelo relacional e são mais flexíveis quanto as propriedades ACID.

Segundo [Lóscio, Oliveira e Pontes \(2011\)](#), os bancos de dados NoSQL normalmente atuam sem um esquema, permitindo que sejam adicionados novos campos sem a necessidade de definir quaisquer mudanças na estrutura do banco de dados. NoSQL pode ser classificado em quatro modelos de dados principais: chave-valor, colunar, documentos e grafos.

Os bancos de dados orientados a documentos utilizam documentos para armazenagem de dados. Documentos são muito flexíveis, já que não necessitam respeitar um

esquema, assim, cada documento pode compreender valores escalares, listas e até mesmo documentos aninhados. Esse modelo utiliza formatos comuns, como JSON, XML, PDF, entre outros (NAYAK; PORIYA; POOJARY, 2013). Este trabalho utiliza o banco de dados orientado a documentos MongoDB.

2.4 Web Crawler

Crawler, *bot* ou *spider* é um *software* para *downloads* de grandes quantidades de páginas *web*. Segundo Oslton e Najork (2010), *crawlers* são utilizados por uma variedade de propósitos. O mais relevante é o uso em motores de busca, que o usa para procurar e indexar páginas *web*, permitindo que usuários as encontrem utilizando palavras-chaves. Contudo, *crawlers* também são utilizados em técnicas de *data archiving*, no qual são feitos arquivamentos em massa de páginas *web* para usos futuros. Outra utilidade se dá em serviços de monitoramento, onde são definidas palavras-chaves, que quando encontradas notificam o usuário. Além disso, também temos *data mining*, em que as páginas *web* são vasculhadas a procura de dados para armazenar ou realizar análises estatísticas.

A partir de uma página *web* inicial, o *crawler* é capaz de automaticamente extrair URLs e recursivamente acessar novas páginas. Devido a sua natureza, o *crawler* consome a maior parte do tempo aguardando receber páginas *web* que requisitou. Por isso, normalmente *crawlers* são *multithread*, para poderem fazer requisições de muitas páginas *web* ao mesmo tempo (THELWALL, 2001).

2.5 Otimização Combinatória

Para Jünger, Reinelt e Thienel (1994), otimização combinatória lida com um tipo específico de otimização com a propriedade de que o conjunto das soluções factíveis são um conjunto finito. A otimização é feita para maximizar ou minimizar algum objetivo, e para isso é necessário ser possível caracterizar o conjunto das soluções factíveis e ter um algoritmo para avaliar a função objetivo de cada solução. Geralmente, a caracterização do conjunto de soluções factíveis podem ser subgrafos que satisfaçam determinada propriedade em um dado grafo. Já para avaliação da função objetivo, são estabelecidos valores de pesos nos elementos do conjunto.

Como o conjunto de soluções factíveis são finitos, os problemas de otimização combinatória poderiam ser resolvidos por enumeração. Entretanto, o conjunto de soluções pode ser tão grande que torna o método de enumeração geralmente impraticável. Por isso, vários problemas em otimização combinatória são resolvidos por outros métodos, como as metaheurísticas, que veremos adiante.

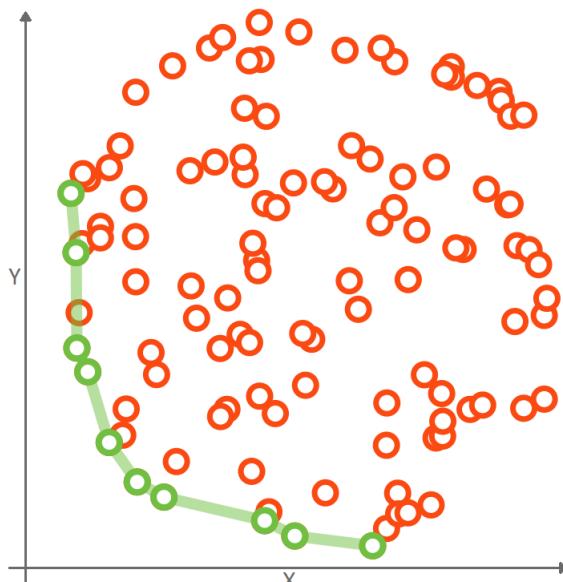
2.5.1 Otimização Multiobjetivo

Ao contrário das otimizações mono-objetivos, a otimização multiobjetivo não possui soluções ótimas no sentido de minimização ou maximização de cada objetivo individualmente. Em vez disso, temos um subconjunto de soluções aceitáveis que são superiores ao restante do conjunto. Tais soluções são denominadas como pareto-ótimas ou eficientes. A partir deste conjunto de soluções, e dependendo da natureza do problema, a escolha da solução será realizada por um *decision maker*² (ARROYO, 2002).

2.5.2 Fronteira de Pareto

Como vimos, quando os critérios para encontrar soluções são multiobjetivos, necessitamos de uma nova forma para encontrar as melhores soluções. Para isso, utilizamos a fronteira de pareto. Por definição uma solução $x^* \in S$ é pareto-ótima se, e somente se, não existe um $x \in S$, de modo que $f_i(x) \leq f_i(x^*)$ para todo $i = 1, \dots, k$ com $f_j(x) < f_j(x^*)$ para algum $j = 1, \dots, k$. Em outras palavras, uma solução x^* faz parte da fronteira de pareto se não for dominada por nenhuma outra solução e nenhum objetivo puder ser melhorado sem que outro objetivo seja piorado (BAYKASOGLU; OWEN; GINDY, 1999). Na Figura 1 podemos ver em verde as soluções do conjunto S que fazem parte da fronteira de pareto em um problema de minimização.

Figura 1 – Fronteira de Pareto.



Fonte – Ormont (2016)

² Pessoa ou ferramenta que decidirá qual solução do conjunto de soluções será utilizada.

2.6 Simulated Annealing

Annealing é uma técnica metalúrgica de tratamento térmico que altera as propriedades físicas de um material para aumentar sua resistência. O processo consiste em aquecer o material até que atinja sua forma líquida e, em seguida, lentamente reduzir a temperatura. Desse modo, as partículas do material irão se rearranjar buscando formas estruturais mais estáveis. A metaheurística *simulated annealing* segue essa ideia para resolver problemas combinatórios. Os principais conceitos do *simulated annealing* são a temperatura, função objetivo, critério de aceitação e cadeias de Markov ([LAARHOVEN; AARTS, 1987](#)). O [algoritmo 1](#) apresenta o funcionamento do *simulated annealing*.

Algoritmo 1: Simulated Annealing Mono-objetivo

```

Gere uma solução inicial x
M := x
C := Número de cadeias de Markov
repeat
    T := Temperatura inicial
    repeat
        Gere uma solução y a partir da vizinhança de x
        if y é melhor que M then
            M = y
        end if
        x := y com probabilidade  $P(x, y, T)$ 
        Diminui T utilizando algum esquema de resfriamento
    until T = Temperatura final
    C := C - 1
until C = 0

```

M é a melhor solução encontrada. C é o numero de cadeias de Markov executadas e T é a temperatura atual do sistema. $P(x, y, T)$ é o critério de aceitação. Nas próximas seções esses tópicos vão ser melhor abordados, explicando seu funcionamento no *simulated annealing*.

2.6.1 Temperatura

A temperatura é um elemento crucial no *simulated annealing*, pois ela define se soluções ruins poderão ser aceitas, com o objetivo de expandir o espaço de busca. Em momentos que a temperatura está alta, existe uma probabilidade maior de se aceitar uma solução ruim, já com a temperatura baixa, a probabilidade é menor.

O *simulated annealing* inicia com uma temperatura alta, que vai decrescendo durante o processo tendendo a zero, esse processo se chama esquema de resfriamento, ou *cooling schedule* ([HAJEK, 1988](#)). A [Equação 2.1](#) e [Equação 2.2](#) mostra exemplos de dois

esquemas de resfriamento, o geométrico e o logaritmo, respectivamente.

$$T = \alpha T \quad (2.1)$$

$$T = T_i / \log(1 + T) \quad (2.2)$$

Onde T é a temperatura atual do sistema, α é um valor entre 0,0 e 1,0 e T_i é a temperatura inicial do sistema.

2.6.2 Função Objetivo

Segundo Kirkpatrick, Gelatt e Vecchi (1983) a função objetivo é o que representa a medida quantitativa de uma solução. Por exemplo, no problema do caixeiro-viajante, onde existe um conjunto de cidades e o objetivo é passar por todas sem repetir cidades com o menor caminho, a função objetivo seria a distância percorrida pelo viajante. Com base na função objetivo, é possível comparar duas soluções para escolher a melhor.

2.6.3 Critério de Aceitação

O critério de aceitação utilizado no *simulated annealing* é nomeado como critério de Boltzman. Ele define que qualquer solução que tenha uma melhora em relação a outra deve ser aceita, mas, em caso negativo, a solução será submetida a Equação 2.3 para ser aceita ou não.

$$P(\text{novoestado}) = e^{-\Delta/T} \quad (2.3)$$

Note que Δ é a diferença entre a função objetivo da solução atual com a solução anterior. T é a temperatura atual do sistema. Essa equação dará como resultado um valor entre 0,0 e 1,0 que será comparado com um número gerado aleatoriamente. Quanto maior a temperatura, maior será o valor de P e mais chance a solução atual terá para ser aceita (LAARHOVEN; AARTS, 1987).

2.6.4 Cadeias de Markov

Cadeia de Markov é definida por Laarhoven e Aarts (1987) como “uma sequência de tentativas, em que o resultado de cada tentativa depende somente do resultado anterior”. Cada cadeia de Markov possui um valor de temperatura inicial fixo, que quando a temperatura do sistema chegar ao valor final, uma nova cadeia de Markov será acionada. Desse modo, com a temperatura alta novamente, acontecem novas iterações que permitem

que possíveis soluções piores que a atual sejam aceitas, fazendo com que o espaço de busca seja melhor explorado.

2.6.5 Simulated Annealing Multiobjetivo

Czyzak e Jaszkiewicz (1998) apresentam em seu artigo um método baseado no *simulated annealing* original para problemas multiobjetivos. No lugar de uma melhor solução, temos um conjunto de potenciais soluções eficientes, além de uma nova equação para o critério de aceitação. O [algoritmo 2](#) descreve o *simulated annealing* multiobjetivo em pseudocódigo.

Algoritmo 2: Simulated Annealing Multiobjetivo

```

Gere uma solução inicial x
M := Ø
Atualize o conjunto M das potenciais soluções eficientes com x
T := Temperatura inicial
repeat
    Gere uma solução y a partir da vizinhança de x
    if y não é dominado por x then
        Atualize o conjunto M das potenciais soluções eficientes com y
        x := y com probabilidade  $P(x, y, T, \Lambda)$ 
    end if
    Diminui T utilizando algum esquema de resfriamento
until T = Temperatura final

```

M é o conjunto das potenciais soluções eficientes. A partir do conjunto M a fronteira de Pareto será calculada, resultando as soluções que não forem dominadas. T é a temperatura atual do sistema. $P(x, y, T, \Lambda)$ é o resultado do critério de aceitação que representa a probabilidade de x receber a solução y , Λ é um vetor de pesos $[\lambda_1, \dots, \lambda_i]$ de cada objetivo.

O critério de aceitação utilizado pelo *simulated annealing* multiobjetivo define um peso λ não negativo para cada objetivo, sendo que $\sum_{i=1}^n \lambda_i = 1$, onde n é o número de objetivos. A [Equação 2.4](#) nomeada de *Rule SL* pode ser vista como uma agregação local de todos os objetivos com a soma ponderada dos pesos (CZYZAK; JASZKIEWICZ, 1998).

$$P(x, y, T, \Lambda) = \min \left\{ 1, \exp \left(\sum_{j=1}^J \lambda_j (f_j(x) - f_j(y)/T) \right) \right\} \quad (2.4)$$

3 MATERIAIS E MÉTODOS

3.1 Materiais

Para a elaboração do sistema utilizou-se um computador pessoal cuja especificações se encontram na [Tabela 1](#).

Tabela 1 – Configuração do computador pessoal.

Processador Intel Core i7-3630QM 2,4 GHz
Memória RAM 8GB
Armazenamento SSD 480GB
Processador Gráfico GeForce GT 630M
Sistema operacional Linux Mint 19.1 Cinnamon

Fonte – Elaborado pelo autor.

Os principais *softwares* para compor o ambiente de programação do projeto foram:

- Visual Studio Code¹ versão 1.38;
- NoSQLBooster² versão 5.2.2;
- DIA³ versão 0.97.2.

3.1.1 MongoDB

Para a persistência de dados, foi utilizado o banco de dados orientado a documentos MongoDB. Esse SGBD faz parte dos bancos de dados NoSQL e possui código aberto. MongoDB foi lançado em 2009 e fornece recursos sofisticados como alta disponibilidade, baixo custo de manutenção e operação, além de flexibilidade para lidar com dados ([MONGODB, 2019](#)).

MongoDB disponibiliza *drivers* para mais de dez linguagens, como JavaScript, Python, Java, C++, entre outros. Ele utiliza um formato de armazenamento semelhante a JSON, o que significa flexibilidade, já que cada documento pode ter campos diferentes, não necessitando seguir a mesma estrutura.

¹ URL do Visual Studio Code: <https://code.visualstudio.com/>

² URL do NoSQLBooster: <https://nosqlbooster.com/>

³ URL do DIA: <http://dia-installer.de/>

3.1.2 Python e suas bibliotecas

Python (versão 3.6.9) foi a linguagem escolhida para implementação do servidor. Essa linguagem foi criada em 1989 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação ([PILGRIM, 2008](#)). A linguagem Python foi utilizada nesse trabalho por ser eficiente, de fácil aprendizado, com facilidades nativas para manipulação de dados e arquivos, além de sua grande variedade de bibliotecas gratuitas, descritas a seguir.

- Flask versão 1.1.1 - Flask é um micro-*framework web*. Foi desenvolvido com o objetivo de ser leve, fácil e rápido para aprender, com a possibilidade de se criar aplicações complexas. Ele foi utilizado para criação de rotas de comunicação com a aplicação *web* e o banco de dados;
- Flask-CORS versão 3.0.8 - Habilita CORS no Flask. CORS permite que uma página *web* acesse recursos de outra página ou servidor que estão em domínios diferentes.
- Flask-JTW versão 3.22.0 - Biblioteca para gerenciar autorização e comunicação entre o cliente e servidor utilizando JWT no Flask;
- PyMongo versão 3.9.0 - Biblioteca utilizada para conectar, realizar consultas e inserções ao banco de dados MongoDB;
- Scrapy versão 1.7.3 - Scrapy é um *framework open source* utilizado para rastreamento e extração de dados estruturados de páginas *web*. Ele foi utilizado na extração de cartas dos sites de venda de *Magic*;
- Multiprocessing versão 2.6.2.1 - Python possui um mecanismo que o protege contra conflitos entre *threads*, ele faz isso definindo que todas as *threads* executarão em um único núcleo do processador. Devido a isso, a biblioteca *Multiprocessing* foi elaborada para executar procedimentos com multiprocessamento. Ela utiliza uma API similar a do Python *Threading*. Essa biblioteca foi utilizada na metaheurística;
- Numpy versão 1.17.1 - Numpy é o pacote fundamental para propósitos científicos e matemáticos em Python. Seu principal uso está em realizar cálculos em matrizes multidimensionais;
- smtplib - SMTP é um protocolo que lida com o envio de e-mail. A biblioteca *smtplib* é utilizada para criar e enviar e-mails em Python. Foi utilizada nesse trabalho para notificar o usuário quando o resultado da cotação estiver disponível;
- Matplotlib versão 3.1.1 - É uma biblioteca para renderização de gráficos. Foi utilizada para gerar gráficos dos resultados encontrados nesse trabalho.

3.1.3 Vue

Vue (pronunciado como View) foi o *framework* escolhido para a implementação do cliente. Segundo Silva e Souza (2017), Vue é um *framework* progressivo do JavaScript de código aberto para o desenvolvimento de componentes reativos para interfaces *web* modernas. Vue foi escolhido para este trabalho por ser uma tecnologia que vem crescendo e por ter uma grande facilidade de aprendizado.

- Vue-router - Responsável por mostrar ou esconder elementos dependendo da URL que se acessa no *browser*, já que toda aplicação está contida em uma só página.
- Vue-axios - É uma biblioteca para comunicação HTTP, utilizada para fazer solicitações AJAX ao servidor.
- Bootstrap-vue - Bootstrap é um *framework* de CSS para desenvolvimento de aplicações responsivas⁴ e para *mobile*. Bootstrap-vue é a versão do Bootstrap para usar com Vue

3.1.4 MTGJSON

Segundo a descrição dos criadores “MTGJSON⁵ é um projeto *open source* que cataloga todas as cartas de *Magic: The Gathering* em um formato portável” (MTGJSON, 2019). O projeto disponibiliza um arquivo JSON com todas as informações necessárias sobre as cartas. MTGJSON foi utilizado neste trabalho para criar um banco de dados onde é possível verificar se as cartas passadas pelo usuário estão com os nomes corretos, nos idiomas inglês ou português.

3.2 Metodologia

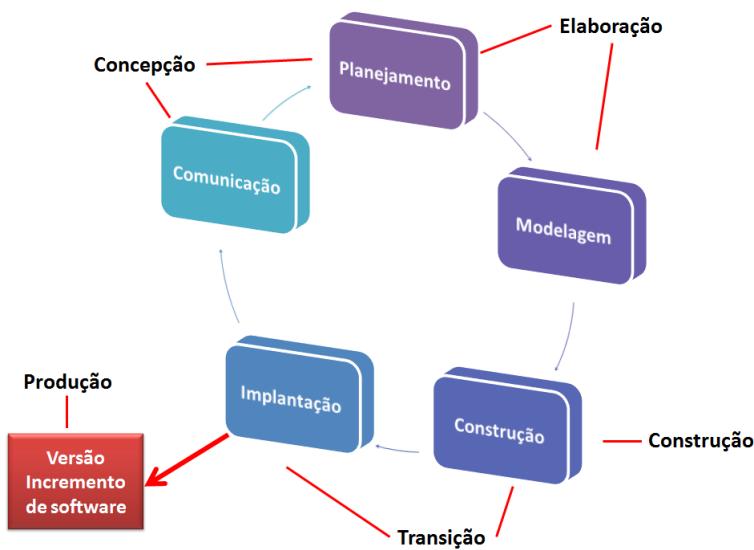
A metodologia seguida neste trabalho foi inspirada no Processo Unificado. Trata-se de um modelo iterativo e incremental, oferecendo uma abordagem para partitionar o trabalho em porções menores ou mini-projetos. O PU utiliza UML como linguagem de modelagem. O ciclo de desenvolvimento é composto por várias iterações das seguintes fases: Concepção, Elaboração, Construção e Transição. A Figura 2 mostra o ciclo de vida do PU. Cada ciclo gera uma versão do sistema que incrementa funcionalidades na versão anterior. (FALBO, 2000)

No processo de Concepção é definido o escopo do sistema, o esboço da arquitetura, a previsão dos custos e do cronograma. A Elaboração consiste em analisar mais cuidadosamente os requisitos funcionais da aplicação que não foram capturados na fase anterior. A

⁴ Página *web* que tem o *layout* preparado para se adaptar ao formato de diferentes tamanhos de tela.

⁵ URL do MTGJSON: <https://mtgjson.com/>

Figura 2 – Fases do Processo Unificado.



Fonte – Pressman e Maxim (2016)

fase de Construção é onde o produto é construído de maneira iterativa e incremental. Já a Transição é onde o produto tem a primeira versão entregue ao cliente para teste.

Este trabalho foi realizado em quatro iterações do PU. Na primeira foi desenvolvido o *web crawler*, tendo seus dados salvos em um sistema de arquivo. Na segunda entrega foi finalizado a metaheurística e a pós-otimização, com os dados ainda sendo salvos em arquivos. Em seguida, foi estruturado o *web service* da aplicação e a migração do sistema de arquivo para o banco de dados. Na última iteração, foi desenvolvida a página *web*.

4 PROJETO E DESENVOLVIMENTO

Neste capítulo é descrito a modelagem do projeto - definição das funcionalidades da aplicação, módulos que compõem a aplicação e como cada material descrito no capítulo anterior foi utilizado. Depois de definida a abordagem, foi realizado o desenvolvimento da aplicação.

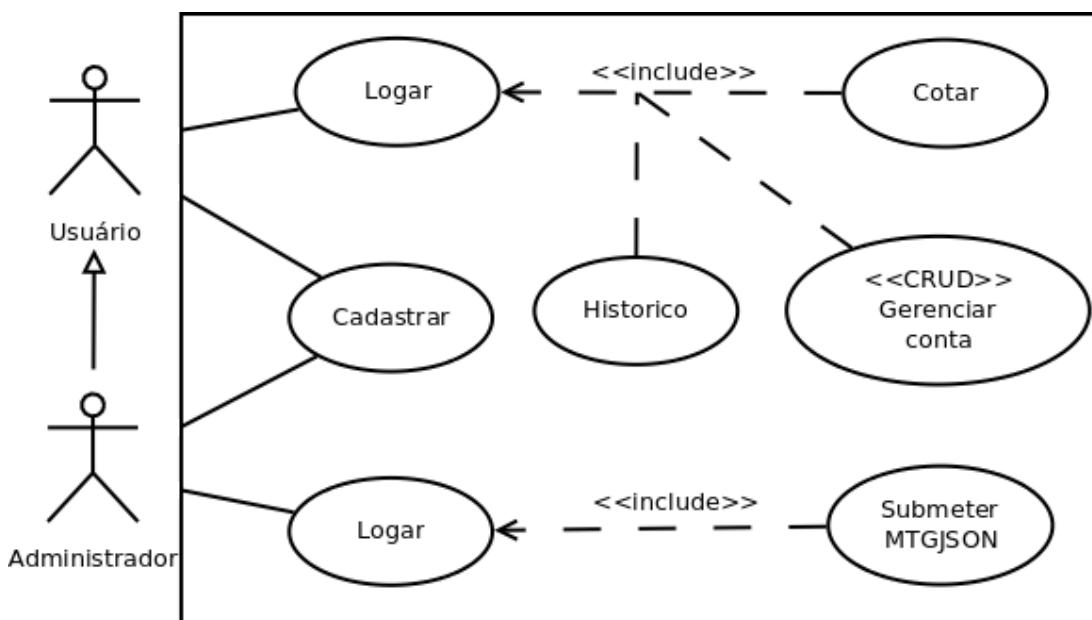
4.1 Modelagem

Para o desenvolvimento de um sistema é preciso primeiramente realizar uma modelagem de forma a atender o problema real a qual a aplicação se destina. Para isso, inicialmente deve ser formulado como os usuários irão interagir com o sistema e definir a estrutura geral do projeto. Nas próximas seções será apresentado o diagrama de caso de uso e os módulos principais do sistema.

4.1.1 Casos de uso

Diagramas de caso de uso organizam os comportamentos do sistema. Eles são utilizados para identificar como cada um dos elementos do sistema irá se comportar, mostrando uma visão externa de como os elementos serão utilizados no contexto da aplicação (BOOCHE; RUMBAUGH; JACOBSON, 2006).

Figura 3 – Diagrama de caso de uso.



Fonte – Elaborado pelo autor.

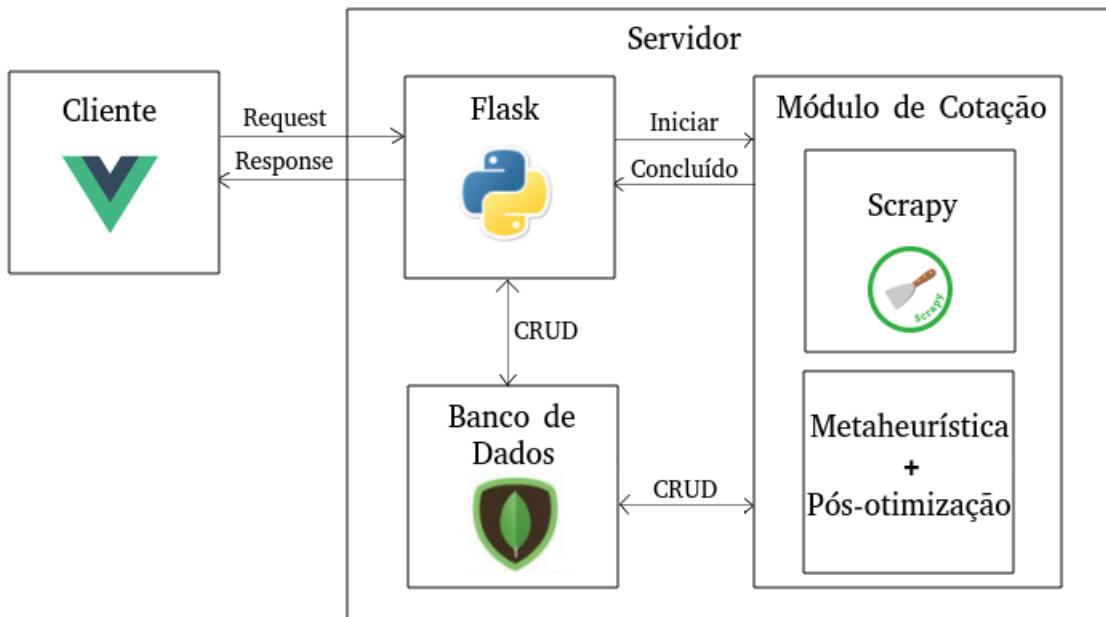
Na [Figura 3](#) é apresentado o diagrama de caso de uso deste trabalho. Foram definidos dois atores, usuário e administrador, que realizam interações com o sistema.

Ambos os atores primeiramente precisam fazer o cadastro e fazer *login* para ter acesso ao sistema. Ao usuário está disponível a interface de cotação de preço, histórico de cotação e gerenciamento de conta. O administrador herda todos os recursos do usuário, com o adicional de ser o responsável por submeter o arquivo MTGJSON ao banco de dados, que, como visto na [subseção 3.1.4](#), é responsável por manter o nome de todas as cartas de *Magic*.

4.1.2 Módulos

A [Figura 4](#) apresenta a divisão dos módulos deste trabalho. A aplicação é dividida em dois grandes módulos, o cliente e o servidor.

Figura 4 – Módulos da aplicação.



Fonte – Elaborado pelo autor.

O cliente é responsável pela interação com usuário, a partir dele o usuário irá enviar requisições e receber respostas do servidor. Na [seção 4.2](#) serão descritos com mais detalhes as interfaces que compõem o cliente e na [seção 4.3](#) serão descritos as etapas de desenvolvimento do cliente.

O servidor é responsável por todo processamento da aplicação. De modo geral, o módulo Flask estabelece rotas de comunicação HTTP com o cliente. A partir dessas rotas, o cliente está apto para enviar requisições ao servidor, como cadastrar, logar ou submeter listas para cotação. Após receber uma requisição de cotação, o servidor realiza a validação

dos dados, armazena no banco de dados e informa ao módulo de cotação que uma nova requisição está em fila. Após o término da requisição, o servidor envia uma notificação por e-mail ao usuário informando-o. Entre a [seção 4.4](#) até a [seção 4.7](#) serão descritos o desenvolvimento dos componentes do servidor.

A partir de uma requisição, o módulo de cotação inicia a ferramenta Scrapy, que por sua vez, extrai os dados das cartas contidas na requisição e salva no banco de dados. Após isso, o módulo da metaheurística inicia a filtragem dos dados, procurando encontrar opções de compras que minimizem o preço e o número de lojas. Por fim, o resultado encontrado passa por um processo de pós otimização baseado no método guloso para tentar melhorar o resultado final. O resultado é salvo no banco de dados e o servidor é notificado.

4.2 Interface

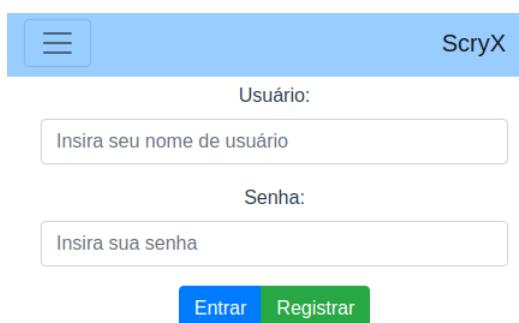
A aplicação recebeu o nome de ScryX. Esse nome surgiu a partir do componente *scry* (vidência) utilizado em *Magic* para que os jogadores possam olhar as cartas do topo do *deck*, podendo organizar ou mandar para o fundo, com o objetivo de controlar as cartas que virão nos próximos turnos.

A seguir serão apresentadas as interfaces do sistema.

4.2.1 GUI Login

A [Figura 5](#) apresenta a interface de *login* do sistema, a partir dessa interface, é possível logar ou cadastrar um usuário. Os campos requeridos são nome de usuário e senha. Caso houver algum erro de autenticação, um alerta de usuário ou senha incorretos é adicionado ao topo da página. Após realizar a autenticação, o usuário é redirecionado para a página inicial e adquire acesso a interface de Cotar e Histórico.

Figura 5 – GUI Login.



Fonte – Elaborado pelo autor.

4.2.2 GUI Cadastro

A Figura 6 apresenta os campos necessários para o cadastro: nome, e-mail, data de nascimento, nome de usuário e senha. Todos os campos devem estar preenchidos e serem válidos, caso contrário, um alerta no topo da página será acionado informando os campos incorretos.

Figura 6 – GUI Cadastro.

The screenshot shows a registration form titled "Cadastro". It contains the following fields:

- Nome:** Input field with placeholder "Insira seu nome".
- E-mail:** Input field with placeholder "Insira seu e-mail".
- Data de Nascimento:** Input field with placeholder "dd/mm/aaaa".
- Usuário:** Input field with placeholder "Insira um nome de usuário".
- Senha:** Input field with placeholder "Insira sua senha".
- Confirme Senha:** Input field with placeholder "Confirme sua senha".

At the bottom are two buttons: "Registrar" (blue) and "Limpar" (red).

Fonte – Elaborado pelo autor.

4.2.3 GUI Perfil

Figura 7 – GUI Perfil - Usuário.

The screenshot shows a user profile page titled "Perfil". It displays the following information:

Nome:	José Andrades	Data de Cadastro:	2019-10-04
Usuário:	jandrades	Data de Nascimento:	1990-01-01
E-mail:	joseandrades1@gmail.com	Alterar Senha	

At the bottom are three buttons: "Alterar E-mail" (blue), "Alterar Senha" (blue), and "Apagar Conta" (red).

Fonte – Elaborado pelo autor.

A Figura 7 apresenta a interface do perfil de usuário. Nessa interface, o usuário pode ver seus dados, alterar e-mail e senha. Também é possível a partir dessa interface apagar a conta. Ao acessar essa opção, aparecerá um alerta informando que essa opção é permanente e um botão de confirmação.

A interface do administrador, Figura 8, possui os mesmos componentes da interface de um usuário normal, com exceção da parte superior, onde ele possui um campo para submeter um arquivo MTGJSON para atualizar os nomes das cartas no banco de dados.

Figura 8 – GUI Perfil - Administrador.

The screenshot shows the ScryX Admin Profile interface. At the top, there's a blue header bar with a menu icon and the text "ScryX". Below it, a sub-header says "Submeter lista de nomes de cartas - MTGJson" with a "Browse" button and a "Enviar" button. The main area is titled "Perfil:" and contains user data: Nome: admin, Usuário: admin, E-mail: admin@admin. To the right, it shows Data de Cadastro: 2019-09-17 and Data de Nascimento: 2000-01-01. There are three buttons at the bottom: "Alterar Senha" (blue), "Alterar E-mail" (blue), and "Apagar Conta" (red).

Fonte – Elaborado pelo autor.

4.2.4 GUI Cotar

Figura 9 – GUI Cotar.

The screenshot shows the ScryX Cotar interface. At the top, there's a blue header bar with a menu icon and the text "ScryX". Below it, a sub-header says "Cotar:". There are two input fields: one for "Digite o nome do deck" and another for "Insira a lista de cartas. Ex: 4 Fog Bank". At the bottom, there's a "Enviar" button.

Fonte – Elaborado pelo autor.

A Figura 9 apresenta a interface de cotação de cartas. O primeiro campo é o nome que irá aparecer como resultado na interface Histórico. O segundo campo é onde deve

ser adicionado a lista com as cartas, seguindo o formato: “quantidade” “nome da carta”. Se houver alguma carta fora do padrão especificado, um alerta será acionado no topo da página informando qual ou quais cartas estão fora do padrão.

Após enviar a lista, o servidor checará se todos os nomes das cartas passadas existem na base de dados. Caso contrário, um alerta irá informar os nomes das cartas inexistentes. Caso tudo esteja correto, um alerta informará o usuário que a requisição está em andamento e vai demorar alguns minutos. Quando a requisição acabar, o usuário será informado via e-mail.

4.2.5 GUI Histórico

A interface Histórico mostra todos os resultados de cotações que o usuário submeteu. A Figura 10 mostra o exemplo de três listas. Ao selecionar uma lista, o componente se expande, apresentando as opções de compra. O motivo de existir mais de uma opção será apresentado na [seção 4.6](#).

Figura 10 – GUI Histórico.



Fonte – Elaborado pelo autor.

Ao selecionar uma opção, uma tabela de resultados é mostrada. Podemos ver um exemplo de resultado na Figura 11. No cabeçalho, em amarelo, está o valor total da compra e a quantidade de lojas nesta opção. Logo abaixo, em azul, estão as informações específicas de cada loja, o nome, o valor total e a lista completa de todas as cartas que devem ser compradas naquela loja. Em caso de alguma carta não ser encontrada, ela aparecerá em uma tabela em vermelho no fundo da opção.

Figura 11 – GUI Histórico - expandido.

Opção 1			
Valor Total:	1435.49	Quantidade de lojas:	3
Loja:	Magicbembarato	Valor:	503.78
Nome	Quantidade	Preço Unitário	Preço
absorb	2	5.26	10.52
cast down	3	2.69	8.07
ethereal absolution	1	1.59	1.59
glacial fortress	1	19.27	19.27
lyra dawnbringer	2	21.56	43.12
negate	2	0.41	0.82
teferi, hero of dominaria	4	102.09	408.36
thief of sanity	1	12.03	12.03
Loja:	Pi Games	Valor:	512.06
Nome	Quantidade	Preço Unitário	Preço
donuts of destruction	2	16.00	32.00

Fonte – Elaborado pelo autor.

4.3 Cliente

No desenvolvimento do cliente foi utilizado o Vue, *framework* do JavaScript. Vue traz um conceito chamado componente. Componentes são blocos reutilizáveis de código que podem conter estruturas e funcionalidades. Utilizar componentes faz com que o código seja mais modularizado, fazendo com que no lugar de um código gigante, temos pequenos módulos se interligando. Podemos ver exemplos de componentes na [Figura 5](#). Temos o componente principal que abrange toda a página e temos dentro dele os componentes cabeçalho e *login*.

A aplicação é uma SPA, o que significa que todo o conteúdo estará em uma única página. Devido a isso, é preciso gerenciar quais componentes serão mostrados na tela, e quais estarão indisponíveis. Para fazer esse gerenciamento, foi utilizado o Vue-router.

O Vue-router vincula URLs em componentes. Sempre que uma URL existente for acionada, o componente correspondente toma lugar do componente atualmente em exibição. O código mostra um trecho onde é definido que o componente *Home* é vinculado ao caminho '/' e o componente *login* é vinculado ao caminho '/login'.

```

1 const routes = [
2   { path: '/', component: Home, },
3   { path: '/login', component: Login, }
4 ]

```

Listing 4.1 – Router.

O Bootstrap-vue foi utilizado para estilizar os componentes sem a necessidade de desenvolver CSS. Além disso, pelo uso dos componentes disponibilizados por este *framework*, a aplicação se tornou responsiva, podendo ser redimensionada para telas grandes e pequenas sem a necessidade de programar essas ações. Todo componente que inicia com a letra com “b” faz parte dos componentes do Bootstrap-vue, como demonstrado no código a seguir.

```

1 <b-form-input type="text" v-model="name" required placeholder="Digite o
   nome do deck"></b-form-input>
2 <b-form-textarea id="textarea" v-model="card_list" placeholder="Insira a
   lista de cartas" rows="10" required></b-form-textarea>
3 <b-button type="submit" variant="primary" v-on:click=submitList()>Enviar
   </b-button>
```

Listing 4.2 – Entrada de dados de cotação.

Para realizar requisições ao servidor, foi utilizado o Vue-axios. Essa ferramenta permite consumir APIs REST do servidor. Cada requisição tem um tipo de ação definido, *GET*, *POST*, *PUT* e *DELETE*, que indica como o servidor irá responder. O código a seguir mostra o exemplo de uma requisição do tipo *POST* de criação de usuário. O axios utiliza a rota “/create_user” para enviar os dados de cadastro ao servidor. Em seguida, recebe uma resposta em formato JSON. A partir daí, o cliente identifica se a requisição teve sucesso ou não, e notifica o usuário.

```

1 axios.post(path + '/create_user', dados_usuario)
2   .then((response) => {
3     if (response.data.status == '0') {
4       this.message = 'Cadastro realizado com sucesso!'
5     } else {
6       this.message = 'Usuario indisponivel!'
7     }
8   });
```

Listing 4.3 – Requisição axios para cadastro.

A maioria das funcionalidades do cliente exige autenticação. Quando o usuário realiza o *login*, o servidor retorna um *token* JWT de acesso. Em toda requisição que exige autenticação, é enviado o *token* pelo axios. Se o servidor não reconhecer autenticidade no *token* do usuário, ele devolve uma resposta de acesso negado.

4.4 Servidor

O servidor foi dividido em três módulos. Flask, módulo de cotação e Banco de dados.

De modo geral, o servidor é responsável pelas rotas que possibilitam que o cliente envie requisições, pelo processamento dessas requisições e armazenamento de dados no banco de dados. Para que o servidor esteja habilitado a receber requisições do cliente, foi utilizado a biblioteca Flask-CORS.

O código a seguir é um exemplo do cabeçalho da função de criação de usuário. O “@app.route” define rotas que o cliente poderá utilizar para fazer requisições. No caso deste exemplo, o cliente deve enviar uma requisição axios contendo os dados do usuário e utilizar o método *POST* para o “endereço_servidor/create_user”. O servidor retorna uma resposta em JSON constando se a requisição obteve sucesso ou não.

```
1 @app.route("/create_user", methods=['POST'])
2 def create_user():
3     dados = request.get_json()
4     ...
```

Listing 4.4 – Rota de criação de usuário.

O Flask utiliza a biblioteca Flask-JWT para fazer o controle da autenticação de usuário. Após receber uma requisição de *login*, o Flask armazena e retorna um *token* ao cliente. Toda vez que um usuário envia uma requisição que exige autenticação, o Flask verifica se aquele *token* associado ao usuário está registrado. Os *tokens* possuem um tempo de uso até que expire, após isso, um novo *login* é necessário. O código a seguir é a rota para requisição do histórico de cotações. O servidor recusará a requisição se não receber um *token* válido.

```
1 @app.route("/history/<username>", methods=['GET'])
2 @jwt_required
3 def history(username):
4     ...
```

Listing 4.5 – Rota de acesso ao histórico.

No servidor, existe o módulo de cotação que funciona de maneira independente do Flask. Ele é responsável por inicializar as ferramentas de cotação. Quando chega uma requisição, ela é inserida na fila de cotações. Primeiramente é inicializado o Scrapy, abordado na [seção 4.5](#). Depois é iniciado o *Simulated Annealing* e a pós-otimização, abordados na [seção 4.6](#) e [seção 4.7](#) respectivamente.

Após o fim da cotação, é utilizado o protocolo SMTP para enviar um e-mail com o nome do *deck* para notificar o usuário que seu resultado está disponível para consulta. A biblioteca smtplib recebe como entrada um endereço SMTP do e-mail a ser utilizado, as informações de usuário e senha do remetente, o e-mail do destinatário e a mensagem a ser enviada.

A comunicação com o módulo do banco de dados é feita através do PyMongo. No código a seguir, a linha 1 conecta a aplicação ao MongoDB. A linha 2 representa a conexão

com o banco da aplicação, “scryx”. A linha 3 representa um exemplo de inserção do nome de uma carta na coleção “*card*”. A partir da variável “db” é possível realizar as operações de CRUD utilizando qualquer coleção do banco.

```
1 client = MongoClient('localhost', 27017)
2 db = client["scryx"]
3 db.card.insert_one({'_id': 'Talent of the Telepath', 'portuguese': 'Talento do Telepata'}
```

Listing 4.6 – PyMongo.

A [Figura 12](#) representa as coleções que compõem o banco de dados. “*card*” é a coleção que possui o nome de todas as cartas em português e inglês. “*queue*” é a coleção que armazena os identificadores das requisições de cotação em fila, a partir dele, o módulo de cotação sabe qual requisição processar. “*request*” guarda os dados da requisição de cotação. Ele possui o identificador da requisição e do usuário, o nome do *deck*, os nomes das cartas e os dados extraídos pelo Scrapy. A coleção “*result*” armazena resultados de cotações. E por último, a coleção “*user*”, que armazena os dados dos usuários.

Figura 12 – Coleções do banco de dados.



Fonte – Elaborado pelo autor.

4.5 Scrapy

Para realizar a cotação, é necessário ter os dados das cartas, que são: nome, loja, quantidade e valor. Esses dados estão disponíveis na página da LigaMagic. Infelizmente não é disponibilizado uma API para que seja possível extrair esses dados. Para resolver este problema foi utilizado o *framework* Scrapy. De modo geral, o Scrapy recebe uma página raiz e expressões regulares para percorrer e extrair dados específicos da página.

Inicialmente, foi necessário entender como a LigaMagic organiza seus dados. Foi descoberto que as URLs referentes as cartas seguem um padrão de “url/nome_carta”. A [Figura 13](#) é um trecho da interface que mostra os valores de uma carta. A partir do HTML que compõe essa interface, deveria ser possível recolher todos os dados necessários. Mas, após uma observação mais precisa, foi descoberto que cada dígito que faz parte do valor da carta, é na verdade um pedaço de uma imagem que contém vários outros dígitos, sendo que essa imagem é gerada aleatoriamente cada vez que a página é acessada.

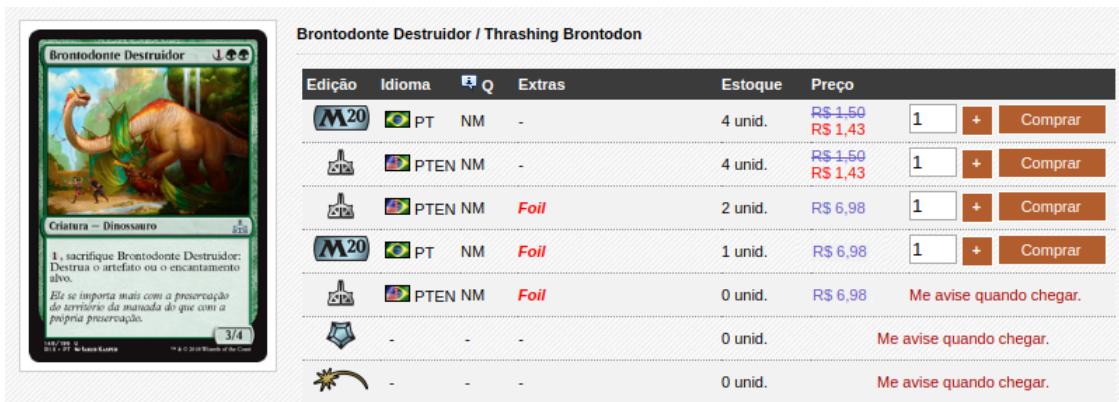
Figura 13 – Algumas das lojas que possuem a carta “Niv-Mizzet, Parun”.

		R\$ 6,50 R\$ 5,20		4 unids	<input type="button" value="1"/>	<input type="button" value="Comprar"/>	<input type="button" value="Ir à loja"/>
		R\$ 5,49		48 unids	<input type="button" value="1"/>	<input type="button" value="Comprar"/>	<input type="button" value="Ir à loja"/>
		R\$ 6,00 R\$ 5,70		2 unids	<input type="button" value="1"/>	<input type="button" value="Comprar"/>	<input type="button" value="Ir à loja"/>
		R\$ 5,70		6 unids	<input type="button" value="1"/>	<input type="button" value="Comprar"/>	<input type="button" value="Ir à loja"/>

Fonte – Ligamagic (2019)

Devido a isso, outra abordagem teve de ser tomada. No lugar de usar a página da LigaMagic para extrair os dados das cartas, ela foi utilizada para encontrar os *links* para as lojas que as vendem. Isso acarreta em dois problemas. O primeiro é desempenho, pois agora o Scrapy deverá abrir muito mais páginas, fazendo o processo tomar mais tempo. O segundo é que é necessário um código diferente para cada loja que não segue o mesmo padrão HTML. Felizmente, a maioria das lojas utilizam um plano da LigaMagic que fornece uma loja virtual para uso, e essas lojas virtuais seguem o mesmo padrão HTML. As lojas que utilizam um padrão diferente não foram abordadas neste trabalho. A Figura 14 representa a interface de venda em uma loja virtual.

Figura 14 – Carta “Brontodonte Destruidor”.



Fonte – Ligamagic (2019)

Para extrair dados específicos no Scrapy é utilizado XPath. O XPath é uma linguagem de consulta que nos ajuda a navegar por documentos que usam marcadores, como os arquivos XML e HTML (W3C, 2016).

Através da ferramenta “Inspecionar” do Google Chrome foi possível identificar os campos onde os dados necessários ficam. A partir do XPath, foi necessário criar diferentes expressões regulares para extrair os dados, pois existem mudanças no padrão, como carta em promoção, carta sem promoção, campo “Extras” presente na interface, campo “Extras”

ausente na interface. O código a seguir, é o XPath do valor em promoção do primeiro campo da [Figura 14](#).

```
1 /html/body/table/tbody/tr[2]/td/table[3]/tbody/tr/td[2]/table/tbody/tr/
    td[2]/div[2]/table/tbody/tr/td[2]/div[2]/table/tbody/tr[2]/td[6]/font
```

Listing 4.7 – XPATH.

O Scrapy oferece arquivos para que seja possível alterar o comportamento da ferramenta, eles são *items.py*, *middleware.py*, *pipelines.py* e *settings.py*. Neste trabalho o único utilizado foi o *pipelines.py*. Nos outros arquivos foram utilizados os valores padrão. Os dados extraídos pelo Scrapy passam pelo *pipeline* em formato JSON. Neste momento é possível fazer correções nos dados, como descartar dados em que a quantidade ou o valor chega como zero devido algum problema na interface da loja. O *pipeline* também foi utilizado para armazenar os dados extraídos no banco de dados.

4.6 Metaheurística

Nesta seção serão abordados as fases de desenvolvimento do *simulated annealing*. De modo geral, o *simulated annealing* é dividido em duas *threads* que funcionam simultaneamente.

A *thread* Produtor é responsável pela geração de resultados. Serão geradas novas soluções até que a temperatura do sistema chegue na temperatura final. A cada iteração, se a solução encontrada for melhor que a anterior, ou se uma solução pior for aceita pelo critério de aceitação, ela é enviada à *thread* Consumidor.

A *thread* Consumidor é composta por um avaliador de soluções. A partir de um conjunto de soluções, é feito o cálculo da fronteira de Pareto. As soluções que não fazem parte da fronteira são descartadas, e as que fazem, são submetidas a novas iterações em novos conjuntos vindos da *thread* Produtor.

O código do *simulated annealing* possui várias constantes para definir diversos valores necessários durante a execução. Para facilitar no gerenciamento dessas constantes, foi criado o arquivo *parameter.txt*, onde todas as constantes são definidas. Exemplos de constantes utilizadas são a temperatura inicial, temperatura final, número de *threads* Produtor, entre outros.

As próximas seções explicarão melhor o funcionamento do *simulated annealing*.

4.6.1 Estrutura de Dados

Primeiramente, foi necessário desenvolver a estrutura dos dados que o Scrapy resulta. Estes dados possuem quatro campos: nome da carta, nome da loja, quantidade e preço. Para representar esses dados, foram utilizados três estruturas.

A estrutura *card_dict* é composta por um dicionário, cuja chave é um *id* dado a carta, com os valores em tupla de nome da carta e quantidade requerida da carta.

```
1 {0: ("chemister's insight", 2), 1: ("piper of the swarm", 4), ...}
```

Listing 4.8 – Estrutura *card_dict*.

A estrutura *store_dict* é composta por um dicionário com a chave *id* e o valor como nome da loja.

```
1 {0: "Arsenal Nerd", 1: "A Taverna", ...}
```

Listing 4.9 – Estrutura *store_dict*.

E por último, a matriz *content_table*. Suas linhas são referentes a cartas e suas colunas referentes a lojas. Alguns dos campos que as cartas possuem não foram representados neste trabalho, como a edição da carta, o estado físico ou se é uma carta *foil*¹. Devido a isso, existe cartas com o mesmo nome na mesma loja com valores de quantidade e preço diferentes. Então, foi necessário fazer com que cada campo da matriz contivesse uma lista de tuplas de quantidade e preço. A Tabela 11 a seguir mostra como ficou a estrutura da matriz.

Tabela 2 – Estrutura *content_table*.

[(10, 0.3),(2, 1.0)]	[(5, 10.0)]	...
[(2, 0.5)]	[(10, 0.5),(5, 0.9)]	...
...

Fonte – Elaborado pelo autor.

Através dessas estruturas foram desenvolvidas funções úteis, como descobrir o preço de uma carta, quantidade de certa carta em uma loja, calcular o preço final da compra, entre outros.

Por último, foi desenvolvido uma estrutura que armazena os resultados. A *result_table* possui a estrutura de acordo com a Tabela 3. Suas linhas são referentes a cartas, e suas colunas referentes a lojas. As posições da matriz armazenam a quantidade de cada carta que será comprada. Quando comparado com a *content_table*, é possível definir, por exemplo, que será comprado 2 unidades da carta “chemister’s insight” na loja “Arsenal Nerd” por 0,6 centavos. Na próxima seção será abordado como foi utilizado essas estruturas para geração de resultados.

4.6.2 Solução Inicial

O primeiro passo para a geração de resultados é criar uma solução inicial. Essa solução inicial não possui requisitos de qualidade no resultado, mas necessita ser factível,

¹ Carta especial que possui uma película brilhante em sua superfície.

Tabela 3 – Estrutura *result_table*.

2	1	...
0	3	...
...

Fonte – Elaborado pelo autor.

isto é, apenas possuir cartas em lojas que realmente as possuem em estoque. O processo de geração da solução inicial é feito passando por cada carta e escolhendo aleatoriamente uma loja que a possui. Então é colocado uma quantidade da carta na posição equivalente daquela loja na *result_table*. Caso a loja escolhida não possuir cartas suficientes, outra loja será sorteada para receber o restante.

4.6.3 Geração de Vizinhança

A partir da solução inicial, novas soluções serão geradas a partir de sua vizinhança. Isto é feito realizando ações de “*swap*”, ou seja, trocar elementos da *result_table* por outros na mesma linha.

A Figura 15 representa o funcionamento da função *swap_change_all*. Essa função recebe uma linha, que se refere a uma carta. O primeiro passo do *swap* é reiniciar aquela linha, fazendo com que todos os valores recebam zero. O segundo passo é selecionar uma loja que irá receber a quantidade requerida daquela carta, respeitando o estoque da loja. Caso a loja selecionada não possua estoque suficiente daquela carta, o segundo passo irá se repetir, selecionando uma nova loja até que a quantidade requerida daquela carta tenha sido satisfeita.

Figura 15 – Funcionamento do *swap*.

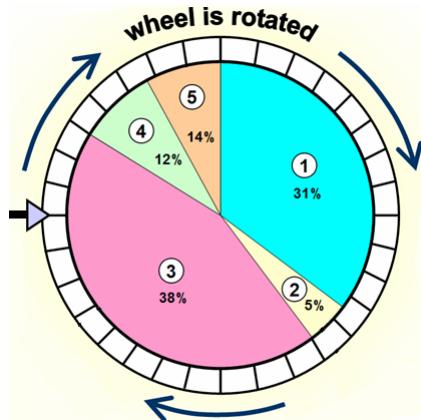
Solução atual					
0	0	0	4	...	
SWAP – Passo 1					
0	0	0	0	...	
SWAP – Passo 2					
0	2	0	0	...	
SWAP – Passo 2					
0	2	2	0	...	

Fonte – Elaborado pelo autor.

A seleção de novas lojas para o *swap* é feito utilizando um conceito chamado de *roulette wheel*. A ideia é que, para cada carta, é distribuído fatias de probabilidade para lojas serem escolhidas, em um intervalo de 0 a 100. Então, é selecionado um número aleatório e a loja que estiver na posição desse número será a escolhida para receber a

quantidade de compra na *result_table*. A Figura 16 representa o modo como a roleta funciona. Cada carta deve possuir sua própria roleta para manter todas as soluções geradas factíveis. Por exemplo, se a loja X não possui determinada carta, logo, ela não pode ser elegida como a loja para receber aquela carta, sendo assim, ela não estará presente na roleta daquela carta.

Figura 16 – *Roulette wheel*.



Fonte – Newcastle (2019)

Foram criados dois modos da roleta. A uniforme, representada pela Figura 17, é a roleta em que todas as lojas possuem o mesmo tamanho na fatia de probabilidade para serem escolhidas. A Figura 18 representa a roleta por quantidade, em que o tamanho dado as fatias de probabilidade são proporcionais a quantidade total de cartas que as lojas possuem. Depois de alguns testes, a roleta por quantidade ficou definida como o método a ser utilizado, pois ela tende a dar prioridade a lojas que possuem maior estoque, evitando colocar na compra lojas que possuem poucas cartas, o que acarreta no aumento do número de lojas no resultado final. A partir do arquivo *parameter.txt* é possível definir em *ROULETTE_OPTION* qual roleta o programa irá executar.

Figura 17 – Uniforme.

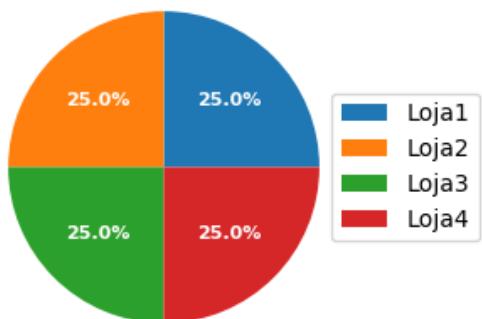
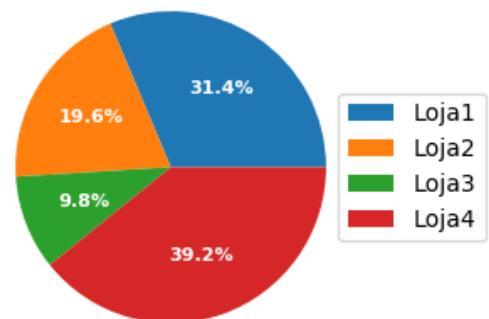


Figura 18 – Quantidade.



Fonte – Elaborado pelo autor.

4.6.4 Avaliação de Resultados

O problema a ser resolvido pelo *simulated annealing* é multiobjetivo. Isso significa que não temos somente um objetivo para minimizar ou maximizar. Para este trabalho foram utilizados três objetivos de minimização: preço, quantidade de lojas e quantidade de cartas faltando.

Por motivos de escopo do trabalho, não foram utilizados os valores reais dos fretes das lojas. Mas ainda assim, o frete é um importante contabilizador no preço final da compra. Devido a isso, foi utilizado o objetivo de minimizar a quantidade de lojas, pois mesmo não sabendo o valor exato do frete, sabemos que quanto menos lojas forem usadas na compra, menor será o valor pago.

Em algoritmo genético existe o conceito chamado *fitness*, que é a forma de avaliar o quão boa uma solução é. Baseado neste conceito, foi criado a função *get_fitness*, que retorna uma tupla com os valores dos três objetivos calculados a partir da *result_table*. Essa função possui uma particularidade, os multiplicadores. Os multiplicadores são valores constantes que multiplicam cada objetivo para evitar que o critério de aceitação não perceba pequenas mudanças nos valores, como por exemplo, os valores de quantidade de lojas como 1 e 2. Após testes, os multiplicadores ficaram definidos como: x100 para quantidade de lojas e x1000 para quantidade de cartas faltando. O multiplicador para preço possui uma diferença entre os outros. Com objetivo de penalizar ainda mais soluções com muitas lojas, o objetivo de preço é submetido a ($\text{Preço} + (\text{Número}_\text{Lojas} * 10)$). Os multiplicadores são constantes disponíveis para serem alteradas no arquivo *parameter.txt*.

O critério de aceitação define se uma solução irá substituir a solução anterior. Para isso, ela deve ser melhor que a solução anterior, ou ser submetida a uma probabilidade de aceitação. O que define se essa probabilidade será alta ou baixa, é o valor resultante da [Equação 2.4](#) abordada no [Capítulo 2](#). Uma alta temperatura do sistema, tende a aceitar soluções ruins, enquanto a temperatura baixa tende a aceitar somente soluções melhores ou com pouca piora no resultado. As constantes *LAMBDA1*, *LAMBDA2* e *LAMBDA3*, são referentes aos três objetivos e estão disponíveis no arquivo *parameter.txt*. A partir delas é possível definir pesos aos objetivos explicitando quais são mais importantes que os outros.

4.6.5 Produtor

A *thread* Produtor é responsável pela geração de soluções. O Produtor pode ser composto por uma ou mais *threads* executando ao mesmo tempo. O código a seguir é a representação do Produtor.

```
1 result_table = gera_solucao_inicial()
2 for i in range(CADEIAS_MARKOV):
3     temperatura_atual = TEMPERATURA_INICIAL
```

```

4     while ((temperatura_atual > TEMPERATURA_FINAL)):
5         for card in card_dict.items():
6             new_result_table = swap_change_all(result_table, card)
7             if (random.uniform(0, 1) <= criterio_aceitacao(result_table,
8                 new_result_table, temperatura_atual)):
9                 result_table = new_result_table
10                solutions.append(result_table)
11                if (len(solutions) >= MAXIMO_SOLUCOES):
12                    solution_deliver.put(solutions)
13                    solutions = []
14                temperatura_atual = resfria_temperatura(temperatura_atual)

```

Listing 4.10 – Produtor.

Gera-se uma solução inicial. Em seguida, existe um *loop* para cada cadeia de Markov definida na constante *CADEIAS_MARKOV*. Será gerada soluções enquanto a temperatura não atingir a temperatura final. Cada carta será submetida a função *swap_change_all* para receber novas lojas. Esse novo resultado passa pelo critério de aceitação, se for aceito, ele se torna a nova solução atual.

Todas as soluções aceitas são armazenadas no Produtor até que atinja um tamanho predefinido na constante *MAXIMO_SOLUCOES*. Após isso, são enviadas à *thread Consumidor* através do método *solution_deliver.put()*. Armazenar soluções no Produtor foi feito pelo motivo de a *thread Consumidor* não conseguir receber soluções enviadas individualmente no mesmo ritmo da *thread Produtor*, o que acarretou em problemas de estouro de memória.

Por fim, é feito o resfriamento da temperatura. O cálculo é feito através da [Equação 2.1](#) abordada no [Capítulo 2](#).

4.6.6 Consumidor

A *thread Consumidor* é responsável calcular a fronteira de Pareto das soluções recebidas. Só pode haver uma *thread Consumidor*.

O código a seguir representa o funcionamento do Consumidor. Inicialmente ele recebe uma constante com o número de *threads Produtor* ativas. Quando um Produtor termina sua execução, ele envia a mensagem “*thread_finished*”.

```

1 n_thread_count = N_PRODUTOR
2 while (n_thread_count > 0):
3     message = solution_deliver.get()
4     if (message == 'thread_finished'):
5         n_thread_count -= 1
6     else:
7         solutions += message
8         solutions = calcula_pareto(solutions)

```

```
9 result_deliver.put(solutions)
```

Listing 4.11 – Consumidor.

A cada conjunto de soluções recebidas, o Consumidor os submete ao cálculo da fronteira de Pareto. Aqueles que não fazem parte da fronteira serão descartados. Os que fazem, permanecem até serem dominados por novas soluções. Quando todas as *threads* Produtor encerrarem, o Consumidor retorna os resultados que estão na fronteira para que passem pelo processo de pós-otimização.

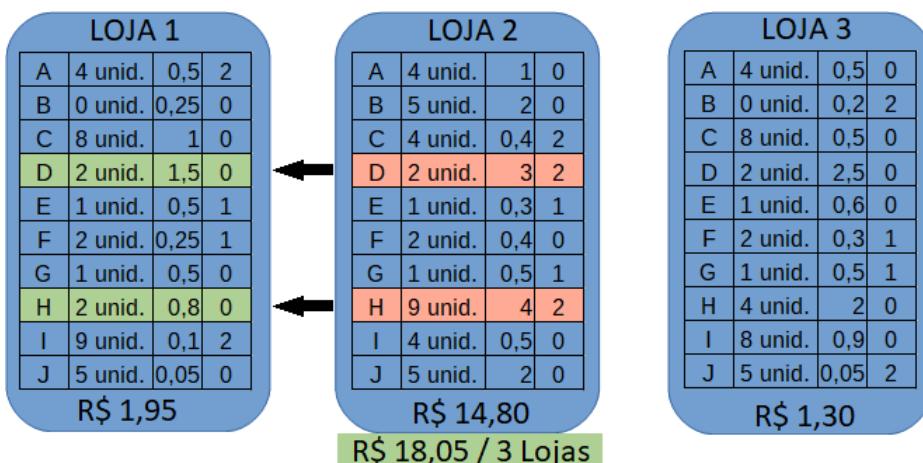
4.7 Pós Otimização

O *simulated annealing* não garante encontrar o ótimo global de um problema. Devido a isso, não existe garantia de que as soluções entregues por ele sejam globalmente ótimas, ou seja, o processo pode resultar em soluções que sejam ótimos locais. A partir dessa observação identificou-se uma oportunidade de melhorar ainda mais os resultados da metaheurística por meio de um procedimento de pós-otimização, baseado no método guloso.

O método guloso ou *greedy* é utilizado para encontrar soluções para problemas de otimização. Segundo [Rocha e Dorini \(2004\)](#), a principal característica do método guloso é tomar decisões apenas com a informação atualmente disponível, selecionando a partir de uma sequência de escolhas, o caminho localmente ótimo.

A pós otimização foi dividida em duas fases: realocação e eliminação. A primeira fase tem o objetivo de reorganizar as cartas nas lojas presentes na solução, tentando reduzir o preço. A [Figura 19](#) representa a fase de realocação. Cada carta da solução é movida para a loja que a possua com o menor preço.

Figura 19 – Fase 1 - Realocação.



Fonte – Elaborado pelo autor.

A Figura 20 é o resultado do exemplo anterior. Pode-se perceber que apenas mudando as cartas de lojas, sem alterar a combinação de lojas que compõem a solução, foi possível conseguir uma redução significativa no resultado.

Figura 20 – Resultado da realocação.

LOJA 1			
A	4 unid.	0,5	2
B	0 unid.	0,25	0
C	8 unid.	1	0
D	2 unid.	1,5	2
E	1 unid.	0,5	1
F	2 unid.	0,25	1
G	1 unid.	0,5	0
H	2 unid.	0,8	2
I	9 unid.	0,1	2
J	5 unid.	0,05	0
R\$ 6,55			

LOJA 2			
A	4 unid.	1	0
B	5 unid.	2	0
C	4 unid.	0,4	2
D	2 unid.	3	0
E	1 unid.	0,3	1
F	2 unid.	0,4	0
G	1 unid.	0,5	1
H	9 unid.	4	0
I	4 unid.	0,5	0
J	5 unid.	2	0
R\$ 1,60			

LOJA 3			
A	4 unid.	0,5	0
B	0 unid.	0,2	2
C	8 unid.	0,5	0
D	2 unid.	2,5	0
E	1 unid.	0,6	0
F	2 unid.	0,3	1
G	1 unid.	0,5	1
H	4 unid.	2	0
I	8 unid.	0,9	0
J	5 unid.	0,05	2
R\$ 1,30			

R\$ 9,45 / 3 Lojas

Fonte – Elaborado pelo autor.

A segunda fase consiste em tentar reduzir o número de lojas. A Figura 21 representa a fase de eliminação. É removido da solução a loja com o menor número de cartas e, se possível, as cartas que ela possuía são movidas para as outras lojas. Em caso negativo, é guardado o nome das cartas que faltaram. Se essa nova solução não tiver faltando mais que 10% do total de cartas e essa solução pertencer a fronteira de Pareto, ela será aceita como solução final.

Figura 21 – Fase 2 - Eliminação.

LOJA 1			
A	4 unid.	0,5	2
B	0 unid.	0,25	0
C	8 unid.	1	0
D	2 unid.	1,5	2
E	1 unid.	0,5	1
F	2 unid.	0,25	1
G	1 unid.	0,5	0
H	2 unid.	0,8	2
I	9 unid.	0,1	2
J	5 unid.	0,05	0
R\$ 6,55			

LOJA 2			
A	4 unid.	1	0
B	5 unid.	2	0
C	4 unid.	0,4	2
D	2 unid.	3	0
E	1 unid.	0,3	1
F	2 unid.	0,4	0
G	1 unid.	0,5	1
H	9 unid.	4	0
I	4 unid.	0,5	0
J	5 unid.	2	0
R\$ 1,60			

LOJA 3			
A	4 unid.	0,5	0
B	0 unid.	0,2	2
C	8 unid.	0,5	0
D	2 unid.	2,5	0
E	1 unid.	0,6	0
F	2 unid.	0,3	1
G	1 unid.	0,5	1
H	4 unid.	2	0
I	8 unid.	0,9	0
J	5 unid.	0,05	2
R\$ 1,30			

R\$ 9,45 / 3 Lojas

Fonte – Elaborado pelo autor.

A Figura 22 é o resultado da fase de eliminação. Podemos perceber que apesar de aumentar o preço da compra, o número de lojas é reduzido, acarretando na redução do

valor do frete. O processo de eliminação será repetido na solução até ela deixe de fazer parte da fronteira de Pareto.

Figura 22 – Resultado da eliminação.

LOJA 1

A	4 unid.	0,5	2
B	0 unid.	0,25	0
C	8 unid.	1	0
D	2 unid.	1,5	2
E	1 unid.	0,5	1
F	2 unid.	0,25	1
G	1 unid.	0,5	1
H	2 unid.	0,8	2
I	9 unid.	0,1	2
J	5 unid.	0,05	0

R\$ 7,05

LOJA 3

A	4 unid.	0,5	0
B	0 unid.	0,2	2
C	8 unid.	0,5	2
D	2 unid.	2,5	0
E	1 unid.	0,6	1
F	2 unid.	0,3	1
G	1 unid.	0,5	1
H	4 unid.	2	0
I	8 unid.	0,9	0
J	5 unid.	0,05	2

R\$ 9,95 / 2 Lojas

Fonte – Elaborado pelo autor.

As soluções que não possuem cartas faltando possuem prioridades sobre as que faltam. Então, mesmo que a fronteira seja composta apenas de soluções em que se falta cartas, as opções de compra ainda terão soluções que não faltam cartas. Desse modo, o usuário é quem tem a escolha, pagar mais e ter todas as cartas, ou economizar um pouco, a custo de algumas cartas.

Por fim, as soluções são convertidas em formato JSON para serem salvas no banco de dados.

5 RESULTADOS

Neste capítulo serão apresentados os resultados deste trabalho. Os resultados estão divididos em quatro seções: apresentação das instâncias, validação do *Simulated Annealing*, validação da pós otimização e resultados finais.

5.1 Instâncias

A Tabela 5 define as particularidades de cada instância. Foram testados diferentes formatos com o objetivo de variar as amostras de teste, pois cada formato possui uma variedade de cartas diferentes disponíveis para usar. O formato *Commander* possui cartas desde o início do jogo, o formato *Modern* possui cartas desde 2003 e o formato *Standard* possui apenas as cartas mais recentes. Cartas mais recentes possuem maior estoques nas lojas, contudo algumas das cartas antigas se tornam mais difíceis de encontrar.

Para cada formato foi escolhido um *deck* que possui a faixa de preço mais alta, entre R\$ 1000,00 a R\$ 3000,00 e outro com a faixa de preço mais baixa, entre R\$ 100,00 a R\$ 200,00.

O formato *Commander* possui exatamente 100 cartas e os formatos *Modern* e *Standard* no geral possuem 75 cartas, podendo variar. O campo “Qtd. Únicas” é a quantidade de cartas diferentes que o *deck* possui. Essa quantidade reflete no desempenho e na qualidade do resultado, já que quanto mais cartas diferentes, maior é o espaço de busca do problema.

Tabela 4 – Instâncias.

	Formato	Faixa Preço	Qtd. Cartas	Qtd. Únicas
Atraxa	<i>Commander</i>	Alto	100	100
Niv-Mizzet	<i>Commander</i>	Baixo	100	83
Storm	<i>Modern</i>	Alto	75	28
UB-Mill	<i>Modern</i>	Baixo	75	23
Esper	<i>Standard</i>	Alto	75	27
Mono-Blue	<i>Standard</i>	Baixo	75	22

Fonte – Elaborado pelo autor.

A listagem das cartas que compõem cada instância está disponível no Apêndice A. As instâncias foram escolhidas através de listas criadas por usuários do site LigaMagic.

5.2 Validação Simulated Annealing

Para mostrar a validação do *Simulated Annealing* foi utilizado os *decks* Atraxa e Niv-Mizzet, pois são os que possuem a maior quantidade de cartas diferentes.

A [Figura 23](#) e [Figura 24](#) demonstram o processo de refinamento do *Simulated Annealing*. O eixo X representa a quantidade de lojas e o eixo Y representa o preço da compra. Cada cor no gráfico representa uma fronteira de Pareto. É criado uma nova fronteira a cada 2000 soluções geradas. Como se trata de um problema de minimização, os melhores resultados são os que se concentram no canto inferior esquerdo.

Figura 23 – Atraxa.

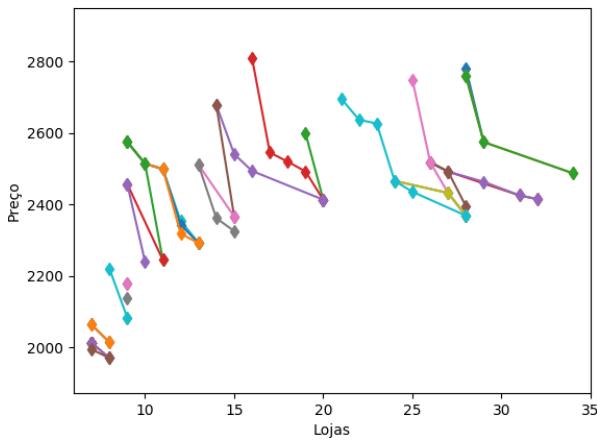
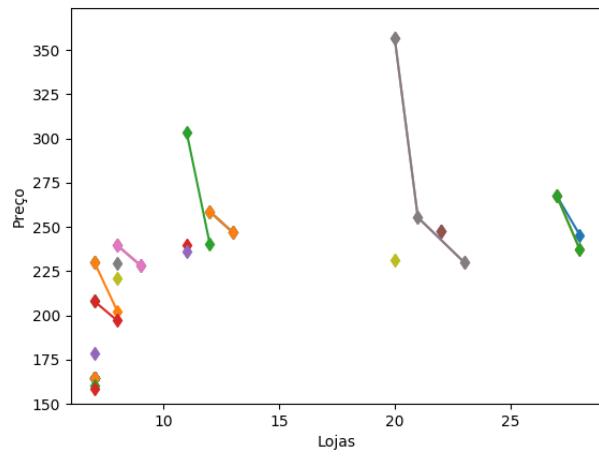


Figura 24 – Niv-Mizzet.



Fonte – Elaborado pelo autor.

Podemos perceber que as primeiras soluções no canto superior direito são ruins, com o preço alto e várias lojas. Na medida que mais iterações acontecem, a fronteira vai melhorando. Na [Figura 24](#) existe um espaço onde a fronteira teve uma mudança abrupta, isso pode ter acontecido devido a duas causas. A primeira pode ser que o *Simulated Annealing* conseguiu encontrar um caminho que levasse a uma solução muito boa. A segunda é que isso pode ter acontecido devido ao fato de que as soluções encontradas eram um ótimo local, fazendo com que melhores soluções não fossem encontradas na vizinhança e, devido a isso, após uma nova cadeia de Markov soluções ruins foram permitidas devido a temperatura alta, fazendo com que o espaço de busca pudesse ser explorado em outras regiões, levando a novos bons resultados.

5.3 Validação Pós Otimização

O objetivo da pós otimização é realocar as cartas na posição ótima utilizando as lojas das soluções do *Simulated Annealing*, além de tentar reduzir o número de lojas na solução. A pós otimização só trabalha a partir do subconjunto de lojas que já estão presente em uma solução, e não no conjunto total de lojas disponíveis.

Figura 25 – Atraxa.

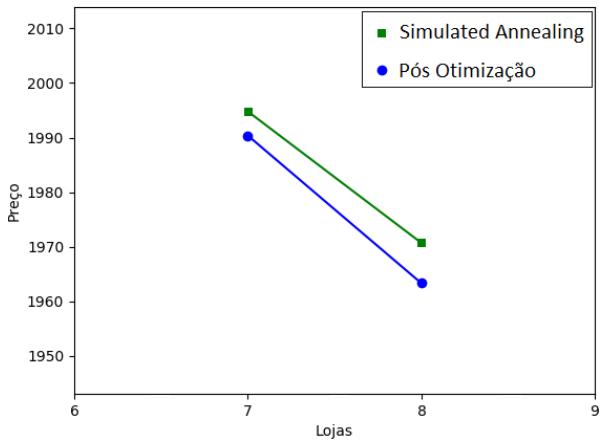
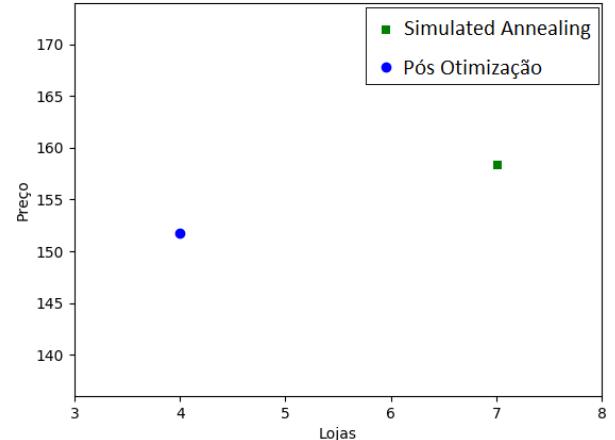


Figura 26 – Niv-Mizzet.



Fonte – Elaborado pelo autor.

A Figura 25 é um exemplo da realocação feita pela pós otimização. Algumas instâncias não possuem mudanças ao passar pela pós otimização, pois já possuem suas cartas alocadas na posição certa, mas isso geralmente ocorre apenas com instâncias pequenas. Nas instâncias grandes, pode haver uma grande mudança no preço ao passar por esse procedimento.

A Figura 26 é uma solução em que a pós otimização conseguiu não apenas reduzir o preço, mas também o número de lojas. Reduzir o número de lojas não acontece de forma tão frequente quanto reduzir apenas o preço, isso se deve ao fato de que a pós otimização irá remover a loja que possui menos cartas e terá que realocar todas as cartas no restante das lojas da solução e dependendo da carta, nenhuma das outras lojas podem contê-la, além de que essa nova solução pode não fazer parte da fronteira das soluções boas.

Em alguns casos, a pós otimização pode oferecer uma solução em que se faltam algumas cartas, para redução do preço e do número de lojas. Essas opções não foram abordadas nos resultados, por motivos de comparação com outra ferramenta e por geralmente os usuários preferirem a compra completa.

5.4 Resultados Finais

Para a validação deste trabalho foi utilizado a ferramenta de cotação do site LigaMagic para comparação de resultados. Essa ferramenta não existia quando esse trabalho foi proposto, o que mostra que o mercado realmente precisava de um serviço desse tipo. A principal vantagem que esta ferramenta possui é que todos os dados já estão em seu banco de dados, não precisando passar pela etapa de extração de dados, que é a etapa mais demorada da ferramenta proposta por esse trabalho. A Figura 27 é a interface da ferramenta de cotação da LigaMagic.

Figura 27 – Ferramenta de cotação da LigaMagic.

The screenshot shows the LigaMagic price calculator interface. At the top left is the MTG Brasil logo and "São Paulo - SP". Below it is a table for purchasing cards:

Card	Estoque	Comprar	R\$ Unit.	R\$ Total
Rastreador de Ulvenwald / Ulvenwald Tracker	1 unid.	<input type="button" value="1"/> +	R\$ 2,95	R\$ 2,95
Boca // Alimentar / Mouth // Feed	23 unids.	<input type="button" value="1"/> +	R\$ 0,95	R\$ 0,95
Asa-solar de Kinjalli / Kinjalli's Sunwing	11 unids.	<input type="button" value="1"/> +	R\$ 3,50	R\$ 3,50

Below the card table is a section for "Formas de Envio" (Shipping Options):

- R\$ 9,00 - carta registrada (com seguro)
- R\$ 31,43 - PAC com seguro
- R\$ 60,43 - SEDEX

To the right of the card table is a sidebar for "Endereço para Envio" (Shipping Address) with fields for name, address, city, state, and zip code, along with a link to "Alterar Lista ou Endereço".

At the bottom right is a "Sumário Geral" (General Summary) box:

- 100 Itens: R\$ 102,34
- 5 Envios: R\$ 51,65
- Total: R\$ 153,99

Buttons include "Adicionar ao Carrinho >" (Add to Cart) and "Finalizar Compra" (Finish Purchase). A checkbox at the bottom accepts terms and conditions from Power9, Universo Magic, and Neowalkers Geek Store.

Fonte – Ligamagic (2019)

A seguir serão apresentados as comparações entre o ScryX e a LigaMagic. Note que a LigaMagic gera apenas uma opção de compra, limitando a possibilidade de escolha do usuário. Após a geração da cotação, a LigaMagic calcula o valor do frete das lojas presentes na compra.

Por motivo de escopo do projeto, o cálculo do frete não foi envolvido neste trabalho. Note, no entanto que a metaheurística e a pós otimização trabalham para reduzir o número de lojas, e indiretamente, o preço do frete. Mas sabemos que o frete é um valor que impacta no valor final. A partir de uma análise nos resultados da LigaMagic, foi percebido que a maioria dos fretes estão no valor de aproximadamente R\$ 10,00 no modo carta registrada dos Correios. Então, para a comparação das duas ferramentas utilizaremos esse modo de frete. No entanto, existem exceções em que o frete pode ser consideravelmente mais alto que esse valor. Esse é um dos motivos do ScryX retornar mais de uma opção, para que o usuário tenha o controle de escolher qual combinação de lojas ele deseja comprar.

A Figura 28 é a comparação do *deck* Atraxa. Essa foi a única instância em que a LigaMagic obteve melhor resultado em um dos objetivos, quantidade de lojas. Mas, quando o preço final é calculado, existe uma grande economia em se comprar nas opções que possuem 7 ou 8 lojas, no lugar da solução da LigaMagic com 5 lojas. A LigaMagic teve o resultado de R\$ 2552,67 em 5 lojas. O preço final, considerando o frete de cada loja a R\$ 10,00 é de R\$ 2602,67. Os preços finais do ScryX foram de R\$ 2060,39 e R\$ 2043,33, gerando uma economia de R\$ 542,28 e R\$ 559,34 respectivamente.

A Figura 29 é a comparação do *deck* Niv-Mizzet. O preço final da LigaMagic é de R\$ 243,13 contra R\$ 191,72 do ScryX. O ScryX obteve uma economia de R\$ 51,41.

Figura 28 – Atraxa.

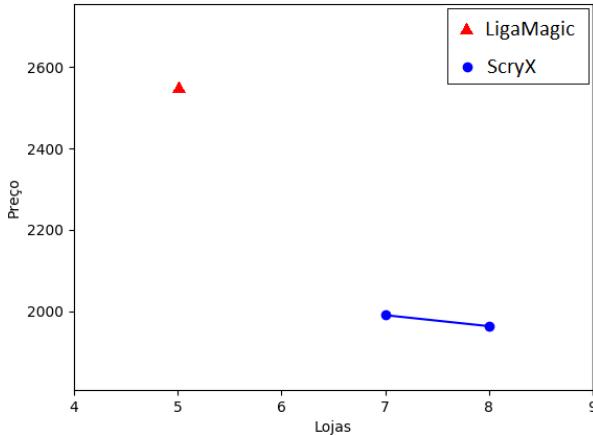
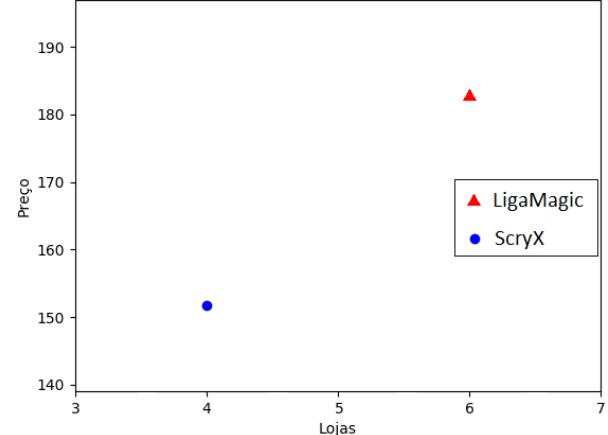


Figura 29 – Niv-Mizzet.



Fonte – Elaborado pelo autor.

A Figura 30 é a comparação do *deck* Storm. A LigaMagic obteve o valor final de R\$ 2975,00. O ScryX obteve três opções no valor final de R\$ 2835,18, R\$ 2759,67 e R\$ 2703,82. A economia varia de R\$ 139,82, R\$ 215,33 e R\$ 271,18 respectivamente.

A Figura 31 é a comparação do *deck* UB-Mill. O valor final da LigaMagic foi de R\$ 169,64. Enquanto os valores do ScryX foram de R\$ 171,26 e R\$ 147,05. Podemos notar que a LigaMagic obteve uma economia de R\$ 1,62 comparando com a opção de 2 lojas do ScryX, mas quando comparado com a opção de 3 lojas, o ScryX obteve uma economia de R\$ 22,59. Com esse exemplo, podemos ver que nem toda opção gerada pelo ScryX é uma boa solução. Por esse motivo fica a critério do usuário qual opção escolher.

Figura 30 – Storm.

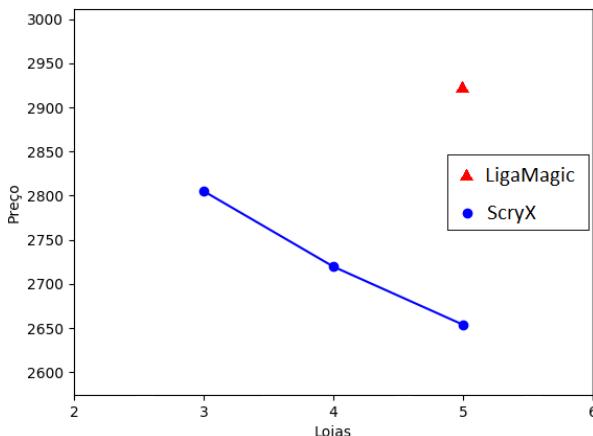
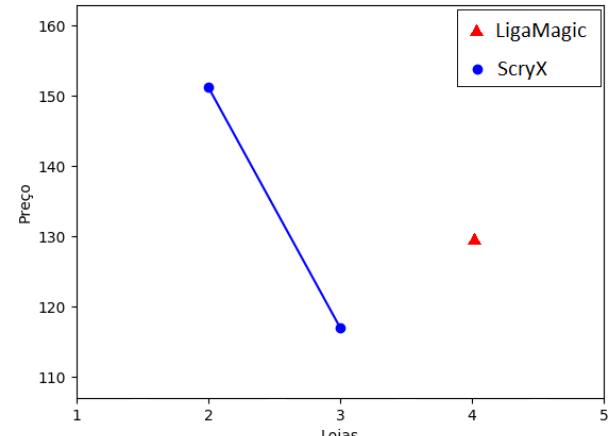


Figura 31 – UB-Mill.



Fonte – Elaborado pelo autor.

A Figura 32 é a comparação do *deck* Esper. A LigaMagic obteve o valor final de R\$ 1671,42. O ScryX obteve os valores de R\$ 1645,93 e R\$ 1531,67. O ScryX obteve uma

economia de R\$ 25,49 a R\$139,75 respectivamente.

A Figura 33 é a comparação do *deck* Mono-Blue. O valor final da LigaMagic foi de R\$ 163,19, enquanto o ScryX obteve R\$ 141,57. Houve uma economia de R\$ 21,62 na opção do ScryX.

Figura 32 – Esper.

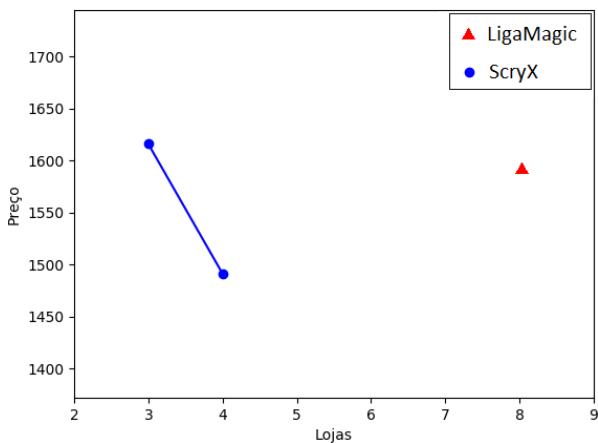
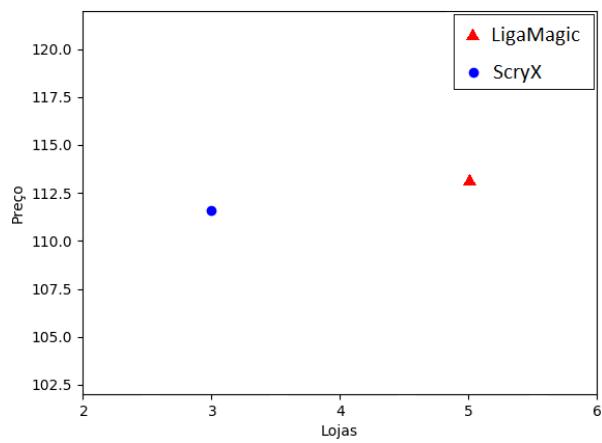


Figura 33 – Mono-Blue.



Fonte – Elaborado pelo autor.

Tabela 5 – Análise de resultados.

	LigaMagic	Opções 1 - 2 - 3			Relação ScryX - Ligamagic		
Atraxa	2602,67	2060,39	2043,33	-	+20,84%	+21,5%	-
Niv-Mizzet	243,13	191,72	-	-	+21,15%	-	-
Storm	2975,00	2835,18	2759,67	2703,82	+4,7%	+7,24%	+9,12%
UB-Mill	169,64	171,26	147,05	-	-0,95%	+13,32%	-
Esper	1671,42	1645,93	1531,67	-	+1,53%	+8,37%	-
Mono-Blue	163,19	141,57	-	-	+13,25%	-	-

Fonte – Elaborado pelo autor.

A Tabela 5 apresenta um panorama geral dos resultados encontrados neste trabalho. A última seção da tabela mostra a porcentagem que será economizado se a compra for realizada no ScryX. Podemos notar que as soluções encontradas pelo ScryX são boas em qualquer formato. Já a LigaMagic possui dificuldade para encontrar uma boa combinação de lojas quando utilizado decks com maior número de cartas diferentes e formatos com maior variedade de cartas, como o *Commander*.

6 CONSIDERAÇÕES FINAIS

A ferramenta ScryX, proposta por este trabalho de conclusão de curso atendeu aos requisitos que foram levantados durante a modelagem do sistema e atendeu aos objetivos específicos propostos.

O ScryX foi capaz de realizar cotações de preço com bons resultados finais em todas as instâncias executadas, tendo uma economia no valor da compra que varia entre 8,37% a 21,5% nas melhores opções de compra.

Essa ferramenta elimina a necessidade dos usuários de realizar cotações de forma manual, que pode demandar muito tempo e não possui garantias de que o preço será o melhor. Essa ferramenta era necessária aos jogadores de *Magic*, fato que é evidenciado pelo surgimento de outro sistema similar durante o desenvolvimento deste trabalho de conclusão de curso.

Uma das instâncias neste trabalho foi testada com o objetivo de realmente ser comprada por um jogador de *Magic*, que classificou a ferramenta como excelente, por eliminar a necessidade de fazer a cotação de forma manual.

Como trabalhos futuros fica o desenvolvimento de outros métodos de implementação do *Simulated Annealing*, como os mencionados no artigo de Czyzak e Jaskiewicz (1998). Também, o desenvolvimento de diferentes metaheurísticas, como exemplos os algoritmos genéticos, com o objetivo de tentar encontrar métodos que talvez encontrem melhores resultados. Além de integrar o cálculo de preço com a API dos Correios para conseguir valores de frete exatos.

Este trabalho escolheu *Magic: The Gathering* como tema para uma ferramenta de cotação de preços, mas existem outros TCGs, como *Yu-Gi-Oh!* e *Pokémon* que podem ser beneficiados por uma ferramenta como essa.

Além disso, uma ferramenta de cotação não está limitada somente a jogos de cartas, existe uma infinidade de produtos em que uma ferramenta desse tipo pode atuar. Exemplos de aplicativos de compras online, como o Mercado Livre¹ e o Taki APP², que possuem diferentes produtos sendo vendidos por diferentes vendedores, poderiam ter integrado aos seus serviços uma opção de receber uma lista de produtos e criar um carrinho de compras otimizado automaticamente. Fica como trabalhos futuros a elaboração de ferramentas de cotações para outras áreas, como as citadas acima.

¹ URL do Mercado Livre: <https://www.mercadolivre.com.br/>

² URL do Taki APP: <https://takiapp.com.br/>

REFERÊNCIAS

- ARENDA, R. C. *Implementação de Aplicação Web para Controle de Gastos da Central Telefônica da UTFPR*. Dissertação (B.S. Thesis) — Universidade Tecnológica Federal do Paraná, 2013. Citado na página [28](#).
- ARROYO, J. E. C. *Heurísticas e Metaheurísticas para Otimização Combinatória Multiobjetivo*. 2002. Universidade Estadual de Campinas. Citado na página [31](#).
- BAYKASOGLU, A.; OWEN, S.; GINDY, N. A taboo search based approach to find the pareto optimal set in multiple objective optimization. *Engineering Optimization*, Taylor Francis, v. 31, n. 6, p. 731–748, 1999. Citado na página [31](#).
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML - Guia do usuário*. [S.l.]: Elsevier Brasil, 2006. Citado na página [39](#).
- CZYZAK, P.; JASZKIEWICZ, A. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-criteria Decision Analysis*, John Wiley Sons, Ltd., v. 7, n. 1, p. 34–47, 1998. Citado 2 vezes nas páginas [34](#) e [65](#).
- DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier, 2004. Citado na página [29](#).
- DUARTE, N. F. B. *Frameworks e Bibliotecas Javascript*. Dissertação (Mestrado) — Instituto Superior de Engenharia do Porto, 2015. Citado na página [28](#).
- FALBO, R. d. A. A experiência na definição de um processo padrão baseado no processo unificado. *Anais do II Simpósio Internacional de Melhoria de Processo de Software*, p. 63–74, 2000. Citado na página [37](#).
- FERREIRA, R. *REST: Princípios e boas práticas*. 2019. Acessado em: 30 de Set. 2019. Disponível em: <<https://bit.ly/2wbQwYS>>. Citado na página [28](#).
- HAJEK, B. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, INFORMS, v. 13, n. 2, p. 311–329, 1988. Citado na página [32](#).
- JWT. *What is JSON Web Token?* 2019. Acessado em: 30 de Set. 2019. Disponível em: <<https://bit.ly/20I8XQq>>. Citado na página [29](#).
- JÜNGER, M.; REINELT, G.; THIENEL, S. *Practical Problem Solving with Cutting Plane Algorithms in Combinatorial Optimization*. 1994. Citado na página [30](#).
- KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. Citado na página [33](#).
- LAARHOVEN, P. J. M. van; AARTS, E. H. L. *Simulated Annealing: Theory and Applications*. [S.l.]: Springer, Dordrecht, 1987. v. 37. 7–15 p. Citado 2 vezes nas páginas [32](#) e [33](#).

- LIGAMAGIC. *Ligamagic*. 2019. Acessado em: 27 de Ago. 2019. Disponível em: <<https://bit.ly/33WmIzr>>. Citado 3 vezes nas páginas 23, 49 e 62.
- LÓSCIO, B. F.; OLIVEIRA, H. R. d.; PONTES, J. C. d. S. Nosql no desenvolvimento de aplicações web colaborativas. *VIII Simpósio Brasileiro de Sistemas Colaborativos*, v. 10, n. 1, p. 11, 2011. Citado na página 29.
- MONGODB. *What Is MongoDB?* 2019. Acessado em: 24 de Set. 2019. Disponível em: <<https://bit.ly/2OVjEeQ>>. Citado na página 35.
- MTGJSON. *Our Mission*. 2019. Acessado em: 27 de Set. 2019. Disponível em: <<https://bit.ly/2lJaXek>>. Citado na página 37.
- NAYAK, A.; PORIYA, A.; POOJARY, D. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, v. 5, n. 4, p. 16–19, 2013. Citado na página 30.
- NEWCASTLE, U. *Roulette Wheel Selection*. 2019. Acessado em: 14 de Out. 2019. Disponível em: <<https://bit.ly/32hYnTs>>. Citado na página 53.
- OBELISK. *An Introduction to Magic: The Gathering*. 2017. Acessado em: 05 de Ago. 2019. Disponível em: <<https://bit.ly/33fTJpS>>. Citado na página 23.
- OLIVEIRA, D. d. J. *Uma proposta de arquitetura para Single-Page Applications*. Dissertação (B.S. Thesis) — Universidade Federal de Pernambuco, 2017. Citado na página 28.
- ORMONT, J. *pareto-frontier*. 2016. Acessado em: 12 de Ago. 2019. Disponível em: <<https://bit.ly/2ORuwia>>. Citado na página 31.
- OSLTON, C.; NAJORK, M. Web crawling. *Foundation and Trends® in Information Retrieval*, v. 4, n. 3, p. 175–246, 2010. Citado na página 30.
- PILGRIM, M. *Dive Into Python*. [S.l.]: Apress, 2008. Citado na página 36.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. [S.l.]: McGraw Hill Brasil, 2016. Citado na página 38.
- ROCHA, A.; DORINI, L. B. *Algoritmos gulosos: definições e aplicações*. 2004. Universidade Estadual de Campinas. Citado na página 56.
- SCAVUZZO, M.; NITTO, E. D.; CERI, S. Interoperable data migration between nosql columnar databases. *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOCW*, 2014. Citado na página 29.
- SILVA, G. N. d.; SOUZA, T. N. d. *Desenvolvimento de uma aplicação para disponibilização de aplicativos classificados conforme taxonomia de bloom*. Dissertação (B.S. Thesis) — Centro Universitário de Anápolis, 2017. Citado na página 37.
- THELWALL, M. A web crawler design for data mining. *Journal of Information Science*, SAGE Journals, v. 27, n. 5, p. 319–325, 2001. Citado na página 30.
- W3C. *XML Path Language (XPath)*. 2016. Acessado em: 10 de Out. 2019. Disponível em: <<https://bit.ly/2M4lECv>>. Citado na página 49.

WIZARDS, o. t. C. L. *The history of magic*. 2019. Acessado em: 05 de Ago. 2019. Disponível em: <<https://bit.ly/2sv7T5k>>. Citado 2 vezes nas páginas 23 e 27.

APÊNDICE A – INSTÂNCIAS

Tabela 6 – *Deck Atraxa - Commander.*

1 Fortified Village	1 Sunpetal Grove	1 Scattered Groves
1 Isolated Chapel	1 Drowned Catacomb	1 Woodland Cemetery
1 Breeding Pool	1 Port Town	1 City of Brass
1 Mana Confluence	1 Choked Estuary	1 Irrigated Farmland
1 Forest	1 Hallowed Fountain	1 Glacial Fortress
1 Island	1 Temple Garden	1 Fetid Pools
1 Swamp	1 Plains	1 Exotic Orchard
1 Hinterland Harbor	1 Prairie Stream	1 Godless Shrine
1 Watery Grave	1 Temple of Malady	1 Temple of Plenty
1 Temple of Enlightenment	1 Temple of Deceit	1 Temple of Mystery
1 Temple of Silence	1 Evolving Wilds	1 Command Tower
1 Overgrown Tomb	1 Sunken Hollow	1 Canopy Vista
1 Path to Exile	1 Swords to Plowshares	1 Oath of Nissa
1 Pyramid of the Pantheon	1 Thrummingbird	1 Winding Constrictor
1 Oath of Ajani	1 Gyre Sage	1 Azorius Signet
1 Golgari Signet	1 Orzhov Signet	1 Simic Signet
1 Ashiok, Nightmare Weaver	1 Magistrate's Scepter	1 Call the Gatewatch
1 As Foretold	1 Astral Cornucopia	1 Commander's Sphere
1 Jace, Cunning Castaway	1 Oath of Gideon	1 Oath of Jace
1 Chromatic Lantern	1 Liliana, the Last Hope	1 Eternal Witness
1 Ajani Steadfast	1 Atraxa, Praetors' Voice	1 Master Biomancer
1 Oracle's Vault	1 Dovin Baan	1 Viral Drake
1 Jace, Architect of Thought	1 Tezzeret's Gambit	1 Elspeth, Knight-Errant
1 Kiora, Master of the Depths	1 Narset Transcendent	1 Fathom Mage
1 Kiora, the Crashing Wave	1 The Chain Veil	1 Crystalline Crawler
1 Sorin, Lord of Innistrad	1 Tamiyo, Field Researcher	1 Ajani, Mentor of Heroes
1 Freyalise, Llanowar's Fury	1 Deepglow Skate	1 Garruk, Primal Hunter
1 Jace, Unraveler of Secrets	1 Liliana Vess	1 Inexorable Tide
1 Nissa, Vital Force	1 Sylvan Reclamation	1 Tamiyo, the Moon Sage
1 Doubling Season	1 Vraska the Unseen	1 Ajani Unyielding
1 Elspeth, Sun's Champion	1 Deploy the Gatewatch	1 Merciless Eviction
1 Teferi, Temporal Archmage	1 Sorin Markov	1 Sorin, Grim Nemesis
1 Vraska, Relic Seeker	1 Garruk, Apex Predator	1 Akroma's Memorial
1 Ugin, the Spirit Dragon		

Fonte – Elaborado pelo autor.

Tabela 7 – Deck Niv-Mizzet - Commander.

1 Lonely Sandbar	1 Forgotten Cave	1 Swiftwater Cliffs
1 Izzet Boilerworks	1 Desolate Lighthouse	10 Island
1 Highland Lake	1 Memorial to Genius	9 Mountain
1 Vivid Crag	1 Izzet Guildgate	1 Vivid Creek
1 Temple of the False God	1 Reliquary Tower	1 Emergence Zone
1 Quest for Ancient Secrets	1 Gorgon's Head	1 Curiosity
1 Elixir of Immortality	1 Brainstorm	1 Fellwar Stone
1 Star Compass	1 Electrostatic Field	1 Doublecast
1 Goblin Electromancer	1 Curious Homunculus	1 Mask of Avacyn
1 Magmaquake	1 Curse of the Swine	1 Gorgon Flail
1 Mizzium Mortars	1 Mind Stone	1 By Force
1 Young Pyromancer	1 Cathartic Reunion	1 Izzet Signet
1 Comet Storm	1 Starstorm	1 Tormenting Voice
1 Frantic Search	1 Seething Song	1 Commander's Sphere
1 Mind Spring	1 Invoke the Firemind	1 Clear the Mind
1 Darksteel Ingot	1 Izzet Locket	1 Ophidian Eye
1 Izzet Cluestone	1 Curator's Ward	1 Pull From Tomorrow
1 Beacon Bolt	1 Saheeli, Sublime Artificer	1 Windfall
1 Blue Sun's Zenith	1 Fateful Showdown	1 Bond of Insight
1 Tamiyo's Epiphany	1 Murmuring Mystic	1 Pull from the Deep
1 Chain Reaction	1 Mystic Retrieval	1 Jace's Sanctum
1 Talrand, Sky Summoner	1 Chemister's Insight	1 Insidious Will
1 The Mirari Conjecture	1 Docent of Perfection	1 Hypersonic Dragon
1 Mana Geyser	1 Metallurgic Summonings	1 Jaya Ballard
1 Jaya, Venerated Firemage	1 Precognitive Perception	1 Psychosis Crawler
1 Spelltwine	1 Melek, Izzet Paragon	1 Niv-Mizzet, Dracogenius
1 Niv-Mizzet, Parun	1 Niv-Mizzet, the Firemind	1 Thousand-Year Storm
1 Coastal Breach	1 Swarm Intelligence	

Fonte – Elaborado pelo autor.

Tabela 8 – Deck Storm - Modern.

2 Steam Vents	4 Spirebluff Canal	1 Hallowed Fountain
2 Island	4 Scalding Tarn	1 Mountain
3 Flooded Strand	1 Shivan Reef	3 Thought Scour
4 Sleight of Hand	4 Serum Visions	3 Remand
4 Baral, Chief of Compliance	3 Goblin Electromancer	4 Manamorphose
2 Grapeshot	4 Desperate Ritual	4 Pyretic Ritual
4 Gifts Ungiven	1 Empty the Warrens	2 Past in Flames
3 Dispel	1 Hope of Ghirapur	1 Shattering Spree
3 Lightning Bolt	2 Blood Moon	2 Empty the Warrens
2 Leyline of Sanctity	1 Crumble to Dust	

Fonte – Elaborado pelo autor.

Tabela 9 – *Deck UB-Mill - Modern.*

1 Nephilia Drownyard	8 Island	4 Swamp
4 Dismal Backwater	2 Evolving Wilds	2 Memory Sluice
2 Increasing Confusion	4 Dream Twist	4 Jace's Phantasm
4 Tome Scour	4 Thought Scour	4 Mind Sculpt
4 Wight of Precinct Six	1 Ultimate Price	4 Crypt Incursion
4 Pilfered Plans	3 Fraying Sanity	1 Jace, Memory Adept
3 Despise	1 Grafdigger's Cage	2 Codex Shredder
3 Ultimate Price	4 Psychic Strike	2 Set Adrift

Fonte – Elaborado pelo autor.

Tabela 10 – *Deck Esper - Standard.*

4 Isolated Chapel	4 Drowned Catacomb	4 Hallowed Fountain
4 Glacial Fortress	1 Island	1 Swamp
1 Plains	3 Godless Shrine	4 Watery Grave
2 Search for Azcanta	3 Cast Down	4 Thought Erasure
3 Negate	4 Absorb	1 Notion Rain
2 Dovin, Grand Arbiter	2 Mortify	3 Vraska's Contempt
4 Kaya's Wrath	2 Precognitive Perception	4 Teferi, Hero of Dominaria
2 Negate	3 Deputy of Detention	2 Cry of the Carnarium
4 Thief of Sanity	1 Profane Procession	2 Lyra Dawnbringer
1 Ethereal Absolution		

Fonte – Elaborado pelo autor.

Tabela 11 – *Deck Mono-Blue - Standard.*

20 Island	4 Mist-Cloaked Herald	4 Siren Stormtamer
4 Dive Down	4 Curious Obsession	4 Opt
2 Spell Pierce	2 Essence Scatter	3 Warkite Marauder
2 Chart a Course	4 Merfolk Trickster	4 Wizard's Retort
4 Tempest Djinn	2 Sleep	1 Sentinel Totem
1 Diamond Mare	1 Selective Snare	2 Disdainful Stroke
1 Negate	4 Exclusion Mage	1 Entrancing Melody
1 Chemister's Insight		

Fonte – Elaborado pelo autor.