

A Case Study for Minimal Cost Shopping in a Cluster of Online Stores

Thiago Alexandre Nakao França¹^a, Rodrigo Campiolo¹^b and Lilian Caroline Xavier Candido²^c

¹Department of Computer Science, Federal University of Technology – Paraná, Campo Mourão, Paraná, Brazil

²Department of Mathematics, Federal University of Technology – Paraná, Campo Mourão, Paraná, Brazil

Keywords: Web Crawler, Integer Linear Programming, Caching, e-Commerce.

Abstract: This work proposes and evaluates mechanisms to converge to the purchase configuration with minimal cost for a product list in an online stores cluster. We collected and stored product prices in a database, and used caching to reduce client response time. We designed, implemented, and compared two Integer Linear Programming solutions to achieve purchase configuration with minimal cost. A case study was conducted to evaluate these mechanisms. The results demonstrated that developed mechanisms found optimal solutions with response time guarantees. Empirical tests in the case study with 100 different products and 118 providers converged in about 9 seconds.

1 INTRODUCTION

Price comparison shopping websites (shopbots) such as Google Shopping, Yahoo Shopping, and Camelcamel enable shoppers to filter and compare products on many online stores. Typically, this kind of application aggregate product listing from a specific group of retailers. In web shopping, users fetch on comparison shopping sites for one specific product they want to acquire. Then the application returns a list of retailers ordered by price, reviews, or other criteria. Most of these applications work well for displaying retailers when the shoppers need to purchase one product, but they do not offer multiple items search features.

There are many niches in which buyers are interested in purchasing multiple items at once, such as trading card games, books, comics, action figures, construction materials, and market foods. In order to build a comparison shopping application for multiple products as input, there are two main issues. First, fetching the product prices from retailers web sites, and second, finding the retailers in which the cost of the products is minimum, considering that each retailer has a shipping cost.

Retrieving prices is a problem due to the need of fetching multiple products data from multiple sources. Usually the price of a product in a singular

source is retrieved via Hypertext Transfer Protocol (HTTP) requests. Therefore the number of requests triggered will be the multiplication of the number of products by the number of sources, considering one program execution.


Determining the best purchase configuration consists of selecting the stores where the purchase price is minimum. Consider a set of products $I = \{1, \dots, n\}$, a set of online stores $J = \{1, \dots, m\}$, and $k = \min(m, n)$. $C_{k,j}$ is the total number of purchase configurations that uses j online stores from m available ones, where $j \leq k$ (equation 1).


$$C_{k,j} = \frac{k!}{j! \cdot (k-j)!} \quad (1)$$


The total number of possible purchase configurations for m online stores is the sum of the number of purchase configurations with j online stores, for each $j \in J$. This is presented in the equation 2, in which C is the total number of possible purchase configurations and $C_{k,j}$ is calculated using the equation 1.

$$C = \sum_{j=1}^k C_{k,j} = 2^{k-1} \quad (2)$$

Due to the exponential number of possible purchase configurations, a brute force algorithm is not indicated to find the purchase configuration with minimum cost. We would have $n(j-1)$ comparisons for each purchase configuration in the worst case (i.e., every retailer has set I in stock). So we modeled, ap-

^a <https://orcid.org/0000-0002-1267-6513>

^b <https://orcid.org/0000-0001-7160-7262>

^c <https://orcid.org/0000-0001-9927-8843>

plied, and compared two Integer Linear Programming models to solve this problem.

We propose and evaluate mechanisms to fetch product prices from a group of online stores to find the purchase configuration in which the total price is minimum. A case study was conducted on online stores that focus on the trading card game Magic: the Gathering.

Our main contributions are: (i) mechanism to collect, store and maintain product prices from various online stores, to reduce the response time; (ii) evaluation of two Integer Linear Programming models to find the optimal purchase configuration of online stores.

The rest of this work is organized as follows. Section 2 presents other works that are related to this research. Section 3 presents the methods applied in this study, such as case study, web crawler, caching mechanisms, proposed models, and experiments specifications. Section 4 shows and discusses the results of the experiments. Section 5 exposes the identified limitations of our approach. And finally, Section 6 exposes our conclusions and future works.

2 RELATED WORKS

This section presents works that address data synchronization and combinatorial optimization problems similar to this work. Cho and Garcia-Molina (Cho and Garcia-Molina, 2000) compared metrics and synchronization politics to maintain local copies from external sources fresh. They define three synchronization politics as (i) uniform, (ii) proportional, and (iii) optimal. The optimal synchronization policy can be used when the change frequency is known. They conclude that the uniform politic keeps the local copies from autonomous databases more updated than the proportional politic and demonstrate that the optimal is better than the previous policies. However, it is needed to know the update rate of the data. Garfinkel (Garfinkel et al., 2006) provides a model to compute a purchase plan that fetches many products from distinct retailers. They propose an algorithm named Greedy Addition of Bundles (GRAB), which is a variation of the Fisher and Wolsey algorithm (FISHER and WOLSEY, 1982). GRAB presented a lower execution time than Integer Linear Programming (ILP) approach in their experiments. Garfinkel (Garfinkel et al., 2008) proposed an approach to integrate market promotions and recommendation services on comparison shopping websites. This approach used ILP to solve a combinatorial problem related to electronic commerce, focusing on maximizing discounts from a

set of applicable promotions. Their model uses restrictions to control the number of bonus products, discount coupons, and the value required to obtain free shipping. The model was applied for a set of Amazon products, and it is described in his work. (Kandi et al., 2018) proposed a resource allocation method for query optimizations in the cloud using an ILP model. The multiple items purchase problem is similar to a capacitated facility location problem. Cooper (Cooper, 1963) developed a model for the classic facility location problem. Its goal was to decide the locations of warehouses and the allocation of customer demand, given the locations and demands of customers. Various extensions of the problem have been proposed in the literature. Gao (Gao et al., 2010) proposed a model for the capacitated facility location problem with freight cost discount. Yu et. al (Yu et al., 2012) proposed a model for capacitated facility location problems in terms of serve radius and economic benefit. Even though there are similar approaches, none of these works has proposed the same model presented in our research. Our work also presents a caching mechanism to keep local copies of the prices and the optimization process applicable in a realistic situation.

3 METHODS

This section presents the methods applied in this research. It includes the case study, the web crawling technique, the caching mechanism, the proposed models to converge to the optimal purchase configurations, and experiments specifications.

3.1 Case Study

In order to evaluate the proposed mechanisms and models, a case study was conducted in a Magic: the Gathering online marketplace.

Magic: the Gathering is a top-rated trading card game in various countries, such as United States of America, Brazil, China, Japan, United Kingdom, Germany, and many more. Many online stores sell the individual cards of the game. For this kind of product, batch purchases are regular, and this happens due to the nature of the game that instigates shoppers to purchase multiple game pieces to play the game.

We developed an application that finds optimal purchase configurations for Magic: the Gathering cards. The prices and stock are collected from online marketplace that catalogs prices from various online stores. With the stock prices available, we fetched

optimal purchase configurations of various combinations of randomly selected input card lists, aiming to evaluate the response time of the proposed mechanisms and application. Details of the run executions are provided in the experiments section.

3.2 Web Crawler

Executing an HTTP request for each card at every online store has a high cost. A simple alternative is to fetch prices from a marketplace that has already indexed the prices from many retailers. In this way, only one HTTP request is needed for each input product.

In this case study, the prices from Magic: the Gathering cards are collected from a marketplace. Unfortunately, the selected source does not provide an application programming interface (API). For that reason, a web crawler that collects its web pages directly was created, respecting the requirements present in the site robots.txt file. Figure 1 illustrates the crawling process at a marketplace.

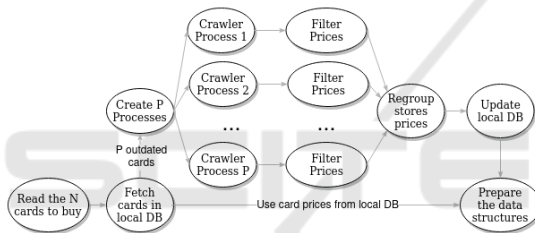


Figure 1: Prices crawling process.

In our context, every input product is a Magic the Gathering card. For each input card, a crawling process executes an HTTP GET request in the Magic: the Gathering target marketplace website. It filters card price and stock for every electronic commerce returned in the response. When all processes have finished their search request, the collected data is centralized in a data structure that organizes the cards present in each electronic commerce. This structure is stored in the local database.

3.3 Caching Mechanism

The use of cache mechanisms reduces the number of HTTP requests on a search. We store products and prices from individual and marketplace websites on the cache.

We implement a uniform update policy for cache consistency. We use a default frequency of 24 hours to refresh the prices. We chose this policy because Cho and Garvia-Molina (Cho and Garcia-Molina, 2000) compared updating metrics and policies for local copies of remote systems. They describe the uni-

form allocation policy have better results than the proportional allocation policy.

In the case study, we use a singular table in a PostgreSQL (PostgreSQL, 2021) database to store the product prices because of two main advantages: (i) the method `execute_values` from `psycopg2.extras` python library that can perform massive inserts on a table quickly, and (ii) the possibility of using Structured Query Language (SQL) queries later to retrieve the updated prices from the cards on the local database.

3.4 Proposed Models

We propose two models to address the combinatorial problem of finding the minimum price for a product list in a group of online stores. For both models, consider a set of product models indexed by $I = \{1, \dots, n\}$ and a set of retailers indexed by $J = \{1, \dots, m\}$. For every product model $i \in I$ there is the purchase quantity q_i . To every retailer $j \in J$, the shipping cost f_j is considered if the retailer j is designated for the purchase of a product. For every product model i associated to a retailer j there is the price cost c_{ij} , also there is the available stock e_{ij} for every product i in the retailer j . Linear Models 1 and 2 are presented in the following subsections. Linear Model 1 solves optimization problems when the desired quantity of each product is equal to 1, and Linear Model 2 is capable of solving problems with more units for each product. The linear programming models were implemented using PuLP Python library.

3.4.1 Linear Model 1

Be the variables x_{ij} and y_j defined by:

$$x_{ij} = \begin{cases} 1, & \text{if the product } i \text{ is purchased in the store } j \\ 0, & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{if the shipping cost from } j \text{ it's used} \\ 0, & \text{otherwise} \end{cases}$$

The problem consists in minimizing the total purchase cost for every $q_i = 1$. The cost is composed of the sum of the prices c_{ij} from selected cards, and the shipping costs f_j from the selected stores. Therefore, the objective function to be minimized is expressed in the equation 3:

$$F = \sum_{j \in J} f_j \cdot y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} \cdot x_{ij} \quad (3)$$

The objective function is subjected to the constraints:

$$\sum_{j \in J} x_{ij} = 1, \quad \forall i \in I \quad (4)$$

$$\sum_{i \in I} x_{ij} \leq n \cdot y_j, \quad \forall j \in J \quad (5)$$

$$y_j, x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \quad (6)$$

The constraint 4 ensures that a product is purchased from a single store. The constraint 5 activates the variable y_j for every store utilized, in this way the shipping y_j is considered in the objective function. The restriction 6 delimits the value interval for y_j and $x_{i,j}$, that can be only integer numbers 0 or 1.

We reinforce that the goal of this model is to minimize the function 3 constrained by 4, 5 and 6, the minimization of this function solves the best purchase configuration problem. In this model, every $q_i = 1$. In other words, there is only one product of each kind as input.

This model is similar to the Dual-Based Procedure for the Unconstrained Facility Location Problem proposed by Erlenkotter (Erlenkotter, 1978), the difference is that in our approach, x_{ij} can only assume the values 0 or 1.

3.4.2 Linear Model 2

Let the variables x_{ij} and y_j be:

$x_{ij} \in \mathbb{Z}$, quantity of items i in store j

$$y_j = \begin{cases} 1, & \text{if the shipping in the store } j \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

The problem consists in minimizing the total purchase cost, which is composed of the sum of the selected card prices c_{ij} and the shipping costs f_j from the selected stores. The function to be minimized is expressed by the equation 7:

$$F = \sum_{j \in J} f_j \cdot y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} \cdot x_{ij} \quad (7)$$

The objective function is constrained by:

$$\sum_{j \in J} x_{ij} = q_i, \quad \forall i \in I \quad (8)$$

$$x_{ij} \leq e_{ij} \quad \forall i \in I, \forall j \in J \quad (9)$$

$$\sum_{i \in I} x_{ij} \leq y_j \cdot \sum_{i \in I} e_{ij}, \forall j \in J \quad (10)$$

$$y_j \in \{0, 1\}, \quad \forall j \in J \quad (11)$$

$$x_{ij} \geq 0, x_{ij} \in \mathbb{Z}, \quad \forall i \in I, \forall j \in J \quad (12)$$

The constraint 8 ensures that the quantity of products do not surpass the desired total. The constraint 9 ensures that the quantity used of each product x_{ij} be lower or equal the stock e_{ij} . The constraint 10 ensures that the total of purchased items is lower than the available stock. The constraint 11 sets the range of possible values for y_j and the constraint 12 define the range of values for $x_{i,j}$.

Finally, we reinforce that the goal of this model is to minimize the function 7, constrained by 8, 9, 11 and 10, the minimization of this function will result in the optimal purchase configuration.

3.5 Experiments

The experiments were performed on an Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz processor, with 20GB of RAM, running Ubuntu 17.04.

We registered the execution time for requests, including 30, 60, and 100 cards. We used the card lists A (Tappedout.net, 2019a) and B (Tappedout.net, 2019b) for the executions. These lists are realistic Magic: the Gathering playable decks.

In order to evaluate the execution time for the proposed models, we performed tests using random input card lists. Both models executed lists with 40 sets of 25, 50, 100, and 200 cards, keeping the number of units of each card equal to 1 for both proposed models. Model 2 can converge to solutions even if the number of units for each card is greater than 1. Thus, we also performed tests using two units for each card.

It is unlikely that every store has every possible product in stock in a real-world scenario. For this reason, we performed tests using the same random input lists, in which every store provides all products. A random numeric price between 1 and 50 was assigned for every product missing from a store.

The evaluation of the mechanisms considers two main factors: (i) returning the purchase configuration with minimum purchase cost and (ii) performing the price fetch and the linear optimization with guarantee in terms of response time.

4 RESULTS AND DISCUSSIONS

This section presents the data crawling time, cache analysis, and linear models performance.

4.1 Web Crawler

We fetched the product prices for lists A and B, varying the number of cards. The results are presented in Table 1.

Table 1: Crawling response time in seconds.

List	30 cards	60 cards	100 cards
A	9.72	18.32	29.20
A	10.35	17.70	29.75
A	9.90	17.69	28.04
B	10.60	18.44	30.10
B	10.11	18.60	29.53
B	10.74	19.01	29.40

Even by distributing the web crawler tasks into multiple processes, the price retrieving still requires about 30 seconds for 100 distinct cards. So, for a list of 100 cards, when all the products to fetch are cache hits, the retrieval time is much smaller, in our experiments the worst time recorded was 0.2 seconds. Therefore, the experiments justify using a caching mechanism.

4.2 Performance of the Linear Models

We tested 40 sets of 25, 50, 100, and 200 cards for both real and virtual price databases, keeping the number of units of each card equal to 1 for both proposed models and testing the performance of model 2 with two units of each product. In each execution of the Integer Linear Programming, the total number of stores considered by the models varied between 120 and 127 stores.

Table 2: ILP Models optimization time in realistic stock.

Quantity of		Model 1 (s)		Model 2 (s)	
cards	units	AVG	SD	AVG	SD
25	1	1.1	0.63	1.16	0.7
25	2	-	-	1.24	1.0
50	1	2.66	3.48	2.33	2.14
50	2	-	-	5.82	7.33
100	1	11.02	11.24	9.1	7.96
100	2	-	-	20.6	15.03
200	1	46.23	72.44	31.18	35.43
200	2	-	-	48.07	47.71

Table 2 presents the average execution time and standard deviation for both of the proposed models in the realistic stock database. It is observed that for 25 cards, both models converged in an average of 1.1 seconds and 1.16 seconds, for 50 cards 2.66 seconds and 2.33 seconds, for 100 cards 11.02 seconds and 9.1 seconds, for 200 cards, model 1 took an average of 46.23 seconds while model 2 took 31.18 sec-

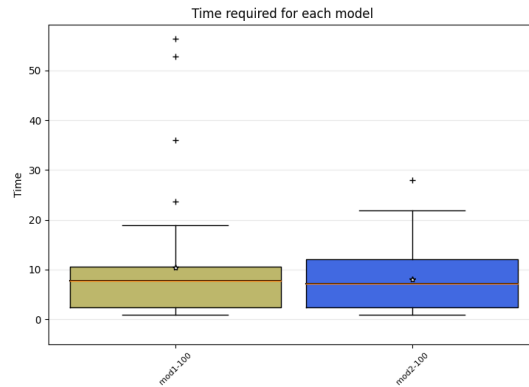


Figure 2: Execution time for 100 cards in seconds.

onds. However, the standard deviation for both cases is high, indicating a relevant variance in model execution time.

Model 2 can find the optimal purchase configuration for more units of the same product. We can observe that in every execution, the fetch time increased. However, the execution time of increasing the number of units is lower than increasing the number of different items on this model.

The execution time distribution for 100 unique input cards is presented in Figure 2. We can observe that for model 1 most of the executions took more than 2 seconds and less than 11 seconds, with some outliers. The lower execution time for model 1 with 100 cards was 0.89 seconds, while the greater was 56.3 seconds. Model 2 got similar results, and the lower execution time was 0.88 seconds and the greatest 27.9 seconds. Overall, model 2 can converge to the optimal solution faster for most inputs.

Figure 3 displays the time distribution for 200 unique input cards. We can see that compared to 100 input cards, the execution time is much greater, and here the better performance for model 2 is even more perceptible. The lower execution time for model 1 with 200 cards was 1.5 seconds, while the greater was 271.71 seconds. Similarly, the lower execution time was 1.6 seconds and the greatest 150.3 seconds for model 2.

We observed that the time to converge to the optimal solution had significant variations. In our experiments with 100 cards, the list random13 converged in 56 seconds, while the list random2 converged in 0.93 seconds. The number of iterations of the models is expected to be associated with the fetch space's characterization, which is defined by the behavior of the constraints that vary depending on the input.

We also investigated the models' behavior when every store has all the products. We used the same input lists from the previous experiments. When a

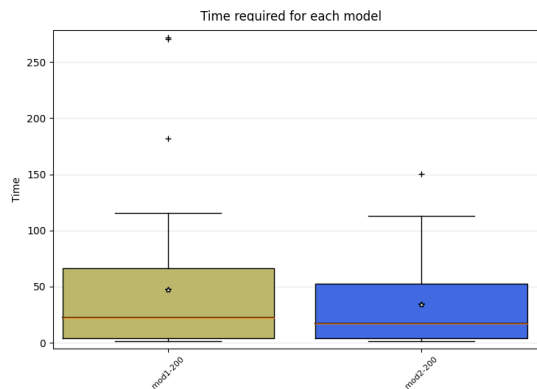


Figure 3: Execution time for 200 cards in seconds.

store misses a particular item from the input list, we insert a virtual product with a random numeric price between 1 and 50.

Table 3 presents the average execution time and standard deviation for both of the proposed models. Every store has all the products in stock, and as expected, in this scenario, the execution time is much greater than in the realistic case for every singular input list.

Table 3: ILP Models optimization time in the virtual stock.

Quantity of		Model 1 (s)		Model 2 (s)	
cards	units	AVG	SD	AVG	SD
25	1	4.41	2.33	5.02	2.54
25	2	-	-	6.21	3.58
50	1	17.8	7.52	17.8	9.34
50	2	-	-	22.5	12.8
100	1	110	112	87.6	69.3
100	2	-	-	113.4	95.9
200	1	1048	731.7	655.8	766.9
200	2	-	-	786.7	860.4

We observed that for 25 cards, the average execution time for both models is 4.41 and 5.02 seconds to find the optimal purchase configuration, for 50 cards 17.86 and 17.84 seconds, for 100 cards 110.06 and 87.67 seconds, and for 200 cards 1048.14 and 655.83 seconds. Tests with two units of each card, model 2 converged in an average of 6.21 seconds for 25 cards, 22.57 seconds for 50 cards, 113.46 seconds for 100 cards, and 786.75 seconds for 200 cards.

As expected, when every store has every input item in stock, the time to converge to the optimal solution scales faster when compared to the realistic stock scenario. It is natural since the fetch space will be larger.

5 LIMITATIONS

The proposed solution has two main limitations: (i) the need for a single marketplace to collect the stock prices and (ii) the fetch space characterization. Updating the stock by collecting prices for many resources in real-time is a problem because of the number of HTTP requests to be made. For this reason, our approach is to fetch from a singular marketplace that already catalogs the stock from many sellers. Another limitation is the exponential processing relative to the stock and input query from a specific execution. In our experiments, for 200 input products, the execution time in both proposed models was high, having averages of 31 seconds and 46 seconds, which would not be acceptable for web applications.

6 CONCLUSION AND FUTURE WORKS

In this work, two Integer Linear Programming models were designed to solve the batch purchase combinatorial problem. We performed tests using Magic: the Gathering online stores. The products used in the experiments are randomly chosen Magic: the Gathering cards.

We performed experiments using the actual stock from online stores and virtual stock experiments. All the online stores have every input card in stock in the virtual experiments. For the realistic scenario, both solutions converged to the optimal solution in less than 60 seconds for most executions. However, in the case in which every store has every item, the execution time is at least double.

In our experiments, we observed that both models converge to the optimal solution when feasible. Model 2 was able to find the optimal solution in less time when compared to model 1. We also observed that the time required to find the optimal solution grows considerably as the number of input items grows. However, the time required to converge when the number of units of each item grows is lower.

In the context of purchasing multiple items from different sources, in cases such as Magic: the Gathering online stores, the proposed models are viable for use since there are about 150 different stores, and due to the nature of this niche, it is common that singular purchases include between 2 and 100 distinct items, since the number of cards contained in a Magic: the Gathering varies between 45 and 100 playable cards.

However, even in niches such as Magic: the Gathering, there are applications that allow ordinary users

to sell their cards online, such as Card Market ¹ and MyCards ². In this scenarios, the number of possible retailers can be considerably larger since there are much more persons playing the game than online stores.

In the current stage of our research, we are exploring heuristic models to find at least reasonable solutions if the computational cost becomes too high. We are working in parallel solutions to improve response time and investigating if choosing between an optimal solution versus reasonable solutions analyzing input from shoppers. We also intent to collect feedback of the solution in a commercial online application.

Yu, H., Gao, L., and Lei, Y. (2012). Model and solution for capacitated facility location problem. In *2012 24th Chinese Control and Decision Conference (CCDC)*, pages 1773–1776.

REFERENCES

- Cho, J. and Garcia-Molina, H. (2000). Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 117–128, New York, NY, USA. ACM.
- Cooper, L. (1963). Location-allocation problems. *Operations Research*, 11(3):331–343.
- Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Oper. Res.*, 26:992–1009.
- FISHER, M. L. and WOLSEY, L. A. (1982). On the greedy heuristic for continuous covering and packing problems. LIDAM Reprints CORE 505, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE).
- Gao, D., Jiao, W., and Zhang, J. (2010). Capacitated facility location problem with freight cost discount. In *2010 7th International Conference on Service Systems and Service Management*, pages 1–5.
- Garfinkel, R., Gopal, R., Pathak, B., and Yin, F. (2008). Shopbot 2.0: Integrating recommendations and promotions with comparison shopping. *Decision Support Systems*, 46(1):61 – 69.
- Garfinkel, R., Gopal, R., Tripathi, A., and Yin, F. (2006). Design of a shopbot and recommender system for bundle purchases. *Decision Support Systems*, 42(3):1974 – 1986.
- Kandi, M. M., Yin, S., and Hameurlain, A. (2018). An integer linear-programming based resource allocation method for sql-like queries in the cloud. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, page 161–166, New York, NY, USA. Association for Computing Machinery.
- PostgreSQL (2021). Postgresql web page. Last access: 2021-03-12.
- Tappedout.net (2019a). A commander deck sample. Last access: 2019-06-10.
- Tappedout.net (2019b). A commander deck sample. Last access: 2019-06-10.

¹<https://www.cardmarket.com/en/Magic>

²<https://mycards.com/>