Searching for Optimum Computer Builds Using Linear Programming

Maxim Edelson[1*] Shaokang Jiang[2*] Junhua Yang[3*] Jinghong Luo[4*]
[1]A16615044 [2] A59012427 [3] A59019976 [4] A15527340
University of California, San Diego
{medelson, shj002, juy024, jil040}@ucsd.edu

December 27, 2023

1 Team Assignments

Team members

Maxim Edelson Shaokang Jiang Junhua Yang Jinghong Luo

Data Curation

Maxim is scrapping data using python and do completeness, correctness checking of all data. Shaokang was scrapping data using Node.js and autoscrapping feature. Both of them are discussing on how to formulate a better, more meaningful data.

Linear Programming Formulation and Analysis

Maxim, Junhua and Jinghong are using python to formulate and analysis the problem. All of us are working to test and revise the python solution code and make it more feasible in running. Shaokang is using GAMS to formulate and analysis the problem.

Paper Writing

Maxim did majority of introduction, problem setup and primal formulation. Shaokang did majority of intended approaches, result and conclusion. Jinghong did intended contributions and contribution vs previous part. All people are working on the primal, dual, KKT formulation and paper reviewing, editing.

2 Introduction

2.1 Motivation

Nowadays, new computer components are introduced into the retail market annually. Customers may find that the abundant component choices make it difficult to choose the optimum combination of hardware components while building a new computer. This project aims to take in the main components of a computer hardware build and provide several of the highest-performing options that can meet a consumer's budget.

2.2 Previous Works

Linear programming is a modern, powerful mathematical technique that is commonly used to solve optimization problems with constraints [1],[2],[7]. In particular, there are already many research publications that focus on using linear programming to solve linearly constrained optimization problems. By leveraging linear programming techniques, researchers and practitioners can efficiently solve complex problems with multiple constraints and objectives, enabling them to make more informed decisions.

One example of using linear programming is the "diet problem", which seeks to find the optimal combination of foods to meet nutritional requirements at the lowest cost. This problem can be formulated as a linear programming problem, where the objective is to minimize the total cost of the food, subject to constraints on the required nutrients. Authors from paper [5] aim to solve the diet problem of McDonald's sets menu in the hope of finding the optimal cost while satisfying the daily calories and nutritional requirements for a person.

2.3 Intended Contributions

In this paper, we have explored state-of-art methods of applying linear programming with constraints to help us formulate our problem and analyze it using primal, dual, and KKT conditions. We came up with our hardware component performance measurement formula in order to formulate the optimization problem and set appropriate constraints based on real-world scenarios. The novelty and main attraction of this paper is applying integer programming techniques to solve computer building problem, which is a relatively new and under-explored area in the computer industry. We also compare the efficiency between commercial and open source solvers.

2.4 Organization of the Paper

The rest of the paper is divided into different sections illustrating our work. First, we go in-depth to discuss the problem statement in section 3, demonstrating the reasoning behind our objective function and constraints, as well as computing the dual formulation and KKT conditions of the primal problem. In section 4, we talk about the method we use to build the linear programming problem. Then we present the results and conclude our work in section 5.

2.4.1 Contribution vs Previous Works

Even though the application of linear programming have been widely used in various fields to solve complex optimization problems, its application in the computer industry has yet received attentions on. With this paper, we hope to make significant impact on the computer community by offering this new way of identifying the computer build to customers who are looking for cost-effective builds with optimal performance. We think by exploring into the novel field of computer building with linear programming makes us unique compared to the other previous works.

3 Statement of the Problem

3.1 Problem Setup

Before formulating the primal problem, it is necessary to discuss how we have arrived to the problem itself. In this paper, we utilized data from the website UserBenchmark.com by web scraping down component names, benchmarks, prices, samples per component, and value for GPUs, CPUs, SSDs, HDDs, and RAM (automatically updating). Using the datasets we collected, we crafted g(s) (equation (1)), which is used as a part of our objective function (equation (3)). This objective function allows us estimate the performance of a hardware based on its price, benchmark, value of the component, and number of benchmark samples.

$$g(s) = \frac{\text{benchmark * value * samples}}{\text{price}} \tag{1}$$

Formula (1) computes a ratio of performance to cost, adjusted by the hardware's overall value and the sample size used to obtain the benchmark score. A higher score indicates better performance for the price. It's important to note that this formula is just one possible way to measure hardware performance, and that there are many other factors that may be relevant depending on the context.

Next, to define our constraints, we look back to our motivation of the paper, which is to find highest-performing hardware build options that can meet the budget. To do this, we need to 1) constrain the total budget of all hardware combinations to be less than the fixed budget we specified, and 2) limit the count of each hardware component in a combination to 1 (i.e., one GPU, one CPU, etc...) because we are looking for the optimum combination of hardware components in terms of performance per cost.

3.2 Primal Formulation

$$\text{From (1), let } g(s) = \frac{s_1 * s_2 * s_3}{s_4} \text{ where } s \in R^4$$

In equation (1), s_1 is the component benchmark (with a range of 0-100), s_2 is the number of recorded benchmark samples min-max normalized between 0-1, s_3 describes whether this component is valuable from the user's perspective (with a range of 0-100), and s_4 is the current component price with no upper bounds.

Let $n \in \mathbb{N}$ be the number of components in each category (CPU, GPU, SSD, RAM)

$$\max \sum_{i\in n} g(c_i) * x_{1i} + \sum_{i\in n} g(d_i) * x_{2i} + \sum_{i\in n} g(e_i) * x_{3i} + \sum_{i\in n} g(f_i) * x_{4i} \tag{2}$$

$$\text{where } \ c, d, e, f \in \mathbb{S}^{nx4}\ ,\ \ x_1, x_2, x_3, x_4 \in \{0,1\}^n$$
subject to

$$\sum_{i\in n}(c_{i,4} * x_{1i}) + \sum_{i\in n}(d_{i,4} * x_{2i}) + \sum_{i\in n}(e_{i,4} * x_{3i}) + \sum_{i\in n}(f_{i,4} * x_{4i}) \leq b, \text{ where } b \in R_{++} \text{ is the budget} \tag{3}$$

$$\mathbf{1}^T x_1 = 1,\ \mathbf{1}^T x_2 = 1,\ \mathbf{1}^T x_3 = 1,\ \mathbf{1}^T x_4 = 1 \tag{4}$$

3.3 Dual Formulations

The Lagrangian function is

$$\begin{aligned}\mathcal{L}(x_1, x_2, x_3, x_4, \lambda, v_1, v_2, v_3, v_4) &= \sum_{i\in n} g(c_i) * x_{1i} + \sum_{i\in n} g(d_i) * x_{2i} + \sum_{i\in n} g(e_i) * x_{3i} + \sum_{i\in n} g(f_i) * x_{4i} \\ &+ \lambda(\sum_{i\in n}(c_{i,4} * x_{1i}) + \sum_{i\in n}(d_{i,4} * x_{2i}) + \sum_{i\in n}(e_{i,4} * x_{3i}) + \sum_{i\in n}(f_{i,4} * x_{4i}) - b) \\ &+ v_1(\mathbf{1}^T x_1 - 1) + v_2(\mathbf{1}^T x_2 - 1) + v_3(\mathbf{1}^T x_3 - 1) + v_4(\mathbf{1}^T x_4 - 1),\ \text{where } \lambda \geq 0\end{aligned} \tag{5}$$

To compute $\max_{x_1,x_2,x_3,x_4} \mathcal{L}(x_1, x_2, x_3, x_4, \lambda, v_1, v_2, v_3, v_4)$, let us take

$$\begin{pmatrix} \frac{\partial\mathcal{L}}{\partial x_1} \\ \frac{\partial\mathcal{L}}{\partial x_2} \\ \frac{\partial\mathcal{L}}{\partial x_3} \\ \frac{\partial\mathcal{L}}{\partial x_4} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \implies \begin{cases} \sum_{i\in n} g(c_i) + \lambda \sum_{i\in n} c_{i,4} + v_1 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(d_i) + \lambda \sum_{i\in n} d_{i,4} + v_2 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(e_i) + \lambda \sum_{i\in n} e_{i,4} + v_3 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(f_i) + \lambda \sum_{i\in n} f_{i,4} + v_4 \mathbf{1}^T = 0 \end{cases}$$

When λ and v_1, v_2, v_3, v_4 satisfy these four equations, we get

$$\begin{aligned}\max_{x_1,x_2,x_3,x_4} \mathcal{L}(x_1, x_2, x_3, x_4, \lambda, v_1, v_2, v_3, v_4) &= (\sum_{i\in n} g(c_i) + \lambda \sum_{i\in n} c_{i,4} + v_1 \mathbf{1}^T)x_1 + (\sum_{i\in n} g(d_i) + \lambda \sum_{i\in n} d_{i,4} + v_2 \mathbf{1}^T)x_2 \\ &+ (\sum_{i\in n} g(e_i) + \lambda \sum_{i\in n} e_{i,4} + v_3 \mathbf{1}^T)x_3 + (\sum_{i\in n} g(f_i) + \lambda \sum_{i\in n} f_{i,4} + v_4 \mathbf{1}^T)x_4 \\ &- \lambda b - v_1 - v_2 - v_3 - v_4 \\ &= 0x_1 + 0x_2 + 0x_3 + 0x_4 - \lambda b - v_1 - v_2 - v_3 - v_4 \\ &= -\lambda b - v_1 - v_2 - v_3 - v_4\end{aligned}$$

The Dual function is

$$\begin{aligned} & \min \; -\lambda b - v_1 - v_2 - v_3 - v_4 \\ & \text{subject to } \begin{pmatrix} \sum_{i\in n} g(c_i) + \lambda \sum_{i\in n} c_{i,4} + v_1 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(d_i) + \lambda \sum_{i\in n} d_{i,4} + v_2 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(e_i) + \lambda \sum_{i\in n} e_{i,4} + v_3 \mathbf{1}^T = 0 \\ \sum_{i\in n} g(f_i) + \lambda \sum_{i\in n} f_{i,4} + v_4 \mathbf{1}^T = 0 \\ \lambda \geq 0 \end{pmatrix} \text{ where } c, d, e, f \in \mathbb{S}^{n\text{x}4} \end{aligned} \tag{6}$$

3.4 KKT Conditions

3.4.1 Primal Constraints

$\mathbf{1}^T x_1 - 1 = 0$
$\mathbf{1}^T x_2 - 1 = 0$
$\mathbf{1}^T x_3 - 1 = 0$
$\mathbf{1}^T x_4 - 1 = 0$

$\sum_{i\in n}(c_{i,4} * x_{1i}) + \sum_{i\in n}(d_{i,4} * x_{2i}) + \sum_{i\in n}(e_{i,4} * x_{3i}) + \sum_{i\in n}(f_{i,4} * x_{4i}) - b \leq 0$

3.4.2 Dual Constraints

$\lambda \geq 0$

3.4.3 Complementary Slackness

$\lambda[\sum_{i\in n}(c_{i,4} * x_{1i}) + \sum_{i\in n}(d_{i,4} * x_{2i}) + \sum_{i\in n}(e_{i,4} * x_{3i}) + \sum_{i\in n}(f_{i,4} * x_{4i}) - b] = 0$

3.4.4 Gradient of Lagrangian With Respect to x Variables

$\nabla_{x_1} = \sum_{i\in n} g(c_i) + \lambda \sum_{i\in n} c_{i,4} + v_1 \mathbf{1}^T = 0$
$\nabla_{x_2} = \sum_{i\in n} g(d_i) + \lambda \sum_{i\in n} d_{i,4} + v_2 \mathbf{1}^T = 0$
$\nabla_{x_3} = \sum_{i\in n} g(e_i) + \lambda \sum_{i\in n} e_{i,4} + v_3 \mathbf{1}^T = 0$
$\nabla_{x_4} = \sum_{i\in n} g(f_i) + \lambda \sum_{i\in n} f_{i,4} + v_4 \mathbf{1}^T = 0$

4 Intended Approaches

In the real world case, a user should be able to select the limitation B value. In our run, we simply set the limitation of B as 2000. And initially, we set all variables to be binary since one computer can only have one components in each machine.

4.1 Quality of choice

Why you make choice There are no previous academic works discussing how to optimally select a computer component combination using linear programming algorithms. As such, we selected to use the benchmark score, sample number, value score, and price to craft what we felt was a reasonable objective function. Our function could reflect the user's perspective in taking multiple pieces of data about a hardware component into account when making a selection.

Why this is good Under normal conditions, we ran the linear programming models under strict constraints, equation (4), of equal to 1 and binary variables. To prove the tightness of relaxation of our solution, we relaxed these constraints to $\leq$ to 1 and nonnegative variables. After rerunning our experiment with these relaxed constraints and receiving the same solution distribution (all solution vectors contained only zeros and ones, i.e., are binary vectors), we can confidently declare that our model solution is tight enough in our context.

Since our problem was a linear programming problem, there would always exist an optimal solution as long as the feasible region exists. Because we are always trying to maximum the objective function, which guarantee us to find a valid solution under the given limitation B. We also tested the optimality based on the KKT conditions described in the previous section. We also tried several different B setup, we are able to find a proven optimal solution in all cases. Which proves the optimality of our model.

4.2 Novelty

Difference from previous work There is no work in the previous discussing how to select a computer using linear programming.

Consumers in the market have to use some search engine and compare each product with days of work previously[3]. Right now, we introduced a new way of thinking what to purchase and how to organize the computer in a more efficient way. User could simply run our script with the latest data to get the recommendation with seconds of work. In addition to a traditional scrapping using static web fetching, we also introduced a way to automatically scrapping data from website by using github action. The script is set to run once per three days. And a contemporary way to scrapping website by simulating user behavior to prevent blocking of IP address by using puppeteer. In this way, user is always able to use the latest data to get the optimization result. See here for the auto scrapping repository.

Our setup of combining the binary vector with the associate price also improves computational time a lot comparing to some intuitive setup of using each components price * each choice vector as the function when calculating the limitation. This reduces the equations required to handle from (Dimension of cpu * Dimension of gpu * Dimension of hdd * Dimension of mem) to only one equation. In the actual experiment, this could reduce computation time from many minutes to 0.016 seconds.

4.3 Computational complexity

Since our problem was a linear programming problem, previous studies[4] have shown that any linear programming is able to run in polynomial-time, so our model should be able to run under polynomial-time. But, since our model only have one dimension in each equation, the actual runtime should be close to O(n). Nevertheless, it is hard to scale the overhead if we have to use mixed integer solver to solve the problem. Fortunately, our model is always able to be converted to a problem using linear programming solver[6], which guarantees our model to run under O(n). However, it is hard and not necessary to measure a runtime like this. In the real world usage case, running a model like this would spent far more time on context switch and read data from memory instead of doing the actual computation. This could also provide a possible reason that our real tests always show a runtime of 0.016 seconds even with more data and more dimensions.

The memory space complexity is also O(n) where n is calculated by sum of Dimension of cpu and Dimension of gpu and Dimension of hdd and Dimension of memory.

5 Conjectured Results, Conclusion, and Future Works

5.1 Results

All models are running on an old laptop, which has 2.5 GHz CPU and 8 GB memory. In our setup of limitation as 2000, with the dataset from auto-scrapping result from here. We could reach the following result:

CPU	GPU
Intel Core i7-6700K	Nvidia GTX 1060-3GB

SSD	RAM
Corsair Vengeance LPX DDR4 3000 C15 2x8GB	Samsung 970 Evo Plus NVMe PCIe M.2 1TB

The time complexity of using mip solver in GAMS is 0.016 seconds. Time complexity of using lp solvers on GAMS could reach the same result with time complexity of 0.01 seconds. For python setup, the average time consumed for 10 runs is 0.065 seconds.

In a run with a more meaningful result with limitation as 400, we could reach the following result:

CPU	GPU
Intel Core i5-10600K	Nvidia GTX 1060-3GB

SSD	RAM
Corsair Vengeance LPX DDR4 3000 C15 2x8GB	Samsung 970 Evo Plus NVMe PCIe M.2 1TB

Time complexity stays the same as previous for GAMS. For python setup, the average time consumed for 10 runs is 0.05 seconds.

By using our way, user is able to get the recommendation result in a fraction of a second instead of manually comparing between various components. We also observed the commercial solvers generally have a better performance than open source solvers on python.

5.2 Conclusion

In this work, we developed a novel and efficient way in finding the best possible computer components pairs for a user to select under a certain budget. By introducing the benchmark, popularity, valuable and price, we formulate the problem as a convex optimization problem which maximizes a customized objective function with the consideration of all four factors. Through experiments under different settings, we show that the proposed convex optimization algorithm could run fast and efficiently generate a good result. Comparing to the previous situation where user have to spent tons of hours in selecting and comparing components when composing a new computer, our way is much more efficient and our model is suitable to run on any computer.

5.3 Future Works

Based on our current solution, possible future works may search for more meaningful objective functions. This would require additional real world analysis and people have to conduct some user studies to learn which factors among price, benchmark, popularity, and value are most important.

As part of improving this work, we would like to add more features for improving user interaction experience. Considering the size of this problem, another possible route is to build a static website by using jsLPSolver and host it on GitHub. This would be a low cost way to make general public get access to our study result and benefit a wider audience.

As our current system automatically updates the data every three days, we utilize the most up-to-date data when searching for the optimal combination. That being said, using a more meaningful combination of data from all up-to-date data may be in the user's best interest as it potentially allows for better and more meaningful combination results.

6 References

References

[1] Gordon H Bradley. "Transformation of integer programs to knapsack problems". In: *Discrete mathematics* 1.1 (1971), pp. 29–45.

[2] George B Dantzig. "Linear programming". In: *Operations research* 50.1 (2002), pp. 42–47.

[3] Richard James. "Buying a computer?" In: *ITNOW* 48.3 (2006), pp. 24–24.

[4] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. 1984, pp. 302–311.

[5] Nurul Farihan Mohamed et al. "An Integer Linear Programming Model For A Diet Problem Of Medonald's Sets Menu In Malaysia". In: 2021.

[6] Günther R Raidl and Jakob Puchinger. "Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization". In: *Hybrid metaheuristics: An emerging approach to optimization* (2008), pp. 31–62.

[7] Alan Sultan. *Linear programming: An introduction with applications*. Elsevier, 2014.