Journal of Universal Computer Science, vol. 31, no. 3 (2025), 298-309
submitted: 25/4/2024, accepted: 14/11/2024, appeared: 14/3/2025 CC BY 4.0

Integer Programming, low complexity Heuristics, and Gaussian instances for the Internet Shopping Optimization Problem with multiple item Units (ISHOP-U)

Fernando Ornelas
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
a2193338003@alumnos.uat.edu.mx)

Alejandro H. García
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0001-8054-6510, ahgarcia@docentes.uat.edu.mx)

Alejandro Santiago
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0002-3265-8531, aurelio.santiago@uat.edu.mx)

Salvador Ibarra Martínez
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0002-7106-6010, sibarram@docentes.uat.edu.mx)

José Antonio Castán Rocha
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0002-4973-0827, jacastan@docentes.uat.edu.mx)

Fausto Balderas
(Tecnológico Nacional de México, Instituto Tecnológico de Ciudad Madero, Madero City, Mexico https://orcid.org/0000-0002-3696-1757, fausto.bj@cdmadero.tecnm.mx)

Julio Laria-Menchaca
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0001-5641-8535, jlaria@docentes.uat.edu.mx)

Mayra Guadalupe Treviño-Berrones
(Faculty of Engineering Tampico, Autonomous University of Tamaulipas, Tampico, Mexico
https://orcid.org/0000-0003-4201-6986, mgtrevino@docentes.uat.edu.mx)

Abstract: The Internet Shopping Optimization Problem with multiple item Units (ISHOP-U) is a recently proven NP-Hard variant of the classical ISHOP, which considers buying more than one unit of the same product. In this work, we propose a new set of instances where the prices of the products follow a Gaussian distribution, which is more realistic in a competitive market than the original instances with random uniform prices. We compute the optimal values of the previous uniform and new Gaussian instances using an Integer Programming formulation in CPLEX. In addition, we also propose two new low-complexity heuristics, the first not metaheuristics approaches proposed for the ISHOP-U, which use a linear representation instead of the original matrix candidate solution, achieving better results than the previous Evolutionary Algorithms for the ISHOP-U from the state-of-the-art.

Keywords: The Internet Shopping Optimization Problem with Multiple Item Units (ISHOP-U), Integer Linear Programming (ILP), Instances, Benchmark, Testbed
Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1
DOI: 10.3897/jucs.150245

1 Introduction

According to the United States Census Bureau, the total retail sales for the first quarter of 2023 were estimated at $1,799.5 billion, an increment of almost 1% from the fourth quarter of 2022 [Census-Bureau, 2023]. Internet sales continue to grow year by year. This situation brings diverse challenges because customers can purchase multiple products from different stores or branches, and the costs of the products and delivery may vary. The above has made it necessary to have new mechanisms that allow customers to purchase their products more efficiently, considering the different aspects they must face. Thus, selecting the adequate approach, either exact, metaheuristic, or heuristic, is necessary to develop satisfactory solutions to bring the desired results for the customers. In this sense, to provide fast and satisfactory solutions, the development should consider the number of customers that will use the solution.

One of the best options for exact solutions is the so-called Integer Linear Programming (ILP), a class of optimization problems aiming to minimize a linear objective subject to linear constraints. They play an important role in operations research and have many real-world applications, such as transportation scheduling, manufacturing planning, and resource allocation [Wu and Lisser, 2023]. ILP has been used widely in the literature, such as [Burkardt and Garvie, 2023], where they presented the ILP approach to solving the Eternity Puzzle, a complex combinatorial tiling problem using 209 polygonal puzzle pieces called polydrafters to cover an almost-regular dodecagon exactly. The method presented can be adapted for a wide range of similar puzzle problems of the Eternity type. The results demonstrated that, with sufficient computing power, the approach would yield a solution to the complete Eternity Puzzle.

Similarly, [Ma et al., 2023] presented a compact vertex-separator-based integer linear programming formulation with fewer variables for solving the Partitioned Steiner Tree Problem (PSTP). The proposed algorithm outperforms all conventional approaches, especially for large graphs with more than ten thousand vertices in experiments with public real-world and synthetic graphs.

In the same way, to improve the quantitative and qualitative performance of Earth observation images that nowadays require increased image data sizes and satellite agility, [Hyun, 2023] presents a mixed integer linear programming (MILP) based algorithm to generate a tracking profile for an antenna-subjected to operational constraints to maximize data transmission time. The results show that the MILP-based algorithm could transmit 97.87% of the data associated with the created solutions of each transmission.

On the other hand, to solve problems in which providing a solution in a short time is crucial, heuristics have been used [Jain et al., 2023]. A heuristic is an approximate, problem-dependent set of instructions, methods, or principles designed to solve a problem at a reasonable computational cost. The heuristics are simple mechanisms designed to determine the cheapest/best/most effective solution among a set of solutions. However, due to the prevalent use of the greedy search strategy, heuristics have the problem of premature stagnation [Odili et al., 2021].

Despite the disadvantages of the heuristics, many researchers use these, such as [Shambour and Khan, 2022], in which a hyper-heuristic approach optimizes the distribution process of pilgrims over Mina tent camps efficiently. The proposed algorithm iteratively selects one heuristic among four predefined low-level heuristics to produce a new solution. After that, the algorithm applies an acceptance strategy to the late move to accept or reject a new solution. The performed simulations show that the proposed approach can effectively explore the search space and avoid falling into local minima during the iteration process.

Also, in the paper of [Łapczyńska et al., 2022], heuristic algorithms were proposed as the solution to the production scheduling process. The paper presents two approaches: practical and theoretical aspects of the usefulness and effectiveness of genetic and greedy algorithms. The results show that it is possible to use both proposed algorithms in the production process scheduling.

Another example is the paper of [Schmitt et al., 2020], in which the authors presented the metaheuristic (heuristics with local optima scaping mechanism) Ant Colony Optimization (ACO) to address the set covering problem. The results suggest that the proposed heuristic can explore different combinations in the solution space, obtaining optimal solutions for severe instances classes and improving the quality of the obtained solutions.

In the original ISHOP-U publication in [Ornelas et al., 2022], three metaheuristic approaches are studied: a Genetic Algorithm, a Cellular Genetic Algorithm, and a Water Cycle Algorithm; metaheuristics in the generality of the cases are superior to simpler heuristics because metaheuristics include at least one mechanism to escape local optima solutions. In this work, we present the first two heuristics for the ISHOP-U, and our simpler heuristics produce results significantly closer to the optimal ones, even without a local optima escape mechanism, which outperforms the current metaheuristic approaches for the ISHOP-U. The above is achieved through a linear representation of the candidate solution, which only produces feasible solutions, unlike the original publication's classical matrix representation.

Thus, both approaches, exact and heuristic, are feasible to solve the Internet shopping problem; in this sense, they are used in this paper to compare their efficiency in solving Uniform and Gaussian instances. The rest of the paper presents the proposed integer linear programming model, the proposed Gaussian instances, a linear representation instead of the original matrix for the ISHOP-U (Section 2), the design of the two low-cost heuristics (Section 3), the experimentation process (Section 4), the results obtained (section 5), and the conclusions (Section 6)

2 Preliminary concepts for the Internet Shopping Optimization Problem with multiple item Units (ISHOP-U)

This section introduces readers to all the necessary mathematical notations and formulations for the Internet Shopping Optimization problem with multiple item Units [Ornelas et al., 2024]. It also explains how we propose new instances following Gaussian distribution prices and propose a linear solution representation of the problem instead of the original matrix representation.

2.1 Integer Linear Programming (ILP) for the ISHOP-U

This section describes the Integer Linear Programming model we implement in the CPLEX IBM optimization software. Given a matrix candidate solution S where their components $s_{i,j}$ represent the number of units to buy from the product j in the store i, n represents the number of different products to buy, m the number of considered stores, d_i are constants for the delivery costs for every i store, l_j is the constant of required units for every j product, and a matrix A where their constant components $a_{i,j}$ represents the number of available units of the j product in the i store; The Equation 1 computes the objective function for the ISHOP-U minimization problem.

$$F(S) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} s_{i,j} \cdot c_{i,j} + y_i \cdot d_i \right) \tag{1}$$

In Equation 1, y_i is a binary variable with the value of 1 when buying one or more products from the i store; otherwise, 0. Since y_i is a variable from our Integer Linear Programming model, it is subject to constraints. Equation 2 adds m constraints to the model, recalling m is the number of stores.

$$\left(\sum_{j=1}^{n} s_{i,j} \leq (M \cdot y_i) \right)_{i=1}^{m} \tag{2}$$

In Equation 2, M is a big constant value (representing ∞), which ensures the y_i values are correct. On the one hand, when $\sum_{j=1}^{n} s_{i,j}$ is equal to 0, y_i cannot take the value of 1; On the other hand, when $\sum_{j=1}^{n} s_{i,j}$ is greater or equal to 1 y_i cannot take the value of 0, both cases would violate the constraint. The model has other m constraints given by Equation 3 to ensure not to violate the maximum number of units available per product per store.

$$\left((s_{i,j} \leq a_{i,j})_{j=1}^{n} \right)_{i=1}^{m} \tag{3}$$

Finally, Equation 4 adds n constraints that guarantee the acquisition of the required number of units per product.

$$\left(\sum_{i=1}^{m} s_{i,j} = l_j \right)_{j=1}^{n} \tag{4}$$

The complexity of the model objective function is $O(m \cdot n)$. At the same time, the number of constraints in the model is $2m + n$.

2.2 Gaussian Instances

In the original ISHOP-U publication in [Ornelas et al., 2022], the proposed testbed consists of three different sizes: small (10 products in 25 stores), medium (25 products in 50 stores), and large (50 products in 100 stores). Where the delivery and product prices follow a uniform random distribution, deliveries are within the range [0,10], and price products are within the range [5,50]. In a real-world scenario, products and delivery prices do not differ drastically from one store to another; instead, prices are

pretty similar. Therefore, to reflect a more realistic scenario, we propose another testbed with the same parameters and sizes of the original uniform instances but following a Gaussian distribution in the products and delivery prices. We use the Box-Muller method described in [Press, 2007] to generate random numbers with a Gaussian distribution. The method uses two random uniform numbers v_1 and v_2 in the range [-1,1] and checks if their sum of square values r^2 (see Equation 5) is inside a unit circle, $0 < r^2 \leq 1.0$. Once sure r^2 is inside the unit circle, Equation 6 can generate a Gaussian number.

$$r^2 = v_1^2 + v_2^2 \tag{5}$$

$$\mu + \sigma \cdot v_1 \cdot \sqrt{-2.0 \frac{\ln r^2}{r^2}} \tag{6}$$

In Equation 6 μ is the mean, and σ is the standard deviation from the desired Gaussian distribution. To generate the product prices, we follow the parameters $\mu = 22.5$, $\sigma = 2.5$, and for the delivery prices $\mu = 5$ and $\sigma = 0.5$. The means (μ) parameters in the Gaussian distribution use the maximum price minus the minimum price in the original uniform instances in [Ornelas et al., 2022] divided by two, e.g., $(50 - 5)/2 = 22.5$. At the same time, the standard deviations (σ) equal the means divided by 10, e.g., $22.5/10 = 2.5$. The above looks for a central tendency similar to the medium prices from the original uniform instances, with a variability of about 10%. The bound of the product prices is a maximum of 50 and a minimum of 5, while the delivery prices are bounded by 10 and 0 as maximum and minimum, respectively. A total of fifteen new Gaussian instances (five small, five medium, and five large), analogous to the original 15 uniform instances testbed in [Ornelas et al., 2022], are available at https: //github.com/AASantiago/ISHOP-U-Instances.

2.3 Linear Representation

The original ISHOP-U matrix S candidate solution representation (see Figure 1) is the fastest to access how many units of a j product are going to be bought for an i store because it is as simple as visiting their element $s_{i,j}$ with a complexity of $O(1)$, this is also the case for check the product store availability constraint directly visiting in the matrix A the position $a_{i,j}$.

$$\begin{bmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,n} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ s_{m,1} & s_{m,2} & \cdots & s_{m.n} \end{bmatrix}$$

Figure 1: Original candidate solution representation of size $m \cdot n$

However, with the representation from Figure 1, verifying the constraint of buying the required units per product is necessary [Del-Angel et al., 2024]. In our experience with Evolutionary Algorithms for the ISHOP-U [Ornelas et al., 2022], minor changes to the candidate solution, also known as mutations or perturbations, usually violate the constraint mentioned above. Moreover, correcting the candidate solution could radically

change the original solution, causing a loss of valuable information for Evolutionary Algorithms and Heuristics. To overcome the drawbacks of feasible constraint checking and solution repairing, we propose using a linear candidate solution representation, as in Figure 2.

$$[\ldots, l_1, \ldots, l_2, \ldots, l_j, \ldots, l_n]$$

Figure 2: New linear candidate solution representation of size $\sum_{j=1}^{n} l_j$

The size of the linear representation in Figure 2 equals the sum of desired units of the products to buy $\sum_{j=1}^{n} l_j$. The first l_1 positions of the linear representation in Figure 2 contain the stores to buy the first product individual units, followed by the subsequent l_2 positions having the stores to buy the second product, followed by the required stores of the other products. The above representation does not need to check the constraint of required units and, thus, does not need to repair solutions. The constraint of available units per store still needs to be checked, and this can be done in $O(1)$ with a counter auxiliary matrix equivalent to the original S matrix. Hence, the linear representation adds a memory requirement of $\sum_{i=1}^{n} l_j$ to the original representation but acquires performance for one less constraint check and no solution repairs.

3 Proposed low-complexity heuristic methods by sorting costs

This section proposes two low-complexity heuristics for the ISHOP-U using the linear representation from Section 2.3. The heart of our proposed heuristics is simple: to sort the available stores by their costs. Let us describe the Sort by Costs Heuristic 1 (SCH1) from Algorithm 1. The first step involves initializing an auxiliary matrix S (equivalent to the S matrix in the ILP model) with zeros and an empty linear representation $Solution$. The main loop of the heuristic (Lines 3-15) iterates over the different n products. An auxiliary list Q saves the stores sorted by the current jth-product prices at the main loop. Next, to know how many units of the current jth-product are in $solution$, an integer counter $units$ is initialized. An inner loop (Lines 7-15) verifies if the counter $units$ reaches the required number of units l_j for the jth-product. In case not, the algorithm confirms the availability of product units in the current x store with the condition $s_{x,j} < a_{x,j}$, if the product is still available at the x store, then the matrix position $s_{x,j}$ and the counter $units$ are incremented by a unit. Consequently, $solution$ adds the store x to their decision variables. For the cases of no available product units in x, the store x is updated with the next store in the list Q.

The algorithm's complexity in Algorithm 1 is bound by their two most expensive operations, the main loop with n steps and the sort of the stores by product price $m \log m$. Therefore, the complexity of the SCH1 heuristic is $O(nm \log m)$. This work proposes a second heuristic with the same complexity named Sort by Costs Heuristic 2 (SCH2). The difference between SCH1 and SCH2 lies in how they sort the stores. In SCH2, instead of sorting the stores by only their product price, they are sorted by the sum of their product price and delivery price. Figure 3 summarizes the steps followed by the SCH1 and SCH2 heuristics. The above heuristics preserve the two solution representations, the matrix S and the linear $solution$ representation. We study the performance of both heuristics in the results section.

Algorithm 1 Pseudocode of the proposed Sort by Costs Heuristic 1 (SCH1)

$S \leftarrow$ Initialize the matrix S with zeros
$solution \leftarrow \emptyset$
for $j = 1$ to n **do**
    $Q \leftarrow$ Sort stores by the price of the jth-product
    $x \leftarrow$ Remove the first store from Q
    $units \leftarrow 0$
    while $units < l_j$ **do**
        if $s_{x,j} < a_{x,j}$ **then**
            $s_{x,j} \leftarrow$ Increment the element $s_{x,j}$ by one
            $units \leftarrow$ Increment $units$ by one
            $solution \leftarrow$ Add to $solution$ the store x
        else
            $x \leftarrow$ Remove the next store from Q
        end if
    end while
end for

4 Experimental setup

Since the algorithms studied in this work are deterministic, we are more interested in their behavior in terms of CPU time. To accurately measure the CPU time, we follow two good practices: i) using the same computer for all the executions and ii) averaging the CPU time of the same algorithm over the same problem instance. The experimental computer is a MacBook Pro (13-inch, Late 2011), 2.4 GHz Intel Core i5, with 8GB of RAM. The CPU time is the average of 100 independent executions. The computer was not executing other applications, algorithms, or algorithm executions simultaneously, except for the boot ones in the macOS High Sierra 10.13.6 operating system. The algorithms implementation was under Java Development kit 19 using as reference the code in [Ornelas et al., 2022], and the Integer programming model was also implemented in Java using the IBM CPLEX library for the Java language. It is relevant to highlight that we use the CPLEX configuration's default option, which uses parallel computing mode among many other close source optimizations. Therefore, we use it as a black box optimization technique, which we are interested in contrasting with the proposed heuristics using a single-core deterministic computation time.

5 Results

This section presents the results of the ILP model from Section 2.1, which gives the optimal results, and the two heuristics from Section 3. The calculated results of the heuristics and the ILP model are cost, CPU time in seconds, and error percentage relative to the optimal value. For all the results from Tables I, II, and III, the best result is highlighted in dark gray, and the second-best result is highlighted in light gray per row. Table II presents the results in terms of the achieved objective values. The CPLEX column is completely highlighted in dark gray because the ILP model computes the

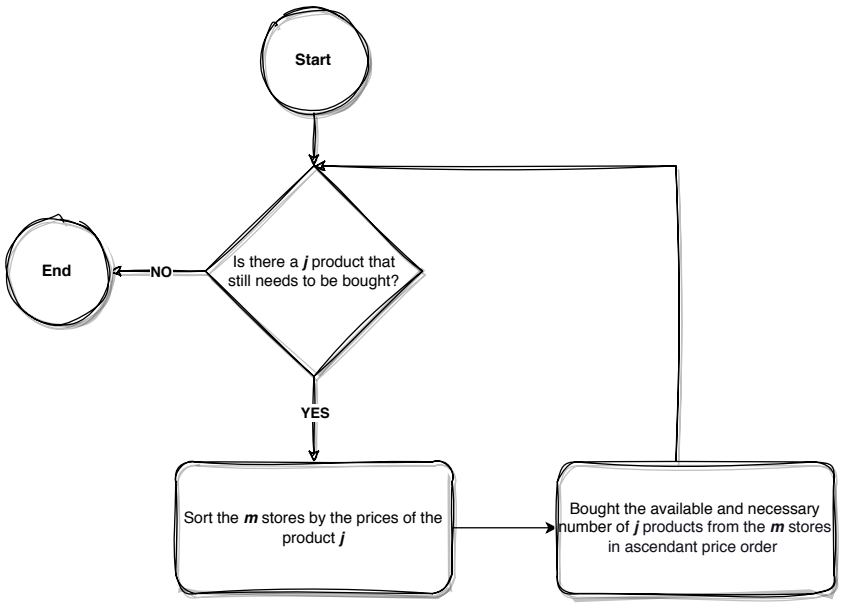

Figure 3: Flowchart diagram of the Sort By Costs Heuristics.

optimal values. Regarding SCH1 and SCH2, SCH1 achieves the best performance with a global percentage error of 0.31% and 25 second-best objective values, while SCH2 achieves a global percentage error of 3.74% and six second-best values. Table I presents a time comparison between the ILP in CPLEX, SCH1, and SCH2, with CPLEX being the slowest and SCH1 and SCH2 the fastest, both having the same average CPU time per instance. The equal CPU time between SCH1 and SCH2 is explained because the heuristics have the same computing complexity $O(nm \log m)$. The CPLEX global average for the 30 instances is 8.16 seconds, while for SCH1 and SCH2, it is 0.7 seconds, an increment in time efficiency of over 1000% with a global error of 0.31%, 3.74% concerning SCH1 and SCH2, respectively. Finally, Table 3 compares SCH1 and SCH2 and the best results from the reported Evolutionary Algorithms in the introductory ISHOP-U paper [Ornelas et al., 2022]. Table 3 clearly shows that SCH1 and SCH2 outperform the reported results in [Ornelas et al., 2022], SCH1 achieving 13 best results and two second-best results, SCH2 achieving two best results and 11 second-best results, and the best result from [Ornelas et al., 2022] achieving only two second-best results. Perhaps the above behavior is explained because the mutation operator in [Ornelas et al., 2022] works over the S matrix representation, producing, in many cases, unfeasible solutions, with the need for reparation making it harder to explore the solution space. The heuristics approach SCH1 and SCH2 differ in that they construct candidate solutions one variable at a time and always guarantee their feasibility.

Instance	CPLEX	SCH1	SCH2
UniformS1	0.15s	0.02s	0.02s
UniformS2	0.14s	0.02s	0.02s
UniformS3	0.14s	0.02s	0.02s
UniformS4	0.15s	0.02s	0.02s
UniformS5	0.16s	0.02s	0.02s
UniformM1	0.25s	0.02s	0.02s
UniformM2	0.21s	0.02s	0.02s
UniformM3	0.20s	0.02s	0.02s
UniformM4	0.21s	0.02s	0.02s
UniformM5	0.21s	0.02s	0.02s
UniformL1	0.41s	0.03s	0.03s
UniformL2	0.45s	0.03s	0.03s
UniformL3	0.45s	0.03s	0.03s
UniformL4	0.42s	0.03s	0.03s
UniformL5	0.38s	0.03s	0.03s
GaussianS1	0.16s	0.02s	0.02s
GaussianS2	0.16s	0.02s	0.02s
GaussianS3	0.16s	0.02s	0.02s
GaussianS4	0.15s	0.02s	0.02s
GaussianS5	0.16s	0.02s	0.02s
GaussianM1	0.22s	0.02s	0.02s
GaussianM2	0.22s	0.02s	0.02s
GaussianM3	0.21s	0.02s	0.02s
GaussianM4	0.27s	0.02s	0.02s
GaussianM5	0.22s	0.02s	0.02s
GaussianL1	0.48s	0.03s	0.03s
GaussianL2	0.48s	0.03s	0.03s
GaussianL3	0.45s	0.03s	0.03s
GaussianL4	0.41s	0.03s	0.03s
GaussianL5	0.48s	0.03s	0.03s
Total	8.16s	0.7s	0.7s

Table 1: Results by time

6 Conclusions

This work presents two high-performance heuristics for the Internet Shopping Optimization Problem with multiple Units (ISHOP-U), SCH1 and SCH2. The proposed heuristics outperform the results in the state of the art of the ISHOP-U mainly because they guarantee feasible solutions guided by the minimum costs available in the considered stores instead of generating random or unfeasible solutions. From Table 3, readers can observe that SCH2 beats SCH1 in two cases for the small instances, while for the medium and large instances, SCH1 outperforms SCH2 in every case. The difference in performance between SCH1 and SCH2 is explained by the fact that in the ISHOP-U formulation, the

Instance	CPLEX	SCH1	SCH2
UniformS1	447.20	452.93	451.98
UniformS2	447.22	447.22	472.60
UniformS3	524.25	529.86	547.58
UniformS4	462.92	479.64	521.77
UniformS5	489.61	492.46	491.87
UniformM1	2164.58	2186.90	2393.79
UniformM2	3069.89	3076.38	3369.41
UniformM3	3248.58	3273.78	3447.34
UniformM4	3259.24	3263.52	3437.27
UniformM5	3536.99	3540.55	3864.45
UniformL1	9110.89	9165.72	10340.07
UniformL2	8072.24	8098.30	9151.04
UniformL3	7554.83	7581.18	8513.42
UniformL4	8846.61	8859.24	10053.56
UniformL5	8351.02	8382.32	9287.65
GaussianS1	1169.48	1182.35	1181.78
GaussianS2	1075.09	1095.68	1096.01
GaussianS3	1093.71	1116.31	1114.45
GaussianS4	1310.70	1332.98	1348.84
GaussianS5	1257.60	1260.50	1260.50
GaussianM1	7022.41	7051.03	7049.42
GaussianM2	6320.00	6344.57	6353.54
GaussianM3	7002.55	7009.51	7050.18
GaussianM4	5847.48	5874.38	5874.41
GaussianM5	5596.45	5618.32	5620.69
GaussianL1	21460.01	21509.10	21619.38
GaussianL2	20568.69	20624.52	20786.90
GaussianL3	23904.89	23955.61	24089.97
GaussianL4	22692.58	22723.48	22807.47
GaussianL5	23248.51	23271.94	23376.67
Error	0%	0.31%	3.74%

Table 2: Results of the proposed heuristics

delivery prices are considered only once for one or more products bought in the same store. Therefore, when the number of products to be purchased increases, the delivery prices are less significant, prejudicing SCH2. For future research, we would like to implement linear representation in Evolutionary algorithms for the ISHOP-U; contrary to the current matrix representation, our linear representation does not produce unfeasible solutions when perturbed. Evolutionary Algorithms can benefit from the linear representation, mainly when the problem grows, which leads to the production of more unfeasible solutions using the matrix solution representation. SCH1 and SCH2 heuristics are now the baseline reference algorithms for benchmarking more complex algorithms and their

Instance	Reference [Ornelas et al., 2022]	SCH1	SCH2
UniformS1	467.04	452.93	451.98
UniformS2	457.64	447.22	472.60
UniformS3	585.75	529.86	547.58
UniformS4	479.69	479.64	521.77
UniformS5	520.69	492.46	491.87
UniformM1	3505.7	2186.90	2393.79
UniformM2	4728.01	3076.38	3369.41
UniformM3	5234.51	3273.78	3447.34
UniformM4	4861.51	3263.52	3437.27
UniformM5	5646.67	3540.55	3864.45
UniformL1	25711.6	9165.72	10340.07
UniformL2	21675.88	8098.30	9151.04
UniformL3	21869.36	7581.18	8513.42
UniformL4	24172.27	8859.24	10053.56
UniformL5	23875.34	8382.32	9287.65

Table 3: Comparison with the best reported results in the state-of-the-art

quality solutions. The above is because of their best performance in the literature.

Acknowledgements

Alejandro Santiago wants to acknowledge the Mexican Secretary of Science, Humanities, Technology, and Innovation (SECIHTI Mexico) for his SNII salary award. This work is partly funded by the Red iberoamericana de inteligencia artificial y analítica de datos 523RT0147.

References

[Burkardt and Garvie, 2023] Burkardt, J. and Garvie, M. R. (2023). An integer linear programming approach to solving the eternity puzzle. *Theoretical Computer Science*, 975:114138.

[Census-Bureau, 2023] Census-Bureau, U. S. (2023). Quarterly retail e-commerce sales 1st quarter. https://www.census.gov/retail/mrts/www/data/pdf/ec_current.pdf Accessed = 2023-07-15.

[Del-Angel et al., 2024] Del-Angel, J., Santiago, A., Ibarra-Martínez, S., Castán-Rocha, J. A., and Treviño-Berrones, M. G. (2024). A linear genetic programming approach for the internet shopping optimization problem with multiple item units (ishop-u). *Computación y Sistemas*, 28(3).

[Hyun, 2023] Hyun, J. H. (2023). Antenna tracking profile generation using mixed-integer linear programming. *Advances in Space Research*, 71(10):4104–4117.

[Jain et al., 2023] Jain, J., Walia, N., Singla, H., Singh, S., Sood, K., and Grima, S. (2023). Heuristic biases as mental shortcuts to investment decision-making: A mediation analysis of risk perception. *Risks*, 11(4).

[Ma et al., 2023] Ma, M., Men, Z., Rossi, A., Zhou, Y., and Xiao, M. (2023). A vertex-separator-based integer linear programming formulation for the partitioned steiner tree problem. *Computers & Operations Research*, 153:106151.

[Odili et al., 2021] Odili, J. B., Noraziah, A., and Zarina, M. (2021). A comparative performance analysis of computational intelligence techniques to solve the asymmetric travelling salesman problem. *Computational Intelligence and Neuroscience*, 2021:6625438.

[Ornelas et al., 2024] Ornelas, F., Santiago, A., Castan Rocha, J. A., Ibarra Martínez, S., and García, A. H. (2024). *Warm Starting Integer Programming for the Internet SHopping Optimization Problem with Multiple Item Units (ISHOP-U)*, pages 153–170. Springer Nature Switzerland, Cham.

[Ornelas et al., 2022] Ornelas, F., Santiago, A., Martínez, S. I., Ponce-Flores, M. P., Terán-Villanueva, J. D., Balderas, F., Rocha, J. A. C., García, A. H., Laria-Menchaca, J., and Treviño-Berrones, M. G. (2022). The internet shopping optimization problem with multiple item units (ishop-u): Formulation, instances, np-completeness, and evolutionary optimization. *Mathematics*, 10(14).

[Press, 2007] Press, W. H. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[Schmitt et al., 2020] Schmitt, M. F. L., Henrique Mulati, M., Aparecido Constantino, A., Hernandes, F., and Alexander Hild, T. (2020). Ant-set: A subset-oriented ant colony optimization algorithm for the set covering problem. *JUCS - Journal of Universal Computer Science*, 26(2):293–316.

[Shambour and Khan, 2022] Shambour, M. K. Y. and Khan, E. A. (2022). A late acceptance hyper-heuristic approach for the optimization problem of distributing pilgrims over mina tents. *JUCS - Journal of Universal Computer Science*, 28(4):396–413.

[Wu and Lisser, 2023] Wu, D. and Lisser, A. (2023). A deep learning approach for solving linear programming problems. *Neurocomputing*, 520:15–24.

[Łapczyńska et al., 2022] Łapczyńska, D., Łapczyński, K., Burduk, A., and Machado, J. (2022). Solving the problem of scheduling the production process based on heuristic algorithms. *JUCS - Journal of Universal Computer Science*, 28(3):292–310.