Optimization of the Book Scanning Problem via Iterated Local Search and Genetic Algorithms

Driton Alija[1], Meriton Kryeziu[1], Endrit Hoda[1], Lirim Islami[1], Lorik Mustafa[1], Yll Berisha[1], Kadri Sylejmani[1,*], Arben Ahmeti[2] and Labeat Arbneshi[1]

[1]Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit p.n., 10000, Prishtina, Kosova
[2]Faculty of Computer Sciences, AAB College, Elez Berisha, Nr.56 10000 Prishtinë, Kosova

Abstract

This short paper presents ongoing work on solving the Book Scanning problem introduced in the Google Hash Code 2020 competition. The problem requires scheduling library signups and book scanning activities to maximize the total score, subject to time and capacity constraints. Two metaheuristic approaches are investigated: Iterated Local Search (ILS) and Genetic Algorithms (GA). A Greedy Randomized Adaptive Search Procedure (GRASP) is used to construct the initial solution, which is subsequently improved through neighborhood-based search. The methods are evaluated on a set of benchmark and synthetically generated instances. Results show that both ILS and GA produce competitive solutions, with GA consistently matching or exceeding best-known results across most instances and demonstrating lower variability. ILS performs similarly in terms of solution quality but tends to exhibit higher fluctuation across runs. The findings highlight the effectiveness of metaheuristic approaches in large-scale, constrained scheduling problems and confirm GA's advantage in consistency and robustness.

Keywords

NP-hard problem, Iterated Local Search, Genetic Algorithms, Metaheuristics, Empirical Analysis

1. Mathematical Model

The Book Scanning Problem [1] can be considered a Set Packing problem with additional time and resource constraints. The Set Packing problem was formally introduced and studied in the context of combinatorial optimization and complexity theory by Garey and Johnson [2]. Below, we provide the notations and the corresponding mathematical formulation.

1.1. Notations

Let B be the set of books and L the set of libraries. Denote by D the total number of days available for scanning. For each book $b \in B$, let s_b be the score awarded when the book is scanned.

For each library $l \in L$, we define $B_l \subseteq B$ as the set of books available in library l. Let σ_l be the number of days required to complete the signup process for library l, and δ_l be the number of books that can be scanned each day from library l once the signup process is completed.

We define the following decision variables:

- $y_l \in \{0, 1\}$, $l \in L$, a binary variable which takes the value 1 if library l is selected for signup, and 0 otherwise.
- $z_{lb} \in \{0, 1\}$, $l \in L$, $b \in B_l$, a binary variable which takes the value 1 if book b is scanned from library l, and 0 otherwise.

6th International Conference in Recent Trends and Applications in Computer Sciences and Information Technologies, May 22–24, 2025, Tirana, Albania

*Corresponding author.
driton.alija@student.uni-pr.edu (D. Alija); meriton.kryeziu@student.uni-pr.edu (M. Kryeziu); endrit.hoda@student.uni-pr.edu (E. Hoda); Lirim.Islami@student.uni-pr.edu (L. Islami); lorik.mustafa2@student.uni-pr.edu (L. Mustafa); yll.berisha3@student.uni-pr.edu (Y. Berisha); kadri.sylejmani@uni-pr.edu (K. Sylejmani); arben.ahmeti@universitetiaab.com (A. Ahmeti); labeat.arbneshi@uni-pr.edu (L. Arbneshi)
https://staff.uni-pr.edu/profile/kadrisylejmani (K. Sylejmani); https://aab-edu.net/ (A. Ahmeti); https://staff.uni-pr.edu/profile/labeatarbneshi (L. Arbneshi)
0000-0002-4428-1161 (K. Sylejmani); 0000-0002-5981-7336 (A. Ahmeti)

- $u_b \in \{0, 1\}$, $b \in B$, a binary variable which takes the value 1 if book b is scanned from any library, and 0 otherwise.
- $p_{l_1,l_2} \in \{0, 1\}$, $l_1, l_2 \in L$, $l_1 \neq l_2$, a binary variable which takes the value 1 if library l_1 is processed before library l_2, and 0 otherwise.
- $t_l \in \{0, 1, \ldots, D-1\}$, $l \in L$, an integer variable which gives the day on which the signup process for library l starts.

1.2. Formulation

Using these definitions, the Book Scanning Problem can be formulated as a Mixed-Integer Linear Programming (MILP) model using the following equations:

$$\text{Maximize} \quad \sum_{b \in B} s_b u_b \tag{1}$$

$$\text{subject to:} \quad \sum_{l \in L} z_{lb} \leq 1 \quad \forall b \in B, \tag{2}$$

$$u_b \geq z_{lb} \quad \forall l \in L, \forall b \in B_l, \tag{3}$$

$$z_{lb} \leq y_l \quad \forall l \in L, \forall b \in B_l, \tag{4}$$

$$p_{l_1,l_2} + p_{l_2,l_1} \leq 1 \quad \forall l_1, l_2 \in L, l_1 \neq l_2, \tag{5}$$

$$t_{l_2} \geq t_{l_1} + \sigma_{l_1} \cdot y_{l_1} - D \cdot (1 - p_{l_1,l_2}) \quad \forall l_1, l_2 \in L, l_1 \neq l_2, \tag{6}$$

$$p_{l_1,l_2} + p_{l_2,l_1} \geq y_{l_1} + y_{l_2} - 1 \quad \forall l_1, l_2 \in L, l_1 \neq l_2, \tag{7}$$

$$t_l + \sigma_l \cdot y_l \leq D \quad \forall l \in L, \tag{8}$$

$$\sum_{b \in B_l} z_{lb} \leq \delta_l \cdot (D - t_l - \sigma_l) \cdot y_l \quad \forall l \in L, \tag{9}$$

$$\sum_{b \in B_l} z_{lb} \leq |B_l| \cdot y_l \quad \forall l \in L, \tag{10}$$

$$u_b \leq \sum_{l \in L} z_{l,b} \quad \forall b \in B, \tag{11}$$

$$y_l, z_{lb}, u_b, p_{l_1,l_2} \in \{0, 1\} \quad \forall l, l_1, l_2 \in L, \forall b \in B, \tag{12}$$

$$t_l \in \{0, 1, \ldots, D-1\} \quad \forall l \in L. \tag{13}$$

The objective function (1) states that we wish to maximize the total score of all scanned books. Constraints (2) require that each book be scanned at most once. Constraints (3) define that a book is considered scanned if it's scanned from at least one library, while constraints (4) ensure books can only be scanned from libraries that are signed up.

Constraints (5) establish that for any two distinct libraries, at most one can precede the other. Constraints (6) enforce that if library l_1 is processed before library l_2, then the signup process for l_2 can only start after the signup process for l_1 is completed. Constraints (7) ensure that for any two libraries that are signed up, one must precede the other in the signup process. Collectively, Constraints (5)–(7) enforce a sequential signup process, ensuring that only one library is processed at any given time.

Constraints (8) guarantee that library signup must be completed within the available days. Constraints (9) limit the number of books that can be scanned from a library based on its daily capacity and the number of days available after its signup process completes. Constraints (10) ensure that the number of books scanned from a library does not exceed the number of books available in that library, while constraints (11) ensure that a book can be scanned only by a signed-up library.

Finally, constraints (12) and (13) state that the decision variables are binary or non-negative integers as appropriate.

2. Solution Approach

Our approach to solving the Book Scanning problem is twofold: one based on single-state solutions, such as Iterated Local Search (ILS) [3], and the other based on population-based methods, such as Genetic Algorithms (GA) [4]. Both approaches have proven effective in solving various planning and scheduling problems across diverse domains, such as technician routing and scheduling [5], and flexible job shop scheduling [6].

2.1. Preprocessing, Search Space and Fitness Function

As part of the preprocessing phase, we begin by extracting the key structural components of the problem: books, libraries, and time constraints. Each book is associated with a unique score, while each library is characterized by its signup time, daily scanning capacity, and the list of books it holds. These books are sorted in decreasing order of their scores to prioritize higher-value selections during scanning. Additionally, we construct a hash table data structure that maps each book to the list of libraries in which it is available. This mapping facilitates efficient lookup during library selection and book allocation in later stages of the algorithm.

Figure 2.1 illustrates a snapshot of a possible solution configuration. Libraries that have already been signed up are represented as a list of hash tables, each mapping a library ID (as the key) to a list of book indices assigned for scanning from that library. Libraries that have not yet been signed up are grouped into a separate list, and books that remain unscanned are also tracked explicitly.

Based on this structure, the **search space** of the problem includes all possible ways to choose and order libraries for signup, all possible subsets of books to scan from those libraries, as well as the possible decisions to leave some libraries unsigned and some books unscanned.

More formally, let L be the set of all libraries, and let B be the set of all books. For each library $l \in L$, let $B_l \subseteq B$ denote the set of books available in that library. Then the search space consists of:

- All possible subsets $L' \subseteq L$ representing the libraries selected for signup, and all permutations of L' representing the signup order;
- For each signed-up library $l \in L'$, all possible subsets $S_l \subseteq B_l$ of books selected for scanning, such that the scanning of S_l can be completed within the remaining time after the library's signup;
- The set of libraries not selected for signup, $L \setminus L'$, and the set of books not scanned in the solution, given by $B \setminus \bigcup_{l \in L'} S_l$.

Effectively, the search space includes decisions about which subset of libraries to sign up, the order in which to sign them up, and the subset of books to scan from each selected library. It also involves deciding which libraries to exclude entirely and which books to leave unscanned.

The **fitness function** (Equation (1) in Section 1.2) evaluates the total score of all uniquely scanned books, taking into account the constraints imposed by library signup durations and daily scanning capacities. If a book is scanned by multiple libraries, its score is counted only once in the final total.

2.2. Initial solution

The initial solution is constructed using four different strategies, with the best among them adopted as the initial solution before the algorithm enters its iterative phase. Below, we provide details about each strategy.

Greedy by Signup and Book Score: This strategy prioritizes libraries with shorter signup times and higher total book scores. Libraries are sorted first by ascending signup days, then by descending total book score. Libraries are selected one by one, ensuring that the total number of days is not exceeded and that no duplicate books are scanned. For each selected library, the most valuable unscanned books are chosen, and the scanned book list and remaining days are updated accordingly.

Greedy by Efficiency: This strategy uses a priority queue (heap) to always select the library with the highest scanning efficiency. Efficiency is defined as the ratio of the potential book score (from the most

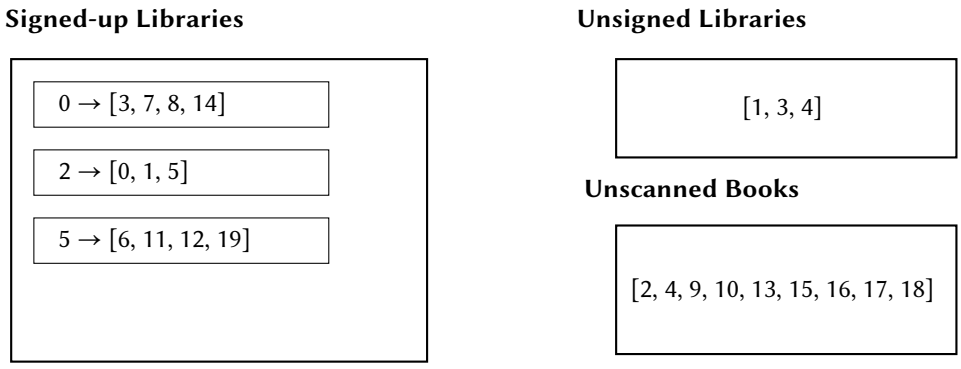

Figure 1: Solution representation for an instance with 20 books and 6 libraries, showing signed-up libraries, their assigned books, unsigned libraries, and unscanned books

valuable books that can be scanned within the remaining time) to the library's signup days. At each step, the most efficient library is selected, scanned books and remaining time are updated, and the heap is adjusted if needed.

Weighted Efficiency with Penalization: This strategy enhances the basic greedy approach by penalizing libraries selected later or those with high signup durations. It introduces two tunable parameters, α (alpha) and β (beta), to control the impact of signup time and selection order. Each library's efficiency is calculated using a weighted formula that balances book value against registration cost and selection delay. At each step, the library with the highest weighted efficiency is selected, and the scanned book list, remaining time, and library count are updated. Parameters α and β can be fine-tuned for optimal performance.

GRASP-Based Initialization: This strategy constructs the initial solution using the Greedy Randomized Adaptive Search Procedure (GRASP) [7]. Libraries are first sorted by ascending signup days and then by descending total book score. At each step, one library is randomly selected from a Restricted Candidate List (RCL), containing the top 5% of libraries according to this ordering. Libraries are added sequentially, with their books scheduled for scanning immediately after signup. The process stops when no additional library can be added without exceeding the time limit.

2.3. Neighborhood Structure

To explore the solution space effectively and escape local optima, we define a set of neighborhood operators that modify the current solution in controlled ways. These operators generate neighboring solutions by altering the structure of the library signup order and book selections, while preserving feasibility with respect to the problem constraints.

The neighborhood is composed of the following six move types:

- **Swap between two signed-up libraries**: Two libraries currently included in the solution are randomly selected, and their positions in the signup sequence are swapped. This move changes the order in which their scanning begins and can affect the number of books ultimately scanned due to timing constraints.
- **Swap of signed library with an unsigned library**: A library that has been signed up is replaced with one that has not been signed up. The unsigned library is inserted into the same position in the signup sequence, and its book selection is initialized based on available scanning time.
- **Swap books between libraries that share common books**: If two signed-up libraries contain overlapping books in their inventories, a book from one library can be swapped with a different book from the other.
- **Swap the last book of a selected library with an unscanned book**: For a given library, the last book scheduled for scanning can be replaced with a book that has not yet been scanned. This allows fine-tuning of the scanned book list, potentially replacing low-value books with higher-scoring alternatives.

- **Random reordering of libraries with reassigned book sets**: This move shuffles the library order and attempts to reassign books from their old libraries to new randomly chosen ones. For each reassigned library, books are filtered to exclude those already scanned or not available in the new library.
- **Insert Library** – Inserts a randomly selected unused library into a random position in the signed library list. From that position onward, the solution is completely rebuilt.

These neighborhood operations provide both diversification and intensification capabilities. By mixing structural changes (e.g., reordering libraries) with content-level changes (e.g., modifying scanned books), the search is guided toward promising regions of the solution space while maintaining sufficient variability to avoid premature convergence.

2.4. Iterated Local Search

The ILS algorithm [8] for the Book Scanning problem combines the described initial solution procedure with iterative refinement and perturbation mechanisms. The implementation follows the ILS with Random Restarts framework described by Luke [9]. The initial solution is generated using the procedure detailed in Section 2.2, which prioritizes libraries with shorter signup times and higher total book scores.

In the iterative phase of the algorithm, the solution is improved using a local search (hill climbing) procedure that applies several neighborhood operations, including swapping signed and unsigned libraries, reordering libraries with overlapping books, applying crossover-based reordering, and replacing the last book in a library with an unscanned, higher-value book. Neighborhoods are selected probabilistically, with 50% assigned to the move that swaps signed-up libraries with unsigned ones, 20% to the library reordering move, and the remaining 30% equally distributed among the other neighborhood operations. Only improving moves are accepted, except with a small probability (10%) of accepting worse solutions to escape local optima.

In each iteration, a perturbation step is applied using a destroy-and-rebuild strategy: a random subset of signed libraries is removed, and candidate replacements are evaluated greedily based on their average book score, scanning efficiency, and signup time. The algorithm runs until a total time limit (default: 300 seconds) or a maximum number of iterations (default: 1000) is reached.

2.5. Genetic Algorithms

The proposed approach [10] is based on the Genetic Algorithm with Elitism described by Luke [9], and initializes the population using small variations of the procedure in Section 2.2, generated through limited hill climbing steps. This introduces diversity while maintaining reasonably fit initial solutions.

After initialization, the algorithm evolves through multiple generations. In each generation, individuals are ranked by fitness, enabling effective selection. The best solution is preserved via elitism to ensure no loss of progress.

Offspring are generated using tournament selection, where two parents are chosen by selecting the best individual from small random subsets of the population. This method favors stronger candidates while preserving diversity by giving weaker ones a chance.

Crossover is performed in a partially random but structured manner. Half of the library signup order is randomly taken from the first parent, with remaining positions filled by non-duplicate libraries from the second parent. Any missing libraries are appended to ensure completeness. This ensures valid offspring while promoting trait inheritance from both parents.

Mutation is applied to each offspring with a probability of 1.0, meaning it always occurs. Instead of random tweaks, mutation runs hill climbing for several steps, integrating local search into the genetic process. This hybrid approach forms a memetic algorithm, combining global evolution with greedy local refinement.

Over successive generations, fitter individuals are more likely to reproduce, guiding the population toward better solutions. Crossover and mutation introduce variation, while selection and elitism ensure

steady improvement. The algorithm terminates after a fixed number of generations, returning the best solution found.

3. Preliminary Computational Study

The preliminary experiments were conducted on a test set of 20 instances, with each algorithm executed five times. Both ILS and GA were run for a total of 12 minutes: 2 minutes were allocated for generating the initial solution, and the remaining 10 minutes were dedicated to the iterative phase of each respective algorithm. All computations were performed on standard computing machines without the use of parallelization or specialized hardware.

The test set used for evaluation consists of two subsets. The first subset includes five benchmark instances originally provided in the Google Hash Code 2020 competition. These instances vary significantly in size and complexity, featuring between 78,600 and 100,000 books, 100 to 30,000 libraries, and scanning periods ranging from 200 to 100,000 days. The average book score ranges from 65 to over 400, and library characteristics span from small, fast-signup libraries to very large ones with high daily throughput.

The second subset contains 15 additional instances generated using a custom instance generator [11]. These synthetic instances introduce controlled variations in structural parameters such as the number of books (ranging from 300 to 100,000), number of libraries (from 11 to 850), and available scanning days (from 14 to 140). The generator also allows for tuning average signup times and book shipment capacities per library. These instances provide a more diverse and challenging testbed for evaluating the scalability and robustness of the proposed algorithms under different constraint combinations.

For evaluation purposes, we use upper bound values as a theoretical benchmark. The upper bound is estimated by calculating, for each library, the total score of the books it could scan if it were signed up first. The libraries are then ranked by their maximum potential contribution, and the scores of the top-k libraries—where k is the number of libraries that can be signed up sequentially within the time limit—are summed.

Additionally, for comparative analysis, we include the results of state-of-the-art approaches found in the literature, specifically those by Bartosz et al. [12], Indzhev et al. [13], and Rodrigues [14], who ranked 5th, 28th, and 33rd, respectively, in the Google Hash Code Competition. In the tables below, for each instance, we present the best result achieved among these three approaches and refer to it as the *Best Known* value.

Table 1 presents the performance of ILS and GA compared to MILP results and the best-known values across all benchmark instances. The column *Best Known* refers to the highest score obtained by any known method from the literature or our experiments. In several cases, both ILS and GA match the best-known results, with GA slightly outperforming ILS in a few larger instances. Integer Programming results are included where available, primarily for smaller instances due to its computational limits. Overall, the results highlight the effectiveness of metaheuristic methods, with GA showing more consistent superiority in complex instances and ILS demonstrating competitive performance in many cases.

Table 2 complements the previous results by reporting standard deviations and relative performance comparisons between ILS and GA. The standard deviations show that GA exhibits lower variability than ILS in 4 instances, while both algorithms have a standard deviation of zero in the remaining 14 instances. In terms of relative performance, both ILS and GA show small average deviations from the best-known or IP-based values, with ILS yielding an average difference of 0.23% and GA achieving the same. When comparing their best solutions directly, the average performance difference between ILS and GA is (0.00%), suggesting that while GA tends to be more stable, both methods are capable of producing similarly competitive results.

In summary, both ILS and GA demonstrate strong performance across the benchmark instances. GA shows greater consistency, with lower variability in several cases and frequent matches with the best-known solutions. ILS is equally competitive in terms of best-case results but tends to exhibit

higher variability, making it less stable across runs. The baseline approaches, including Integer Programming (where applicable), remain effective—particularly in smaller or less complex instances—but are consistently outperformed by the metaheuristic methods in larger and more challenging problem settings.

Table 1
Raw performance results of ILS and GA compared to Integer Programming and best-known values across all instances

Instance	Upper Bound	MILP	Best Known	ILS (Best)	ILS (Avg)	GA (Best)	GA (Avg)
b_read_on	8,901,900		**5,822,900**	**5,822,900**	5,822,900	**5,822,900**	5,822,900
c_incunabula	8,589,141		**5,690,888**	5,689,822	5,678,260	5,690,336	5,690,097
d_tough_choices	8,577,400		**5,109,000**	5,028,530	5,023,256	5,008,120	5,008,018
e_so_many_books	11,563,321		**5,240,809**	5,073,545	5,062,232	5,108,054	5,104,739
f_libraries_of_the_world	8,385,642		**5,348,248**	**5,348,248**	5,348,248	**5,348,248**	5,348,248
B300_L11_D20	32,269	22,085	**22,085**	**22,085**	22,085	**22,085**	22,085
B500_L14_D18	87,815	48,465	**48,465**	**48,465**	48,465	**48,465**	48,465
B750_L20_D14	116,730	65,520	**65,520**	**65,520**	65,520	**65,520**	65,520
B1k_L18_D17	172,211	127,579	**127,579**	**127,579**	127,579	**127,579**	127,579
B1.5k_L20_D40	718,226	345,553	**345,553**	**345,553**	345,553	344,479	344,479
B2.5k_L25_D50	1,393,905	671,491	**671,491**	**671,491**	671,491	**671,491**	671,491
B4k_L30_D60	2,874,948	1,383,173	**1,383,173**	**1,383,173**	1,383,173	**1,383,173**	1,383,173
B5k_L90_D21	6,414	3,597	3,582	**3,597**	3,597	**3,597**	3,597
B6k_L35_D70	4,757,757	2,307,803	**2,307,803**	**2,307,803**	2,307,803	**2,307,803**	2,307,803
B9k_L40_D80	7,360,581	3,476,231	**3,476,231**	**3,476,231**	3,476,231	**3,476,231**	3,476,231
B50k_L400_D28	136,276		**121,715**	**121,715**	121,715	**121,715**	121,715
B60k_L70_D140	36,629,631		**19,806,745**	**19,806,745**	19,806,745	**19,806,745**	19,806,745
B90k_L850_D21	869		**869**	**869**	869	**869**	869
B95k_L2k_D28	1,542,230		1,364,177	**1,366,194**	1,366,194	**1,366,194**	1,366,194
B100k_L600_D28	135,536		**129,326**	**129,326**	128,953	**129,326**	129,326

Acknowledgments

We thank all students of the Master's study program in Computer or Software Engineering, generation 2024/2025, at the Faculty of Electrical and Computer Engineering, University of Prishtina, for their contributions in implementing various algorithm components in Python, as well as developing supporting tools such as the validator, instance generator, and solution visualizer.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-4o and Claude.ai to check the grammar and spelling of the paper, as well as to analyze and reason about the mathematical model of the problem. Following the use of these AI tools, the authors reviewed and edited the content as necessary and assume full responsibility for the content of the publication.

References

[1] P. Avnimelech, Google hash code archive, https://github.com/pierreavn/google-hashcode-archive, 2023. Accessed: 2025-04-04.

Table 2
Standard deviations and relative performance comparisons of ILS and GA

Instance	ILS (STD)	GA (STD)	ILS vs. MILP / Best Known (%)	GA vs. MILP / Best Known (%)	ILS vs. GA (Best) (%)
b_read_on	0.00	0.00	0.00	0.00	0.0000
c_incunabula	7,209.97	176.28	0.02	0.01	0.0001
d_tough_choices	8,255.56	74.11	1.58	1.97	-0.0041
e_so_many_books	10,105.90	2,953.06	3.19	2.53	0.0068
f_libraries_of_the_world	0.00	0.00	0.00	0.00	0.0000
B300_L11_D20	0.00	0.00	0.00	0.00	0.0000
B500_L14_D18	0.00	0.00	0.00	0.00	0.0000
B750_L20_D14	0.00	0.00	0.00	0.00	0.0000
B1k_L18_D17	0.00	0.00	0.00	0.00	0.0000
B1.5k_L20_D40	0.00	0.00	0.00	0.31	-0.0031
B2.5k_L25_D50	0.00	0.00	0.00	0.00	0.0000
B4k_L30_D60	0.00	0.00	0.00	0.00	0.0000
B5k_L90_D21	0.00	0.00	0.00	0.00	0.0000
B6k_L35_D70	0.00	0.00	0.00	0.00	0.0000
B9k_L40_D80	0.00	0.00	0.00	0.00	0.0000
B50k_L400_D28	0.00	0.00	0.00	0.00	0.0000
B60k_L70_D140	0.00	0.00	0.00	0.00	0.0000
B90k_L850_D21	0.00	0.00	0.00	0.00	0.0000
B95k_L2k_D28	0.00	0.00	-0.15	-0.15	0.0000
B100k_L600_D28	476.52	0.00	0.00	0.00	0.0000
Average	1,302.40	160.17	0.23	0.23	0.00

[2] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, San Francisco, 1979.

[3] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, in: F. Glover, G. A. Kochenberger (Eds.), Handbook of Metaheuristics, Springer, Boston, MA, 2003, pp. 320–353. doi:10.1007/0-306-48056-5_11.

[4] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[5] A. E. Yahiaoui, S. Afifi, H. Allaoui, Enhanced iterated local search for the technician routing and scheduling problem, Computers & Operations Research 160 (2023) 106385. URL: https://www.sciencedirect.com/science/article/abs/pii/S0305054823002496. doi:10.1016/j.cor.2023.106385.

[6] L. Meng, W. Cheng, B. Zhang, W. Zou, W. Fang, P. Duan, An improved genetic algorithm for solving the multi-agv flexible job shop scheduling problem, Sensors 23 (2023) 3815. URL: https://www.mdpi.com/1424-8220/23/8/3815. doi:10.3390/s23083815.

[7] M. G. C. Resende, C. C. Ribeiro, Greedy randomized adaptive search procedures, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, volume 146 of *International Series in Operations Research & Management Science*, Springer, 2010, pp. 283–319. doi:10.1007/978-1-4419-1665-5_11.

[8] D. Alija, L. Islami, Y. Berisha, Ils for book scanning problem, https://github.com/dritonalija/ils_book_scanning, 2024. Accessed: 2024-04-05.

[9] S. Luke, Essentials of Metaheuristics, 2nd ed., Lulu, 2014. Available at https://cs.gmu.edu/~sean/book/metaheuristics/.

[10] M. Kryeziu, E. Hoda, L. Mustafa, Genetic algorithm for book scanning problem, https://github.com/meritonkryeziu0/genetic_book_scanning, 2025. Accessed: 2025-04-05.

[11] Book Scanning Instance Generator, Book scanning instance generator, https://bookscanning-ig.

netlify.app/, 2025. Accessed: 2024-04-05.
[12] B. Kostka, Ghc 2020 - google hash code 2020 repository, https://github.com/Kostero/ghc20, 2020. Accessed: 2025-05-21.
[13] E. Indzhev, Book scanning solution - google hash code 2020, https://github.com/indjev99/Optimizational-Problems-Solutions/blob/master/book_scanning.cpp, 2020. Accessed: 2025-05-21.
[14] P. Rodrigues, Hash code models, https://github.com/pedromig/hashcode-models/tree/main, 2021. Accessed: 2025-05-21.