Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

An iterated local search for the Book Scanning Problem

Driton Alija
University of Prishtina
Kadri Sylejmani
kadri.sylejmani@uni-pr.edu
University of Prishtina
Arben Ahmeti
AAB College
Uran Lajçi
University of Prishtina
Yll Berisha
University of Prishtina

Research Article

Keywords: Book Scanning Problem, Google Hash Code, iterated local search, metaheuristics, combinatorial optimization, automatic algorithm configuration

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11489610/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

An iterated local search for the Book Scanning Problem

Driton Alija [ORCID1], Kadri Sylejmani [ORCID1,2*], Arben Ahmeti [ORCID3], Uran Lajçi [ORCID1], Yll Berisha [ORCID1]

[1]Faculty of Electrical and Computer Engineering, University of Prishtina, Bregu i Diellit p.n., Prishtina, 10000, Kosovo.
[2]Center for Artificial Intelligence, University of Prishtina, Prishtina, 10000, Kosovo.
[3]Faculty of Computer Science, AAB College, Rr. Elez Berisha, Nr. 56, Zona Industriale Fushë Kosovë, Prishtina, 10000, Kosovo.

*Corresponding author(s). E-mail(s): kadri.sylejmani@uni-pr.edu;
Contributing authors: driton.alija@student.uni-pr.edu; arben.ahmeti@aab-edu.net; uran.lajqi@uni-pr.edu; yll.berisha3@student.uni-pr.edu;

Abstract

Digitizing a library network's collections before a deadline requires coordinating library sign-up and book assignment, and this plan determines how much of the collections' value is scanned. Sequential sign-up, limited scanning capacities, and overlapping collections make the two decisions interdependent, since activating one library delays the others and each book counts only once. The Book Scanning Problem of Google Hash Code 2020 captures this setting, and its contest solvers are tailored to individual instances. We propose an iterated local search that optimizes the library sign-up order and derives a feasible book assignment from each order with fast decoders. It constructs an initial solution, improves library selection and ordering by local search, and applies perturbations and restarts when progress stalls. Its main parameters are configured automatically, once for all instances. We also provide a formal model and a public benchmark of 324 instances: the official contest instances and three generated families, divided into 256 training and 68 evaluation instances. Ablations trace the solution quality to the local search and identify library insertion and replacement as the operator group with a significant effect. With the best of ten runs per instance, the method scores 2.362% more in total than the per-instance best of six existing methods and is better on 24, equal on 31, and worse on 13 of the 68 instances. On the

five official contest instances it remains competitive, 0.058% below the reference total. Paired statistical tests confirm the advantage, and a simple upper bound proves 21 solutions optimal.

Keywords: Book Scanning Problem, Google Hash Code, iterated local search, metaheuristics, combinatorial optimization, automatic algorithm configuration

1 Introduction

Large-scale digitization is not only a question of scanning technology. It is also a planning problem combining a hard deadline with library-specific sign-up times, varying daily scanning capacities, and overlapping book collections. The Book Scanning Problem posed in the Google Hash Code 2020 qualification round makes this interaction explicit (Google 2020). A library must complete a sign-up process before it can scan, only one library can be signing up at any moment, and every signed library then scans in parallel at its own daily rate. A book may be held by several libraries, but its score is earned only once. Consequently, selecting one library changes both the time available to all later libraries and the marginal value of their collections. Fig. 1 illustrates these rules on a small instance with twelve books and four libraries: the schedule shown scans ten of the books within the ten-day horizon, every book held by two libraries is scanned only once, and the last library in the order cannot complete its sign-up before the deadline and is left out.

The two coupled decisions (the library sign-up order and the assignment of books to libraries) produce a large and irregular search space. A constructive rule can identify promising libraries quickly, but an early choice may consume sign-up time or duplicate books that could have been scanned more profitably elsewhere. Exact formulations are informative on small instances, yet their library-pair and library–book decision variables grow rapidly. The state-of-the-art solvers developed for the original contest therefore rely on specialized greedy criteria, randomized local improvement, simulated annealing, satisfiability encodings, or mathematical-programming components, often tailored to individual official instances (Sayed 2020; Kostka 2020; Indzhev 2020). This leaves a gap between highly specialized contest solvers and a general, transparently analyzed method evaluated across many instance families.

This paper develops a solution method based on iterated local search (ILS), the metaheuristic framework of Lourenço et al. (2003) that alternates local optimization with perturbation. The search state is a permutation of libraries, and an order is decoded into a feasible schedule whenever a candidate must be scored in full. The separation between ordering and assignment makes a broad set of neighborhoods inexpensive to express: some moves reorder the currently contributing (active) libraries, others exchange active and inactive libraries, and a third group exploits coverage and book-rarity information. The implementation additionally separates an inexpensive screening evaluation from full reconstruction, so that expensive evaluations are spent only on promising candidates.

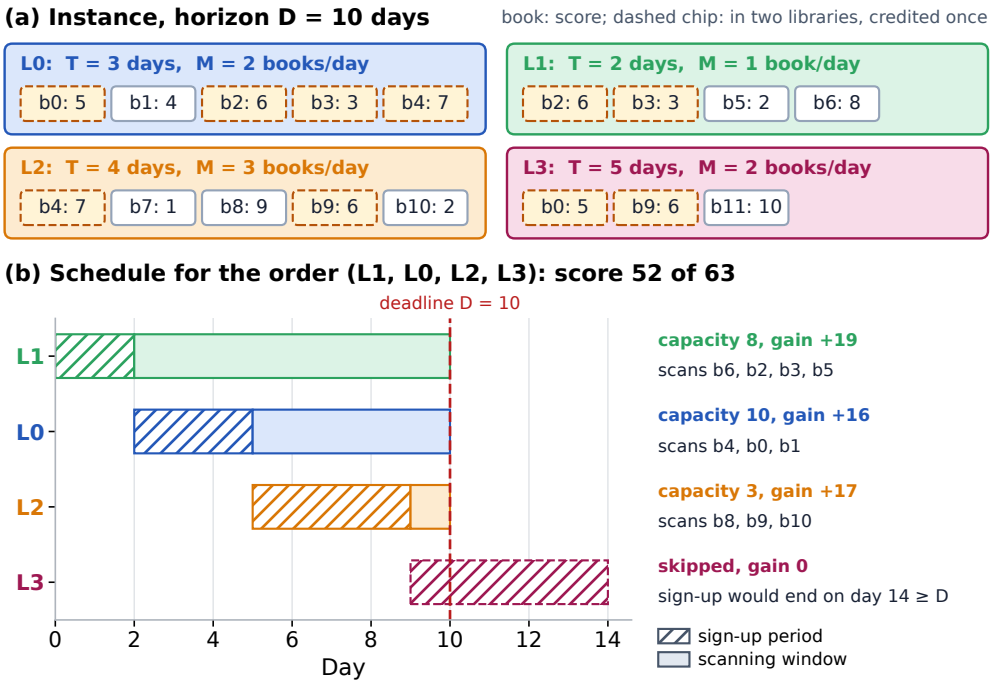


Fig. 1 A small instance of the Book Scanning Problem and one of its schedules. (a) Four libraries, each with a sign-up time T, a daily scanning rate M, and an inventory of books with scores. The horizon is $D = 10$ days. (b) The schedule obtained for the sign-up order (L1, L0, L2, L3). Libraries sign up one after another (hatched bars) and scan in parallel from the day after their sign-up completes until the deadline (filled bars). The sign-up of L3 would end on day 14, after the deadline, so L3 is skipped. The schedule scores 52 of the 63 points available. The optimum for this instance is 53.

1.1 Related work

The method proposed here builds on the ILS framework (Lourenço et al. 2003) and on the metaheuristics that share its components: variable neighborhood search (Mladenović and Hansen 1997), adaptive large neighborhood search (Ropke and Pisinger 2006), and the greedy randomized adaptive search procedure (GRASP) (Feo and Resende 1995), whose randomized greedy construction supplies its diversified starting solutions. The problem itself is related to the classical maximum coverage, orienteering, and prize-collecting problems. Problem-specific prior work consists of the public solutions developed for the Book Scanning Problem during and after the contest: the Errecto solution (Kostka 2020), the solutions of Sayed (2020) and of Indzhev (2020), and the flow-based book allocation of Rodrigues (2023a). The Errecto and Sayed solutions provide the comparison methods of this study. The automated design of heuristics with large language models, such as FunSearch (Romera-Paredes et al. 2024) and its application to Hash Code problems (Veličković et al. 2024), is a more recent direction that we regard as complementary. These groups are discussed in turn, followed by the positioning of the present method.

Iterated local search and neighborhood-based metaheuristics

ILS alternates intensive local optimization with perturbations that move the search into a new basin of attraction (Lourenço et al. 2003; Luke 2013). In practice, performance depends primarily on the solution representation and its decoder, the neighborhood operators, the acceptance rule, and the cost of evaluating a move, rather than on the basic ILS loop itself. Recent scheduling applications likewise combine problem-specific neighborhoods, restarts, and fast feasibility checking (Yahiaoui et al. 2023). Cycling among several neighborhoods relates the present method to variable neighborhood search (Mladenović and Hansen 1997), and the use of differently scaled removal and reinsertion mechanisms is central to adaptive large neighborhood search (ALNS) (Ropke and Pisinger 2006). The method proposed here is not an ALNS: it preserves a full permutation and applies comparatively small order moves. The comparison is still useful, because it identifies the insertion/replacement group (which changes the set of contributing libraries) as the mechanism that corresponds most closely, at the level of strategy, to destroy-and-repair, whereas pure reordering operates within the current set. GRASP retains greedy structure while diversifying construction: it forms a restricted list of high-quality candidates and chooses among them at random (Feo and Resende 1995). In the present method, this principle diversifies the initial solutions handed to ILS rather than replacing local search.

Related optimization problems

Because each book counts only once, the Book Scanning Problem is a coverage problem: without capacity limits, a set of signed libraries earns the score of the union of their inventories. For maximizing a coverage function over sets of bounded cardinality, the greedy algorithm guarantees at least the fraction $1 - 1/e$ of the optimum (Nemhauser et al. 1978), and no polynomial-time algorithm guarantees a larger constant fraction unless P = NP (Feige 1998). Khuller et al. (1999) attained the same factor for budgeted maximum coverage, in which sets have costs and a budget limits their total cost, with a greedy rule based on the uncovered weight per unit cost. Budgeted maximum coverage is a restricted case of the Book Scanning Problem (Sect. 2.4). That reduction preserves objective values, so the threshold of Feige (1998) also holds for the Book Scanning Problem. The priority rule of Eq. (16) follows the cost-effectiveness principle of the budgeted greedy rule, with the sign-up time as the cost. The selection and sequencing of libraries before a deadline also relate the problem to the orienteering problem, which selects and orders a subset of locations to maximize the collected score within a time budget (Vansteenwegen et al. 2011; Gunawan et al. 2016), and to the prize-collecting traveling salesman problem, in which a tour visits only some of the cities and balances the collected prizes against travel costs and penalties (Balas 1989). Unlike the scores and prizes of these problems, the value of a library is not fixed in advance. It depends on the position of the library in the sign-up order, which sets its scanning capacity, and on the books that the other signed libraries also hold. For a fixed order, the book assignment can be solved exactly as a flow problem (Rodrigues 2023a,b), so the combinatorial difficulty lies in the selection and ordering of the libraries.

Public solutions for the Book Scanning Problem

Most problem-specific prior work exists as contest implementations rather than as benchmark literature. The Errecto solution, ranked fifth in the qualification round, combines instance-specific strategies, randomized local changes to the library order, and careful greedy reconstruction (Kostka 2020). Its strategies were chosen separately for each of the five official instances, which tailors the solver to those instances but leaves the choice for an unseen instance to the user. Sayed's repository comes from the extended round that followed the four-hour qualification round of 20 February 2020 (Taub 2020) and kept the problem open for further submissions. The team ranked third of about 4,700 in that round, as the repository reported on 10 March 2020 (Sayed 2020). The repository combines a greedy constructor with mixed-integer linear programming (MILP), Boolean satisfiability (SAT), and simulated annealing, and assigns one of these components to each official instance. Indzhev's solution builds the library order greedily under a coverage-weighted score in which each book contributes its value divided by one plus the number of already selected libraries that hold it, and divides this score by the sign-up time, with instance-specific adjustments and random noise on two official instances. It then refines the order by sorting the libraries by the score each contributed and reassigning the books, until ten consecutive rounds bring no improvement (Indzhev 2020). Rodrigues implements several models of the problem behind a common interface of constructive moves, local moves, and an upper bound, on which general solvers such as GRASP, iterated greedy, beam search, and a MAX-MIN ant system can run (Rodrigues 2023a). One of these models separates the library-order decision from book allocation and solves the latter as a minimum-cost maximum-flow problem. That work shows both the benefit and the computational cost of treating assignment exactly (Rodrigues 2023a,b).

Automated heuristic design

Recent work goes beyond the automatic configuration of parameters and automates the design of the heuristic itself. FunSearch pairs a large language model that proposes small programs with an automatic evaluator that steers the search toward better-performing candidates (Romera-Paredes et al. 2024), and the idea has been applied to combinatorial problems from programming competitions (Veličković et al. 2024). In that study a human-written backbone heuristic is paired with an evolved scoring function, and the resulting solutions outperform the rank-one team on five Hash Code qualification rounds, the 2020 Book Scanning round among them. These results were obtained in a post-hoc evaluation of the contest problems. We view this direction as complementary rather than competing: the present study asks how far a carefully engineered and transparently analyzed classical metaheuristic can go on this problem, and its instrumented benchmark provides a natural reference point for future automated-design comparisons.

Positioning of the present method

Compared with contest solvers that select their strategies by hand for each official instance, the proposed method applies the same metaheuristic to every instance. It starts from the best of several constructive strategies and searches a permutation of

libraries with thirteen neighborhood operators in three groups, including structure-aware exchanges designed for this problem. A screening evaluation filters candidate moves before full rescoring, and a replacement-biased perturbation, a temperature-based acceptance criterion, and restarts from an archive of good solutions move the search out of local optima.

1.2 Contributions and organization

Our preliminary conference study (Alija et al. 2025) introduced an earlier version of the mixed-integer model of Sect. 2.3, together with early variants of ILS, a genetic algorithm, and GRASP construction, evaluated on 20 instances with five runs per method. The present work continues only the ILS line, reuses no result of that study, and is substantially broader. It introduces the multi-constructor and multi-decoder architecture, the screened two-stage local search, and the adaptive perturbation and restart mechanisms; configures the method automatically; and evaluates it on 324 instances split into training and evaluation sets. The final protocol comprises 680 evaluation runs of the tuned configuration, component and operator-group ablations under the same ten-seed protocol, a deliberately weakened initialization experiment, and 100 instrumented convergence runs.

The paper makes four contributions:

1. a mathematical model of the problem: a formal statement, a mixed-integer linear formulation that makes both decision layers explicit, and an NP-hardness argument;
2. a customized ILS for the Book Scanning Problem, built on the permutation representation, a portfolio of fast decoders, two-tier candidate evaluation, multi-strategy construction, and controlled perturbation, acceptance, and restart mechanisms, with its main parameters selected by automatic configuration;
3. a diverse set of thirteen neighborhood operators in three groups, which adapt classical permutation and set-exchange moves to the active prefix of the order and add structure-aware operators specific to this problem. In the operator-group ablation (Sect. 5.2), only the insertion/replacement group has a statistically significant effect;
4. a benchmark of 324 instances from four structurally different families, assembled from the official contest instances and three public instance sources, with a fixed training/evaluation split and archived run records, which allows configuration, evaluation, and future comparisons on a common basis.

These contributions are validated by an experimental study, built around two hypotheses stated in Sect. 4, that compares the tuned method with a demanding per-instance reference formed by the best of six external methods, using paired nonparametric tests, scale-normalized sensitivity analysis, and optimality certificates from a simple upper bound, and that explains where the solution quality comes from through component, operator-group, initialization, convergence, and seed-stability analyses.

The central claim is deliberately empirical. The proposed method is competitive across a broad benchmark and improves on the external reference in aggregate, but it does not dominate every instance, and we make no claim of new best-known values for

the official contest instances. What we consider the method's real value is a transparent, analyzable, and reproducible integration of classical metaheuristic components, together with an account of which of those components actually earn their place.

The remainder of the paper is organized as follows. Sect. 2 formalizes the problem, gives a mixed-integer formulation, and establishes its complexity. Sect. 3 describes the proposed method. Sect. 4 presents the experimental design, including the automatic configuration of the parameters, Sect. 5 the results, and Sect. 6 the discussion and limitations. Sect. 7 concludes.

2 Problem definition

This section defines the Book Scanning Problem and the notation used in the rest of the paper. Sect. 2.1 describes the rules of the problem, Sect. 2.2 states it as an optimization model, and Sect. 2.3 gives a mixed-integer linear formulation. Sect. 2.4 shows that the problem is NP-hard and introduces the simple upper bound that Sect. 5.7 uses to certify that some of the reported solutions are optimal.

2.1 Problem description

An instance specifies a set of books, a set of libraries, and a time horizon of D days, numbered $0, \ldots, D-1$. Each book has a nonnegative score that is earned at most once, regardless of how many libraries hold a copy. Each library requires a fixed number of days to sign up and, once signed, can ship a limited number of books per day to the scanning facility.

Sign-up is sequential: at any moment, at most one library can be in the sign-up process, and the next sign-up starts only after the previous one has finished. Scanning is parallel: from the day after its sign-up completes, each signed library scans at its own daily rate, and the facility itself imposes no global daily limit (Google 2020) (Fig. 1). This asymmetry is the source of the central trade-off. Every day spent signing up one library reduces the scanning window of all later libraries, so a library with a valuable collection but a long sign-up time may be less attractive than a smaller library that activates quickly and contributes rare books.

A solution chooses which libraries to sign up, in which order, and which books each of them ships. A library whose sign-up ends on day D or later can ship no books, and the submitted schedule lists only libraries that ship at least one book (Google 2020). Libraries left out of the schedule take no sign-up time.

2.2 Formal model

Let $\mathcal{B} = \{0, \ldots, B-1\}$ denote the books and $\mathcal{L} = \{0, \ldots, L-1\}$ the libraries. Book b has score $s_b \geq 0$. Library ℓ holds the subset $\mathcal{B}_\ell \subseteq \mathcal{B}$, requires T_ℓ days for sign-up, and scans at most M_ℓ books per day afterwards.

A solution determines a partial permutation

$$\pi = (\pi_1, \ldots, \pi_k), \qquad k \leq L, \tag{1}$$

of the libraries that complete sign-up. Library π_i becomes active on day

$$a_{\pi_i} = \sum_{j=1}^{i} T_{\pi_j}, \tag{2}$$

contributes only if $a_{\pi_i} < D$, and then has effective capacity

$$c_{\pi_i} = (D - a_{\pi_i})\, M_{\pi_i}. \tag{3}$$

For each active library, the schedule selects a set $X_{\pi_i} \subseteq \mathcal{B}_{\pi_i}$ with $|X_{\pi_i}| \le c_{\pi_i}$. The objective is

$$\max_{\pi, X} \; f(\pi, X) = \sum_{b \in \bigcup_{i=1}^{k} X_{\pi_i}} s_b. \tag{4}$$

The union in Eq. (4) captures the uniqueness rule: scanning a duplicate copy adds no score.

2.3 Mixed-integer formulation

A mixed-integer linear program makes both decision layers explicit. An earlier version of this model appeared in our preliminary study (Alija et al. 2025). Let $y_\ell \in \{0,1\}$ indicate that library ℓ signs up, $z_{\ell b} \in \{0,1\}$ that book $b \in \mathcal{B}_\ell$ is scanned from ℓ, and $u_b \in \{0,1\}$ that book b is scanned at least once. Let $p_{\ell q} \in \{0,1\}$ indicate that selected library ℓ precedes selected library q, and let the integer variable $\sigma_\ell \in \{0, \dots, D-1\}$ be the day on which the sign-up of ℓ starts. Sign-up times are positive integers, as in the contest data. A library with $T_\ell \ge D$ cannot complete its sign-up early enough to scan any books, so it is removed before the model is built. The removal leaves the optimal value unchanged, and $\mathcal{L}$ denotes the remaining libraries throughout this subsection. With the big-M constants $H^T = D + \max_\ell T_\ell$ and $H^C_\ell = |\mathcal{B}_\ell| + M_\ell D$, the model reads

$$\max \quad \sum_{b \in \mathcal{B}} s_b\, u_b \tag{5}$$

$$\text{s.t.} \quad z_{\ell b} \le u_b, \qquad \forall \ell \in \mathcal{L},\ b \in \mathcal{B}_\ell, \tag{6}$$

$$u_b \le \sum_{\ell:\, b \in \mathcal{B}_\ell} z_{\ell b}, \qquad \forall b \in \mathcal{B}, \tag{7}$$

$$z_{\ell b} \le y_\ell, \qquad \forall \ell \in \mathcal{L},\ b \in \mathcal{B}_\ell, \tag{8}$$

$$\sum_{b \in \mathcal{B}_\ell} z_{\ell b} \le M_\ell (D - \sigma_\ell - T_\ell) + H^C_\ell (1 - y_\ell), \qquad \forall \ell \in \mathcal{L}, \tag{9}$$

$$\sigma_\ell + T_\ell \le D + H^T (1 - y_\ell), \qquad \forall \ell \in \mathcal{L}, \tag{10}$$

$$p_{\ell q} + p_{q\ell} \ge y_\ell + y_q - 1, \qquad \forall \ell \ne q, \tag{11}$$

$$p_{\ell q} + p_{q\ell} \le 1, \qquad \forall \ell \ne q, \tag{12}$$

$$p_{\ell q} \le y_\ell, \quad p_{\ell q} \le y_q, \qquad \forall \ell \ne q, \tag{13}$$

$$\sigma_q \ge \sigma_\ell + T_\ell - H^T (1 - p_{\ell q}), \qquad \forall \ell \ne q. \tag{14}$$

Constraints (6)–(7) link the per-library scanning decisions to the unique credit variable u_b: a book may be scanned by several libraries, but the objective counts it once. Constraint (8) restricts scanning to signed libraries, (9)–(10) impose the effective post-sign-up capacity and the horizon on sign-up completion, and (11)–(14) create a non-overlapping sequential sign-up order among the selected libraries. The constant H_ℓ^C switches constraint (9) off for an unselected library: its right-hand side is then nonnegative for every admissible start day, which the preprocessing above guarantees because $T_\ell < D$ holds for every remaining library. The model has $O(B + L + \sum_\ell |\mathcal{B}_\ell| + L^2)$ variables and constraints. The library-pair variables alone become prohibitive when L reaches the thousands. We use the model as a formal specification of the problem, not as a solver in the experiments. The MILP1 and MILP2 comparison methods of Sect. 4.4 come from the Sayed code base and solve integer programs of their own, for library selection and for the book assignment of a fixed library order. BORT provides a Google OR-Tools implementation of a related MILP formulation for small cases (Berisha 2025). It shares the variable families of the formulation above but differs in its big-M constants and in its linking and ordering constraints.

2.4 Complexity and a simple upper bound

The problem combines subset selection, sequential ordering, and capacity constraints, and contains well-known NP-hard structures (Garey and Johnson 1979). More precisely, it contains budgeted maximum coverage (Khuller et al. 1999) as a restricted case. Given weighted elements, sets with positive integer costs, and a budget K, create one book per element and one library per set, let the set cost be the sign-up time, set $D = K + 1$, and give each library a daily rate no smaller than its inventory size. Any collection of sets with total cost at most K then signs up within the horizon and scans its complete union in the remaining time, and conversely the contributing libraries of any schedule respect that budget. The optimization problem is therefore NP-hard. Independently of hardness in the complexity-theoretic sense, the search space is enormous: a solution using exactly k libraries admits $L!/(L-k)!$ ordered selections, and the L libraries admit $L!$ orders before book assignment is considered. For $L = 100{,}000$, the number 100,000! has approximately 456,574 decimal digits.

A simple upper bound is

$$UB = \sum_{b \in \bigcup_{\ell \in \mathcal{L}} \mathcal{B}_\ell} s_b. \tag{15}$$

It assumes that every available book can be scanned and thus ignores sign-up times and capacities. It is generally loose and is never interpreted as an accurate optimality gap. It is nevertheless valid: whenever a solution attains it, that solution is provably optimal. Sect. 5.7 uses this property to certify a subset of the reported solutions.

3 The proposed iterated local search

The proposed method is an iterated local search that operates on a permutation of the libraries and delegates the assignment of books to a portfolio of fast decoders. It works in two phases. A construction phase generates several complete library orders under different priority rules, scores them with the decoding portfolio, rebuilds the

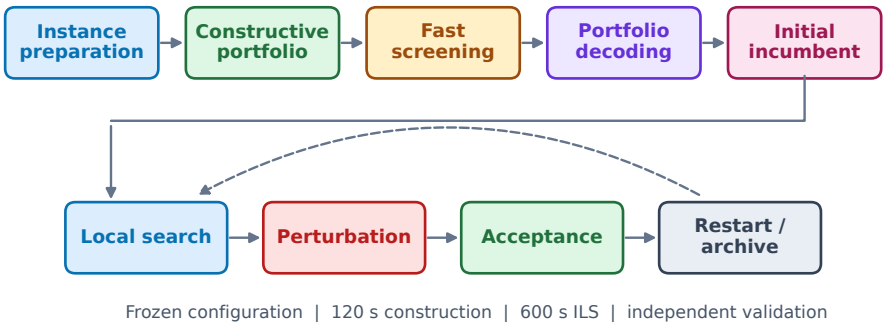


Fig. 2 Main stages of the proposed method. The upper row is the construction and initialization phase. The lower row is the iterated local search loop. Dashed feedback denotes a stagnation restart from either the archive of good solutions or a fresh construction. The 120 s construction and 600 s ILS budgets apply to the final experiments.

most promising orders in full, and adopts the best as the initial solution. The search phase then repeats rounds in which the current home solution is perturbed, improved by a two-stage local search, compared with the global incumbent, and accepted or rejected by a temperature-based criterion. When the search stagnates, it restarts from an archive of good solutions or from a fresh construction. At the end, the incumbent is exported and passed through an independent output validator (Fig. 2).

The contribution of the method lies less in the ILS loop, which is classical, than in how its components are instantiated for this problem: a representation whose local moves change the activation days and capacities of all later libraries at once (Sect. 3.1); a two-tier evaluation that spends full rebuilds only on candidates that pass a screening test (Sect. 3.2); a multi-strategy constructor (Sect. 3.3); thirteen neighborhood operators in three groups (Sect. 3.4), from which a two-stage local search samples its moves (Sect. 3.5); and perturbation, acceptance, and restart mechanisms that operate on the same representation (Sect. 3.6). The configured values of the tunable parameters of these components, the manually set constants that depend on the instance size, and the fixed budget rules of each local-search call are reported with the experimental design (Sect. 4.3).

3.1 Representation and decoding portfolio

The search state is a permutation of all libraries, read as the concatenation of the currently effective (signed) prefix and the remaining, inactive libraries. This representation has three properties that the search exploits. First, it is compact: an order is far smaller than a full day-by-day plan. Second, it is always repairable: every permutation decodes to a valid schedule. Third, its local moves have a clear interpretation: moving a library earlier changes the activation days and capacities of everything behind it, and exchanging an active library with an inactive one changes the composition of the plan.

The permutation always contains all libraries. The decoder skips any library whose sign-up would end on day D or later and continues through the order rather than stopping, because a later library with a shorter sign-up time may still be feasible. Libraries that receive no books after assignment are omitted from the exported schedule. After every rebuild the permutation is stored in canonical form, with the contributing libraries first, in their relative order, followed by the others, so that the active prefix of the search state always coincides with the decoded schedule. This convention maps every search state to a valid partial schedule while retaining inactive libraries as candidates for insertion or replacement. Fig. 3 shows the representation on the instance of Fig. 1: the home permutation with its decoded schedule, and the effect of one reordering move and one replacement move. The reordering move reaches a score of 53, which is optimal for this instance.

A decoder first scans the order to determine activation days and effective capacities, then evaluates up to three assignment rules:

1. *Sequential*: visit active libraries in sign-up order and give each its highest-scoring not-yet-assigned books until its capacity is exhausted. This is the fastest rule and matches the natural reading of the output format.
2. *Global*: process books by decreasing score and assign each to a feasible active library, preferring the library with the largest useful residual capacity and, among equals, the earlier position. This helps when a high-value book is held by several libraries.
3. *Balanced global*: process books by decreasing score but prefer the feasible library with the largest ratio of residual capacity to inventory size, avoiding the premature filling of one library when inventories overlap heavily.

The decoder returns the highest-scoring schedule among the applicable rules. All three rules are greedy: the score of the schedule that a rule builds is computed exactly, but the assignment is not guaranteed to be optimal for the given order. Throughout the paper, the *full portfolio score* of an order is this best value over the applicable rules, as opposed to the screening score of Sect. 3.2. The balanced rule is enabled only on instances with at most 3000 libraries, where its additional cost is affordable. All three rules enforce the same sign-up, capacity, membership, and uniqueness constraints. With per-library book lists preordered by score, sequential decoding takes $O(L + C)$ time in the worst case, where $C = \sum_\ell |\mathcal{B}_\ell|$ is the total number of book copies. With the global score order cached, the two global rules are also linear in $B+C$. Algorithm 1 shows the sequential rule, which doubles as the screening evaluator of the two-tier candidate evaluation described next.

3.2 Two-tier candidate evaluation

Local search requires a large number of candidate evaluations, and a full rebuild for every move would consume the entire budget. Evaluation is therefore split into two tiers. The first tier is the sequential decoder of Algorithm 1, used as an inexpensive *screening evaluator*: it scores an order with the sequential rule alone, without the two global rules and without reconstructing the schedule details, and a candidate whose screening score falls below that of the current solution is discarded. Throughout the paper, the *screening score* of a candidate denotes this first-tier value. Because the

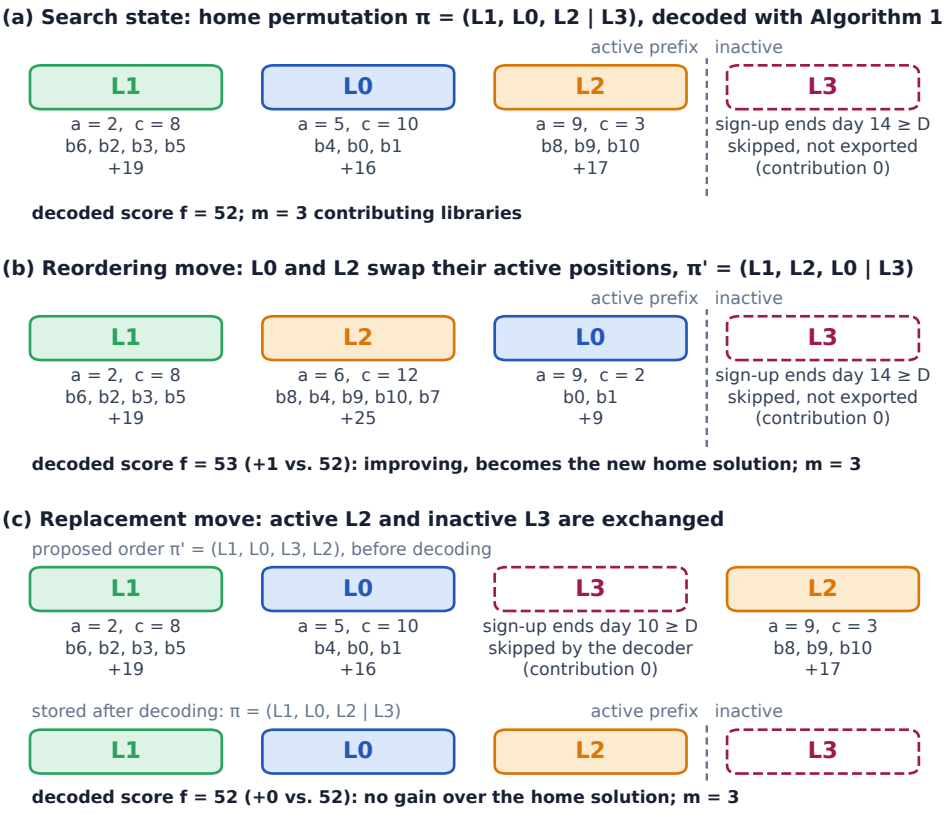


Fig. 3 The permutation representation on the instance of Fig. 1. (a) The home permutation and its decoded schedule: the first three libraries are active, whereas L3 cannot complete its sign-up before the deadline and is inactive. (b) A reordering move swaps two active positions. L2 now activates earlier with a larger capacity, and the decoded score rises from 52 to 53. (c) A replacement move exchanges the active L2 with the inactive L3 and proposes the order (L1, L0, L3, L2). The decoder skips L3, whose sign-up would end on day 10, continues with L2, and returns the same score as before. The upper row of panel (c) shows the proposed order before decoding, without an active prefix. The lower row shows the state stored after decoding, (L1, L0, L2 | L3), with L3 in the inactive suffix. In both cases the decoder, not the operator, determines which libraries contribute.

portfolio returns the best of its rules, the screening score never exceeds the full portfolio score, so the filter can discard a move that would have improved the full portfolio score. The second tier is the full rebuild through the decoding portfolio. The screening score never replaces the true objective: no candidate is accepted on first-tier evidence alone, and every surviving move is rescored with the full portfolio before it can change the search state. The role of the first tier is purely to save time.

Algorithm 1 Sequential decoding of a library order

Require: order π, scores s, inventories $\mathcal{B}_\ell$, sign-up times T_ℓ, rates M_ℓ, horizon D

$t \leftarrow 0$; $\mathcal{P} \leftarrow \emptyset$; $\Sigma \leftarrow [\,]$
for all ℓ in π **do**
  if $t + T_\ell \geq D$ **then**
    continue ▷ skipped, not terminating: a later library may still fit
  end if
  $t \leftarrow t + T_\ell$; $c \leftarrow (D - t)\, M_\ell$
  $Q \leftarrow$ the first $\min(c, |K|)$ books of $K = \mathcal{B}_\ell \setminus \mathcal{P}$, by nonincreasing score
  if $Q \neq \emptyset$ **then**
    append (ℓ, Q) to Σ; $\mathcal{P} \leftarrow \mathcal{P} \cup Q$
  end if
end for
return schedule Σ and score $\sum_{b \in \mathcal{P}} s_b$

3.3 Multi-strategy construction

No single priority rule is robust across all four instance families used in the evaluation, so the constructive stage screens the strategy families of Table 1. The adaptive variants share the priority form

$$p_\theta(\ell \mid \mathcal{P}, t) = \frac{G_\theta(\ell \mid \mathcal{P}, t)}{\max(1, T_\ell)^\alpha}, \tag{16}$$

where $\mathcal{P}$ is the set of books already covered by the libraries selected so far, t is the sign-up completion time of the current partial order, and θ indexes the gain variant: G_θ estimates the value that library ℓ would still contribute from its unassigned books that fit the remaining capacity, either at raw score (*raw*) or weighted by book rarity (*rare*). The exponent α controls how strongly long sign-up times are penalized. The constructor evaluates a pool of α values, and the pool itself is one of the configured parameters (Sect. 4.3).

All generated orders are first ranked by their full portfolio score (Sect. 3.1), computed without reconstructing the schedule. Only the three best candidates are rebuilt in full, and the best of them becomes the initial solution. An unreconstructed score alone can therefore never determine the reported starting point. The constructive stage also fills an archive of good candidate orders that later restarts may reuse. A restart that rebuilds from scratch uses the same portfolio, or a single randomized heap construction when its reconstruction budget is short (Sect. 4.3).

3.4 Neighborhood operators

The local search draws its moves from thirteen operators that form three methodological groups (Table 2). *Reordering* moves optimize the relative order of the libraries already selected. They are the classical permutation neighborhoods (pairwise swap, adjacent swap, single-element relocation, segment reversal, and block relocation) restricted to the active prefix of the order. *Insertion/replacement* moves change which libraries contribute. They adapt the exchange neighborhoods of set-selection problems to the ordered setting, so that a library entering the active set also receives a

Table 1 Constructive strategies screened before ILS starts.

Strategy	Decision rule	Type
Sorted	Increasing sign-up time, ties broken by larger total inventory score. Simple and fast; also reused as the deliberately weak start of Sect. 5.3.	determ.
Static greedy	One capacity-limited priority per library, computed once before ordering; raw and rarity-weighted variants.	determ.
Adaptive heap greedy	Lazy marginal-gain updates on a max-heap: a library is re-evaluated only when it surfaces, so that already covered books lose value without rescoring all libraries; the top-book raw variant sweeps the configured α pool, and the capacity-limited raw and rare variants use one value of the pool (Sect. 4.3).	determ.; stoch. on restarts
Fill greedy	Prefers libraries that can actually fill their remaining capacity with useful books, not only those with high nominal value; raw and rare variants.	determ.
Branch fill greedy	Keeps a loose upper bound per library and computes exact marginal value only for candidates whose bound is competitive; enabled for $L \leq 3000$.	determ.
Noisy heap	Adaptive heap with at most 4% multiplicative priority noise, producing alternative orders near the greedy one.	stoch.
GRASP	Restricted candidate list of the top-ranked libraries (ratio set by the configuration), sampled uniformly (Feo and Resende 1995). Each step scores at most 64 of the remaining libraries and stops at 12 candidates before the ratio is applied.	stoch.

position, and one of them (critical-path insert) additionally ranks its candidates by rarity-weighted value per sign-up day. *Structural* moves are problem-specific operators designed for this method rather than adapted from general neighborhoods. They exploit instance information: the decoded contribution of each active library, the rarity-weighted value of the books not yet scanned, unique coverage, and a detected paired-choice structure. Together, the three groups let a single search reorder the active libraries, change which libraries contribute, and target the structure of the instance, and the operator-group ablation of Sect. 5.2 measures what each group adds. The base weight w_0 of each operator is multiplied by the configured weight of its group (Table 7) and normalized into sampling probabilities.

Let m denote the number of currently active libraries, so positions $0, \ldots, m-1$ form the effective prefix and the remaining positions hold inactive candidates. Unless a candidate rule is stated explicitly, indices are sampled uniformly at random. Every operator returns a complete library permutation. The decoder then reconstructs the book assignment and removes non-contributing libraries, so no move can bypass feasibility checking. An operator whose preconditions are not met returns no candidate.

The two conditional structural operators are activated only when their structural tests succeed. An instance is classified as uniform only when book scores, sign-up times, and scan rates are all constant. The paired-choice test additionally requires an even number of libraries, a feasible one-from-each-pair selection, and inventories that induce valid two- or three-literal book clauses. When the tests fail, these operators are disabled and their weight is redistributed. When the structure is present, they receive an instance-adjusted share.

Table 2 The local-search operators. The implementation and the archived logs use the same names, written with underscores in place of spaces. w_0 is the base sampling weight, multiplied by the configured group weight before normalization.

Operator	Group	w_0	Candidate move
Swap signed	Reorder	2.20	Choose two active positions and exchange their libraries.
Move signed	Reorder	2.40	Remove one active library and reinsert it at another active position.
Swap neighbor libraries	Reorder	0.80	Exchange the libraries at a uniformly sampled adjacent pair of active positions.
Reverse segment	Reorder	1.30	Choose two active endpoints and reverse the inclusive subsequence between them.
Block reinsert	Reorder	1.90	Remove a contiguous active block of size $\min\{8, \max(2, \lfloor m/10 \rfloor)\}$ and reinsert it at another active position without changing its internal order.
Swap signed with unsigned	Ins./repl.	1.60	Choose one active and one inactive position and exchange their libraries.
Insert library	Ins./repl.	1.10	Remove a uniformly sampled inactive library and insert it at a position from 0 through m.
Remove library	Ins./repl.	2.80	Remove a uniformly sampled active library and append it to the inactive part of the permutation, freeing sign-up time.
Critical-path insert	Ins./repl.	0.70	Among at most 12 inactive libraries with the shortest sign-up times, select the library with the largest rarity-weighted book value per sign-up day and insert it within the first third of the active prefix.
Diversity swap	Struct.	0.60	Replace the active library with the smallest decoded contribution by the best of at most 24 sampled inactive libraries, ranked by rarity-weighted value of books not yet scanned per sign-up day.
Sampled best exchange	Struct.	0.45	Form up to 24 exchanges between six weak active positions and at most 24 ranked inactive candidates, score each with the screening evaluator, and return the exchange with the best improving screening score.
Coverage exchange	Struct.	0.20	On uniform instances, remove a small sample of libraries with low unique coverage, refill by uncovered book value, and apply one-exchange coverage improvement.
Paired-choice flip	Struct.	0.10	On detected paired-choice instances, represent the choice from each library pair as a binary variable and flip choices that increase the number of covered book clauses before rebuilding the order.

3.5 Two-stage local search

Each local-search call reserves at least 15% of its time budget for full-evaluation refinement. In the first stage, an operator is sampled according to the normalized weights and proposes a neighboring order. The screening evaluator scores the order. A move whose screening score decreases is discarded, and every surviving move is verified with the full portfolio score and fully rebuilt before acceptance. The second stage uses full portfolio scoring only: it samples broader order neighborhoods and then applies deterministic cleanup to adjacent positions, short prefix relocations, and exchanges across the active/inactive boundary, acting as a final polish of the current candidate. The screening stage stops after a configured number of consecutive non-improving trials (the local non-improvement limit of Table 7) or when the time budget of the call, set by the size profile of Table 8, is exhausted. The full-evaluation refinement then runs until that budget ends.

3.6 Perturbation, acceptance, and restart

The perturbation is intended to displace the search from the current basin of attraction, not to destroy the solution: it applies a change of strength κ to the home solution. It has two modes. In *replacement* mode, it removes $\max(1, \lfloor \kappa/2 \rfloor)$ weak active libraries, ranked by contributed score per sign-up day, and inserts highly ranked inactive candidates in their place. In *reordering* mode, it reverses or re-ranks an active segment of $\max(3, \kappa)$ libraries, or shuffles one of $\max(2, \kappa)$ libraries, within the segment caps of Table 8. The strength responds to stagnation. When the last m outer rounds have not improved the home solution, it is

$$\kappa = \max\bigl(2, \min(\kappa_{\max},\, \kappa_0 + \eta\, m)\bigr), \qquad \kappa_{\max} = \max\bigl(c, \lfloor n/d \rfloor\bigr), \tag{17}$$

where κ_0 is the base strength, η the growth per stagnant round, n the number of active libraries, and (c, d) the constants of the size profile (Table 8). The counter m returns to zero when the home solution improves and after every restart. The base strength κ_0, the growth η, and the probability of the replacement mode are configured parameters (Table 7).

An improving candidate always becomes the new home solution. A worse candidate with relative loss

$$\delta = \frac{f(S_{\text{home}}) - f(S')}{f(S_{\text{home}})} \tag{18}$$

is accepted, as in simulated annealing (Kirkpatrick et al. 1983), with probability

$$\Pr(\text{accept}) = \exp\!\left(-\frac{\delta}{\tau(r)}\right), \qquad \tau(r) = \tau_0\, \gamma^{\,r}, \qquad \tau_0 = \frac{0.01}{-\log p_0}, \tag{19}$$

where $r \in [0, 1]$ is the elapsed fraction of the ILS budget and p_0 and γ are configured parameters. The parameter p_0 has an interpretable meaning: it is the initial acceptance probability of a candidate that is exactly 1% worse than the home solution. The parameter γ is the ratio of the final to the initial temperature.

Algorithm 2 The proposed iterated local search

Require: instance I, construction budget T_0, ILS budget T

$(S, A) \leftarrow$ MultiConstruct(I, T_0) ▷ Sect. 3.3; A is the candidate archive
$S^\star \leftarrow S$; $S_{\text{home}} \leftarrow S$; $h \leftarrow 0$
while elapsed ILS time $< T$ **do**
  $S' \leftarrow$ Perturb(S_{home})
  $S' \leftarrow$ TwoStageLocalSearch(S') ▷ screening, then full verification
  if $f(S') > f(S^\star)$ **then**
    $S^\star \leftarrow S'$
  end if
  if $f(S') \geq f(S_{\text{home}})$ **or** Eq. (19) accepts **then**
    if $f(S') > f(S_{\text{home}})$ **then**
      $h \leftarrow 0$
    else
      $h \leftarrow h + 1$
    end if
    update S_{home} and archive A
  else
    $h \leftarrow h + 1$
  end if
  if the protected restart threshold is reached **then**
    $S_{\text{home}} \leftarrow$ an archived or freshly constructed solution
    $S_{\text{home}} \leftarrow$ TwoStageLocalSearch(S_{home})
    if $f(S_{\text{home}}) > f(S^\star)$ **then**
      $S^\star \leftarrow S_{\text{home}}$
    end if
    update archive A; $h \leftarrow 0$
  end if
end while
export $S^\star$ and validate the output
return $S^\star$

A restart is considered after a configured number of stagnant outer rounds, where a round is stagnant when it does not improve the home solution. To avoid discarding a nearly optimal trajectory, the threshold is protected when the home solution lies within 0.3% of the global incumbent: the configured threshold is then tripled, or raised by two rounds, whichever gives the larger value. A restart builds a fresh constructive solution with a configured probability and otherwise samples from the archive A filled during construction, which holds both the best construction candidates and a bounded set of accepted home solutions. The reconstruction budget of a restart is capped at a configured share of the remaining time and at the absolute cap of Table 8. Algorithm 2 shows the complete control flow.

4 Experimental design

The experiments evaluate the solution quality of the configured ILS relative to six external methods on instances held out from configuration, and examine how its components contribute to that quality. They test two hypotheses:

H1 With its final configuration fixed before the ten-seed evaluation runs, the ILS scores higher than the per-instance best of the six external methods on the evaluation instances, both in total score and in the paired per-instance comparison.

H2 Within the final configuration, running the iterated loop for the full ILS budget yields higher per-instance run means than a single local-search pass applied to the constructed solution.

H1 is tested on best-of-ten scores, with run means as a robustness check (Sects. 5.5–5.6), and H2 on per-instance run means (Sect. 5.1). The remaining analyses are explanatory questions rather than hypotheses. They examine which diversification mechanisms and operator groups contribute within the final configuration (Sects. 5.1 and 5.2), how strongly the final quality depends on the initial construction (Sect. 5.3), and how the search progresses over time and across seeds (Sect. 5.4).

This section presents the benchmark and its training/evaluation split (Sect. 4.1), characterizes the instances (Sect. 4.2), and describes the parameter configuration and execution protocol (Sect. 4.3), the comparison methods (Sect. 4.4), and the metrics and statistical tests (Sect. 4.5).

4.1 Instances and evaluation split

We conducted the experiments on 324 unique benchmark instances from four families. The five official Google Hash Code instances are kept exclusively for evaluation, because the public comparison methods were designed directly for them. The remaining families are *real-world-based* instances produced with the Book Scanning Problem Input Generator, ranging from micro cases to instances with two million books (Gjoka and Morina 2026); *seed-based* instances generated from the official instance structures with controlled seeds (Alija 2026b); and a broad *synthetic* collection (Lajçi 2026b). Within each generated family, instances follow a deterministic catalog order. Every fifth instance is assigned to the evaluation set and the other four to training. The resulting evaluation set is fixed in the repository. Throughout the paper, *evaluation set* refers to this set, and *test* is reserved for the statistical tests of Sect. 4.5. Table 3 reports the split and Table 4 the structural ranges. Sect. 4.2 summarizes the training instances by structural group and lists the characteristics of every evaluation instance individually. As Fig. 4 shows, the real-world-based family stretches over five orders of magnitude in book count, the seed-based and Google Hash Code families concentrate near 10^5 books but vary by orders of magnitude in library count, and the synthetic family fills intermediate regimes. This heterogeneity motivates reporting family totals and per-instance paired statistics in addition to grand totals.

Table 3 Benchmark split and final ILS evaluation runs (ten seeds per evaluation instance).

Family	Training	Evaluation	Total	Final runs
Google Hash Code	0	5	5	50
Real-world-based	36	9	45	90
Seed-based	108	27	135	270
Synthetic	112	27	139	270
Total	256	68	324	680

Table 4 Structural ranges of the complete 324-instance catalog. Entries are minimum/median/maximum.

Family (n)	Books	Libraries	Days
Google Hash Code (5)	78.6k / 100k / 100k	100 / 1k / 30k	200 / 1k / 100k
Real-world-based (45)	50 / 18k / 2M	4 / 55 / 2k	4 / 60 / 280
Seed-based (135)	44.2k / 73.5k / 100k	61 / 888 / 32.7k	59 / 6.8k / 100k
Synthetic (139)	1.3k / 97.2k / 100k	1 / 5.4k / 29.3k	110 / 35.2k / 100k

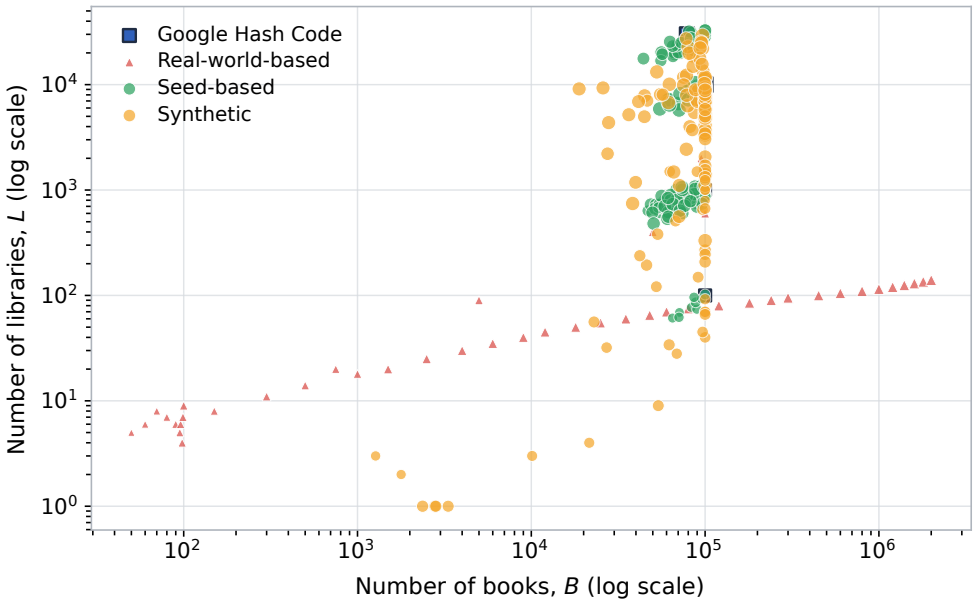


Fig. 4 Structural scale of all 324 instances. Both axes are logarithmic. Marker area encodes the time horizon.

4.2 Instance characteristics

The families differ by orders of magnitude in size and in structure, and the results of Sect. 5 are read by family and by instance. This subsection therefore characterizes the instances beyond the ranges of Table 4. Table 5 summarizes the 256 training instances used for parameter configuration, grouped by the most representative characteristic of each family: real-world-based instances by collection size (number of books), seed-based instances by the base official structure from which they were extended (types `b`–`f`, the second letter of a hybrid name), and synthetic instances by quartiles of the library count (Q1–Q4). The min–max intervals are the actual values within each group, so consecutive boundaries are not necessarily contiguous. Medians are reported because some instances have extreme values that would distort mean-based summaries. Table 6 then lists the structural characteristics of each of the 68 evaluation instances, against which the per-instance results of Sect. 5.8 can be read. The per-instance catalog of the training instances is part of the repository data (Alija 2026c).

Table 5: Grouped summary of the 256 training instances used for parameter configuration. Ranges are minimum–maximum within the group. Copies, Score, Freq., Signup, and Cap. are group medians of the total book occurrences and of the per-instance average book score, book frequency, library sign-up time, and daily library capacity.

Group	n	Books min–max	Libr. min–max	Days min–max	Copies med.	Score med.	Freq. med.	Signup med.	Cap. med.
Real-world-based									
Micro	12	50–750	4–20	4–20	202	63.22	2.54	7.46	15.79
Small	8	1,500–25,000	20–90	21–110	35,348	487.96	5.25	21.70	206.36
Medium	8	48,000–180,000	65–2,000	28–170	260,381	612.48	2.97	29.70	596.70
Large	8	300,000–1,800,000	95–135	190–270	649,124	744.98	1.37	39.33	1,566.78
Seed-based									
Type `b`	8	64,950–100,000	61–102	59–229	60,843	100.00	1.77	11.17	1.00
Type `c`	15	54,838–100,000	5,746–10,925	99,999–100,000	92,805	300.77	1.96	505.72	49,011.99
Type `d`	21	44,181–100,000	16,877–32,747	2,025–9,842	116,653	100.00	2.57	1.50	1.00
Type `e`	29	46,779–100,000	603–1,079	238–1,078	287,806	100.00	4.35	4.99	1.25
Type `f`	35	49,720–100,000	481–1,081	7,891–33,678	281,410	125.49	4.26	165.71	6.96
Synthetic									
Q1	28	1,783–100,000	1–667	110–86,428	251,460	354.25	5.02	25.00	476.50
Q2	28	27,464–100,000	795–5,181	160–85,923	952,454	290.92	522.99	258.00	21,127.50
Q3	28	41,394–100,000	5,332–8,504	6,435–100,000	999,888	291.00	1,215.60	275.00	25,110.00
Q4	28	52,700–100,000	8,649–29,302	13,312–100,000	252,410	263.33	516.85	276.50	27,278.00

Table 6: Structural characteristics of the 68 evaluation instances, grouped by family (the `.txt` suffix of instance names is omitted). Copies is the total number of book occurrences across libraries; Score, Freq., Signup, and Cap. are per-instance averages of the book score, book frequency, library sign-up time, and daily library capacity.

Instance	Books	Libr.	Days	Copies	Score	Freq.	Signup	Cap.
Google Hash Code								
`b_read_on`	100,000	100	1,000	100,000	100.00	1.00	11.81	1.00
`c_incunabula`	100,000	10,000	100,000	150,282	300.76	1.94	506.08	49,511.38
`d_tough_choices`	78,600	30,000	30,001	220,800	65.00	2.81	2.00	1.00
`e_so_many_books`	100,000	1,000	200	491,942	125.49	4.96	5.43	1.49
`f_libraries_of_the_world`	100,000	1,000	700	509,342	401.11	5.12	166.19	7.47
Real-world-based								

Continued on next page

Table 6 (continued)

Instance	Books	Libr.	Days	Copies	Score	Freq.	Signup	Cap.
B1000k_L115_D230	1,000,000	115	230	653,038	750.18	1.36	41.38	1,574.59
B100_L9_D18	100	9	18	388	62.46	3.88	9.44	20.00
B1k_L18_D17	1,000	18	17	4,307	285.16	4.33	15.94	64.61
B2000k_L140_D280	2,000,000	140	280	895,057	800.03	1.24	42.99	1,701.08
B240k_L90_D180	240,000	90	180	460,268	674.81	2.24	33.69	1,289.23
B35k_L60_D120	35,000	60	120	192,290	589.68	5.51	30.53	534.25
B6k_L35_D70	6,000	35	70	36,458	489.69	6.08	25.17	185.46
B90_L6_D11	90	6	11	155	50.32	2.09	5.00	11.50
B90k_L850_D21	90,000	850	21	25,183	19.97	1.20	17.51	54.16
Seed-based								
b_075x_012t	89,643	74	98	55,768	100.00	1.69	11.04	1.00
bc_060x_008t	60,743	7,110	100,000	60,713	100.00	1.82	503.72	49,028.59
bc_075x_020t	91,720	8,885	100,000	95,960	100.00	1.87	506.13	49,232.23
bd_060x_012t	66,395	20,186	3,623	79,186	100.00	2.24	1.50	1.00
bd_090x_008t	88,685	26,623	3,190	162,867	100.00	2.73	1.50	1.00
be_060x_020t	64,150	629	620	187,171	100.00	3.41	4.93	1.27
be_090x_012t	100,000	1,014	608	450,815	100.00	4.87	5.00	1.24
bf_075x_008t	77,459	770	10,139	293,849	100.00	4.19	164.59	6.93
bf_090x_020t	84,605	909	29,937	412,032	100.00	5.17	164.67	6.95
c_075x_012t	91,207	8,367	100,000	89,990	300.77	1.74	506.60	48,850.90
cd_060x_008t	65,305	18,474	2,214	72,171	300.77	2.11	1.50	1.00
cd_075x_020t	85,411	22,535	6,754	113,408	300.78	2.22	1.50	1.00
ce_060x_012t	69,662	649	396	192,923	300.77	3.17	5.09	1.25
ce_090x_008t	100,000	971	387	427,607	300.77	4.52	4.99	1.25
cf_060x_020t	57,420	680	22,392	206,366	300.78	3.89	164.65	6.98
cf_090x_012t	99,017	899	18,013	403,963	300.78	4.34	166.98	7.05
d_075x_008t	68,867	23,422	2,818	117,900	65.00	2.67	1.50	1.00
d_090x_020t	82,871	27,543	8,253	168,401	65.00	2.94	1.50	1.00
de_075x_012t	63,095	755	458	280,759	65.00	4.83	5.06	1.28
df_060x_008t	48,721	729	9,732	220,756	65.00	4.89	166.88	6.90
df_075x_020t	63,536	895	29,690	337,407	65.00	5.62	165.87	6.95
e_060x_012t	66,585	672	414	202,940	125.49	3.35	5.14	1.25
e_090x_008t	100,000	955	386	424,400	125.49	4.44	5.06	1.26
ef_060x_020t	58,606	701	23,498	215,793	125.49	3.91	167.61	6.93
ef_090x_012t	74,028	1,005	20,075	461,449	125.49	6.33	166.46	6.99
f_075x_008t	61,188	841	11,102	321,719	401.12	5.38	165.02	7.02
f_090x_020t	73,488	1,064	35,524	486,927	401.12	6.71	166.94	6.95
Synthetic								
synthetic_10	84,402	8,802	482	999,983	142.12	2,754.77	426.00	20,498.00
synthetic_100	100,000	5,327	28,757	999,868	266.76	1,599.79	336.00	35,204.00
synthetic_105	27,116	32	716	119,838	318.61	4.46	26.00	90.00
synthetic_110	18,843	9,111	46,751	999,983	785.74	53.07	400.00	48,268.00
synthetic_115	100,000	7,054	88,048	999,875	437.44	1,240.54	480.00	28,867.00
synthetic_120	100,000	5,081	82,258	999,869	398.43	2,096.16	188.00	21,200.00
synthetic_125	100,000	6,706	54,523	999,889	344.13	1,062.58	235.00	27,124.00
synthetic_130	100,000	3,352	91,540	999,871	284.66	1,298.53	53.00	19,313.00
synthetic_135	100,000	6,227	36,992	740,157	233.59	805.39	455.00	41,302.00
synthetic_15	1,271	3	130	2,433	41.34	2.03	4.00	166.00
synthetic_20	38,270	745	73,522	566,103	810.93	14.79	485.00	18,770.00
synthetic_25	100,000	4,832	60,494	999,845	340.77	1,390.61	235.00	13,181.00
synthetic_30	3,324	1	1,519	2,565	890.94	1.00	151.00	10.00
synthetic_35	100,000	1,111	64,501	239,657	247.75	361.47	35.00	44,471.00
synthetic_40	100,000	11,426	50,649	241,386	363.50	382.55	520.00	34,595.00
synthetic_45	25,859	9,307	44,152	999,994	542.88	48.19	145.00	40,128.00
synthetic_5	46,509	7,033	6,753	747,354	193.86	4,982.36	52.00	26,366.00
synthetic_50	96,585	22,085	88,451	288,681	294.67	596.45	294.00	14,331.00
synthetic_55	100,000	7,248	56,396	766,722	272.52	3,262.65	404.00	6,604.00
synthetic_60	77,949	2,437	64,294	999,991	431.39	12.83	436.00	17,324.00
synthetic_65	95,490	24,914	52,437	225,156	224.33	386.20	152.00	4,129.00
synthetic_70	100,000	1,544	8,584	501,031	365.80	690.13	329.00	37,090.00
synthetic_75	100,000	7,578	33,769	999,995	276.46	1,915.70	112.00	31,409.00
synthetic_80	57,016	8,031	8,548	999,957	646.13	4,566.01	360.00	6,673.00
synthetic_85	2,823	1	1,671	2,823	920.18	1.00	52.00	8.00

Continued on next page

Table 6 (continued)

Instance	Books	Libr.	Days	Copies	Score	Freq.	Signup	Cap.
synthetic_90	78,361	7,931	58,347	175,090	386.14	383.13	429.00	34,318.00
synthetic_95	99,079	10,191	100,000	171,070	348.01	328.98	395.00	38,226.00

4.3 Parameter configuration and experimental protocol

The tunable parameters of the ILS of Sect. 3, which govern acceptance, perturbation, restarts, the local search, construction, and the weights of the operator groups, were configured with iRace (López-Ibáñez et al. 2016) on the 256 training instances (Sect. 4.1). iRace applies iterated racing: candidate configurations are run on successive training instances, and those that are statistically worse are discarded early, so that the evaluation budget concentrates on the promising ones. The race had a budget of 5,000 target experiments, six parallel workers, and 300 s of ILS time per target run. The final elite of the race, that is, the best surviving configuration, is used unchanged in all final experiments. Table 7 lists the selected values, the search domains, and the role of each parameter. The selected constructor exponent pool, *explore*, corresponds to $\alpha \in \{0.4, 0.5, 0.75, 1.0, 1.5, 2.0\}$ in Eq. (16), and the capacity-limited constructor variants use its fourth value, $\alpha = 1.0$. For the perturbation, iRace selected the lower end of the base-strength domain, $\kappa_0 = 2$, and no growth, $\eta = 0$. The strength of Eq. (17) is therefore constant: each perturbation exchanges a single library or rearranges a segment of two or three libraries. With the selected restart threshold of three stagnant outer rounds, the tuned configuration keeps every perturbation local and assigns the response to prolonged stagnation to the restart mechanism.

The configuration protocol included a post-selection stage. After the racing phase, each of the five final elite configurations was evaluated once on the 68 evaluation instances, and the best of the five was retained. This stage did not change the outcome, because the elite of the training race was also the best of the five. It does mean that the evaluation set was used once before the final runs, so it is not a fully blinded holdout. All headline results come from the subsequent ten-seed protocol.

The tuned configuration is interpreted as a validated outcome of a controlled selection process, not as proof that each individual parameter is better than every manual alternative: such a claim would require a symmetric head-to-head protocol against manual configurations, which was not part of this design. Consistent with this reading, the ablations of Sects. 5.1–5.2 start from the tuned configuration and disable one mechanism at a time. They measure component contributions within the final configuration, not the effect of tuning itself.

The implementation also uses constants that were set manually before tuning. Those stated where they are used in Sect. 3 are the share of each local-search call reserved for full-evaluation refinement, the protection margin of the restart threshold, the noise levels of the constructors, the candidate-list sizes of the operators, and the library-count thresholds that enable the balanced decoding rule and the branch fill constructor. A further set of manually set constants depends on the instance size. Each instance is assigned to one of four size profiles by the rules in the caption of Table 8, and the profile sets the time budget of each local-search call, the size of the archive of accepted home solutions, the absolute cap on a restart construction, and the caps

Table 7 Automatically configured parameters: selected value, iRace search domain, and role.

Parameter	Value	Domain	Role
Initial acceptance probability p_0	0.0320	$(0.001, 0.15)$	Initial probability of accepting a candidate that is 1% worse, Eq. (19).
Temperature ratio γ	0.1699	$(0.01, 0.25)$	Final/initial temperature ratio of the acceptance criterion.
Restart threshold	3	$[2, 12]$	Stagnant outer rounds before a restart is considered.
Base perturbation strength κ_0	2	$[2, 15]$	Perturbation strength without stagnation, Eq. (17).
Perturbation growth η	0	$[0, 6]$	Increase of the strength per stagnant outer round (none selected).
GRASP restricted-list ratio	0.1308	$(0.01, 0.25)$	Fraction of top-ranked libraries eligible for random greedy selection.
Restart construction budget ratio	0.1491	$(0.10, 0.50)$	Share of remaining time a restart may spend on fresh construction.
Replacement perturbation probability	0.6794	$(0.30, 0.90)$	How often perturbation replaces weak libraries rather than reordering a segment.
Fresh-restart probability	0.4035	$(0.10, 0.70)$	Probability that a restart constructs anew instead of reusing the archive.
Local non-improvement limit	316	$[50, 500]$	Consecutive failed local trials before the local phase stops.
Reordering group weight	0.6317	$(0.40, 2.50)$	Relative weight of the reorder operator group.
Insertion/replacement group weight	2.3634	$(0.40, 2.50)$	Relative weight of moves that change the active set.
Structural group weight	1.9964	$(0.40, 2.50)$	Relative weight of the structural operator group.
Constructor exponent pool	explore	{default, wide, dense, signup, explore}	Categorical pool of α values for Eq. (16).

on segment lengths and perturbation strength. Each local-search call is additionally limited to a share of the remaining ILS budget (18% for the initial call, 8% per round, and 12% after a restart), and the local search after a restart uses a non-improvement limit equal to the larger of the profile floor and half the configured limit. The profile also carries default values of the restart threshold, the perturbation strength and its growth, and the local non-improvement limit, which the configured values of Table 7 override. A fresh restart keeps the constructor portfolio of Table 1 unless its reconstruction budget is at most eight seconds. It then runs a single randomized heap construction whose noise level is drawn from $[0.03, 0.10]$.

Each final run uses at most 120 s for construction and 600 s for the ILS loop. Reported wall-clock times additionally include export and validation. The tuned

Table 8 Manually set constants of the instance-size profile. With L libraries, D days, and $O = \sum_\ell |\mathcal{B}_\ell|$ book occurrences, an instance is huge if $L \geq 15{,}000$, $O \geq 700{,}000$, or $D \geq 50{,}000$; otherwise large if $L \geq 3000$, $O \geq 300{,}000$, or $D \geq 10{,}000$; otherwise medium if $L \geq 500$, $O \geq 100{,}000$, or $D \geq 1000$; otherwise small. An instance with $L \leq 250$ and $D \leq 1000$ is never large or huge. n is the number of active libraries.

Constant	Small	Medium	Large	Huge
Local-search time per call, initial/round/restart (s)	3.2/1.6/2.0	2.6/1.3/1.6	1.8/0.9/1.2	1.1/0.5/0.7
Archive of accepted home solutions	8	6	5	4
Restart construction cap (s)	60	45	30	20
Restart non-improvement floor	140	150	120	80
Segment cap, reorder/shuffle	10/6	14/8	18/9	20/10
Perturbation strength cap $\max(c, \lfloor n/d \rfloor)$, as (c, d)	(4, 8)	(5, 7)	(6, 6)	(8, 5)

configuration is evaluated with the ten seeds

$$\{1, 12, 23, 34, 45, 56, 67, 78, 89, 100\},$$

which control all stochastic choices in construction, local search, perturbation, and restart. The method is implemented in Python 3.11.2. Instance data are stored as flat numerical arrays rather than object graphs, and the numerical kernels are just-in-time compiled with Numba (Numba Developers 2026). The experiments ran on a Google Cloud Compute Engine compute-optimized virtual machine of type `c2-standard-8`, running Linux with 8 vCPUs and 32 GiB of RAM. Up to six runs executed in parallel, which shortens the total experiment time without changing the per-run budget. Every one of the 680 final runs used its full ILS budget. Every output is checked by an independent validator for format errors, repeated libraries or books, book membership, daily capacity, and the sign-up horizon. Only runs with valid outputs enter the analysis. Each result row archives the validation status, method, experimental variant, instance, family, seed, initial score, final score, simple upper bound, wall-clock time, and technical metadata.

4.4 Comparison methods

Two independent open-source contest solutions provide the comparison code. The first originates from Sayed's third-ranked extended-round solution and exposes Greedy, two MILP variants, SAT, and simulated annealing (SA) (Sayed 2020; Alija 2026a). The second is the qualification solution Errecto, which placed fifth of 10,725 teams (Avenel 2022; Kostka 2020; Lajçi 2026a). The adapters restructure input parsing, output writing, and experiment orchestration. The Sayed adapter also modifies MILP preprocessing and solution extraction, including the removal of repeated library indices, and adds heuristic branches for instances beyond those targeted by the original procedures. Greedy is the constructive baseline of that code base; MILP1 formulates the library-selection problem exactly, while MILP2 is a more general optimizer with a large iteration budget suited to bigger instances, both solved with SCIP (Achterberg 2009); SAT uses a Boolean satisfiability encoding solved with CryptoMiniSat (Soos et al.

2009); SA applies simulated annealing with adaptive refinement. Errecto consists of three separate programs, which its authors assigned by hand to the official instances: a two-phase local search over the library order and two greedy constructions written for `c_incunabula.txt` and `d_tough_choices.txt`. No rule in the code chooses among them for a given instance, so we run all three programs on every instance, official or generated, the local search with two seeds, and record the best of the four scores as the Errecto result. Every run of Greedy, MILP1, SAT, and SA is capped at 600 s of wall-clock time per instance, the same limit that the ILS search phase receives; a run that reaches the cap is terminated and recorded without a score. The runner enforces the cap by terminating the solver process. SCIP receives no time limit of its own, and the adapted code extracts and checks a schedule only after the solver returns, so a MILP run stopped at the cap leaves no checked schedule whose score could be recorded. A score therefore counts only when the method returns a complete schedule within the limit. MILP2 was not held to this cap on every instance, and some of its runs took longer than 600 s. MILP1 has valid results on 41 of the 68 evaluation instances and MILP2 on 56. Each Errecto run terminates on its own well inside the same limit, and all other methods cover the full set.

For each instance, the *external portfolio* is defined as the highest valid score among Errecto, Greedy, MILP1, MILP2, SAT, and SA. This is a deliberately demanding reference, because its constituent method can change from instance to instance. It is not an algorithm a practitioner could run at the cost of one constituent. Each comparison method, and each of the four Errecto runs, is executed once per instance: they are deterministic given their fixed seed (SA uses controlled randomness but reproduces the same result on every repetition), whereas the stochastic ILS receives ten seeds to expose its variability. Every portfolio value comes from a run that ended within the 600 s limit of the ILS search phase. The ILS additionally spends up to 120 s on construction before that phase.

4.5 Metrics and statistical analysis

The primary metric is the final score. For ILS, the per-instance comparison value is the best of the ten runs (*best-of-ten*), which measures the quality achievable with multiple independent starts. The per-instance run mean is reported alongside as the fairer estimate of expected single-run quality. Aggregate advantage over a comparison C is

$$\Delta(C) = 100\,\frac{\sum_i f_{\mathrm{ILS}}(i) - \sum_i f_C(i)}{\sum_i f_C(i)}, \tag{20}$$

and the scale-normalized per-instance gap is

$$g_i(C) = 100\,\frac{f_C(i) - f_{\mathrm{ILS}}(i)}{f_C(i)}. \tag{21}$$

Throughout the paper, including the ablation tables, gaps follow the convention of Eq. (21): *negative values indicate ILS (or full-configuration) superiority*, zero indicates equal scores, and positive values indicate that the comparison scores higher. Applied to summed scores, Eq. (21) gives the aggregate gap $-\Delta(C)$. An aggregate advantage

of ILS is therefore positive when the text states it as $\Delta(C)$ and negative in the gap columns of the tables, whose headers repeat the sign convention.

Paired comparisons use the two-sided exact Wilcoxon signed-rank test on the nonzero paired differences (Wilcoxon 1945), with p-values adjusted by Holm's sequential procedure within each test family (Holm 1979). All signed-rank p-values in the paper are computed from the exact null distribution of the test statistic, never from its normal approximation, regardless of the number of nonzero pairs. The test families are the five external comparisons, the four component ablations, the three operator-group ablations, and the single initialization test. Per-instance comparison counts, that is, the numbers of instances on which the score of ILS (or of the full configuration) is better than, worse than, or equal to that of the comparison method, are always reported next to the tests, because the medians of many comparisons are zero and significance then rests on the direction and ranks of the nonzero differences. The tables abbreviate these counts as B/W/E (better/worse/equal). The external comparisons pair ILS best-of-ten scores. The ablation and initialization tests of Sects. 5.1–5.3 pair per-instance run means, since both sides of those comparisons are ten-run stochastic configurations. MILP1 and MILP2 are excluded from stand-alone Wilcoxon tests because their instance coverage is incomplete, but their valid scores still strengthen the per-instance portfolio. Because score scales differ by orders of magnitude across instances, aggregate scores are always read jointly with family totals, per-instance comparison counts, the scale-normalized gaps of Eq. (21), and the instance-level data of Sect. 5.8. All tests use full-precision archived values, not rounded table entries.

5 Results

All results refer to the 68 evaluation instances and to the frozen configuration of Table 7, with ten seeds per instance. The reference point throughout is the full configuration: its best-of-ten total is 477,429,716 and the sum of its ten-run means is 477,309,618. We first decompose where that quality comes from, by removing one mechanism at a time (Sect. 5.1), one operator group at a time (Sect. 5.2), and the initial construction (Sect. 5.3), and then examine how the search behaves over time and across seeds (Sect. 5.4). These four subsections test H2 and address the explanatory questions of Sect. 4. The comparison against the external methods, which tests H1, follows in Sects. 5.5–5.7.

5.1 Component ablation

The component ablation isolates the contribution of each mechanism of the ILS loop. Starting from the tuned configuration, we disable one mechanism at a time (probabilistic acceptance, perturbation, or restart), and a fourth variant applies a single local-search pass to the constructed solution and skips the loop entirely. Every variant keeps the remaining tuned parameters and the full ten-seed protocol, so that a difference is attributable to the disabled mechanism rather than to a change of parameters. In the *no perturbation* variant, each outer round re-enters local search from the unchanged home solution. The loop does not degenerate, because the local search itself is stochastic (operators are sampled at random from the weighted catalog), so

Table 9 Component ablation on the 68 evaluation instances, ten seeds per variant. The gap compares best totals under Eq. (21) (negative: full ILS superior). B/W/E counts the instances on which the score of the full configuration is better than, worse than, or equal to that of the variant. These counts and the Holm-adjusted p-values pair per-instance run means of the full configuration against the variant. Bold marks the reference full configuration, not the column maximum.

Configuration	Best total	Mean total	Gap (%, < 0 favors full ILS)	B/W/E	Adjusted p
Full ILS	**477,429,716**	**477,309,618**	–	–	–
One local search only	476,181,564	475,600,821	−0.262	37/0/31	5.821×10^{-11}
No probabilistic accept.	477,415,465	477,319,444	−0.003	15/16/37	0.931
No perturbation	477,465,158	477,330,571	+0.007	12/20/36	0.358
No restart	477,359,379	477,303,271	−0.015	18/13/37	0.635

successive rounds still follow different search trajectories, and acceptance and restart remain active. The variant therefore isolates the contribution of the perturbation *step*, not of diversification as a whole.

In Table 9, the one-pass variant scores 1,248,152 best-of-ten points below full ILS, which scores better on all 37 instances with a nonzero mean difference and is equal on the remaining 31. This is the clearest component effect (Holm-adjusted $p = 5.821 \times 10^{-11}$) and supports H2. It identifies the iterated loop, rather than any single diversification device, as the main source of quality beyond construction.

The other three variants show no statistically detectable difference from the full configuration after Holm correction. Removing perturbation even yields slightly higher best-of-ten and mean totals. The difference of 35,442 best-of-ten points (0.007%) is distributed inconsistently across instances and is not significant. Two cautions govern the reading of these null results. First, they are statements about detectability, not equivalence: no equivalence test was performed, and with 36–37 equal pairs among the 68 the design has little power against effects of this size, which are of the same order as the seed-to-seed variability of Sect. 5.4 (mean coefficient of variation 0.0078%). Second, the apparent redundancy of perturbation has a structural explanation specific to this configuration: the tuned perturbation is replacement-biased (probability 0.6794 of exchanging weak active libraries for promising inactive candidates), so its diversification role overlaps with the insertion/replacement operator group, which has the largest tuned weight and is the only one whose removal produces a significant effect (Sect. 5.2). On this benchmark the two channels overlap, and disabling the perturbation step shifts the same kind of exploration onto the stochastic local search.

This redundancy is conditional on the tuned configuration and the instance distribution, not a general property of the method. Under a different configuration (a larger base strength or nonzero stagnation growth, both inside the iRace domains of Table 7) or on instance families whose local optima form deeper basins than those observed here, the perturbation step remains the canonical ILS escape mechanism (Lourenço et al. 2003), and nothing in this one-factor-at-a-time experiment rules out such a role. For this reason, and because exchanging the frozen configuration for the nominally

Table 10 Operator-group ablation on the 68 evaluation instances. Columns as in Table 9.

Configuration	Best total	Mean total	Gap (%, < 0 favors full ILS)	B/W/E	Adjusted p
Full ILS	**477,429,716**	**477,309,618**	–	–	–
No insertion/replacement	477,293,236	477,108,533	−0.029	28/2/38	2.794×10^{-8}
No reordering moves	477,334,136	477,267,413	−0.020	15/16/37	1.000
No structural moves	477,413,931	477,221,144	−0.003	15/16/37	1.000

better no-perturbation variant after inspecting evaluation results would constitute precisely the post-hoc evaluation-set adaptation that the protocol of Sect. 4 is designed to exclude, perturbation is retained in the proposed method. We therefore do not claim an independent average benefit for each diversification device under the present budget. The defensible interpretation is that local search provides most of the quality, while probabilistic acceptance, perturbation, and restart act as alternative exploration, stabilization, and recovery paths whose benefit depends on the instance, the seed, and the configuration. Notably, the one-pass variant finishes in about 16 s on average, so it is an attractive budget-constrained alternative when its 0.262% aggregate reduction is acceptable.

5.2 Neighborhood operator analysis

The operator-group ablation removes one of the three operator groups of Table 2 at a time, renormalizes the sampling probabilities of the remaining operators, and reruns the full ten-seed protocol. It gives a sharper answer than the component ablation (Table 10). Removing the insertion/replacement group reduces the best total by 136,480 points and the mean total by 201,085, about 0.042% of the full-configuration mean total. The effect is small, but the full configuration scores better on 28 of the 30 instances with unequal run means, and the Holm-adjusted signed-rank test is significant ($p = 2.794 \times 10^{-8}$). Removing the reordering or structural groups produces small, inconsistent differences that do not survive correction.

This pattern is structurally plausible. Reordering can only optimize activation dates within the set of libraries already selected. Insertion and replacement cross the active/inactive boundary and can introduce books that no rearrangement of the current set could recover. The finding is consistent with the automatic configuration, which assigned this group the largest tuned weight (2.3634 versus 0.6317 for reordering).

Fig. 5 complements the ablation with the acceptance counts of the individual operators recorded during the instrumented convergence runs of Sect. 5.4, using the operator names of Table 2. The counts measure how often each operator produced an accepted move, not how much each contributed to the final score, and are therefore read together with Table 10 rather than in place of it.

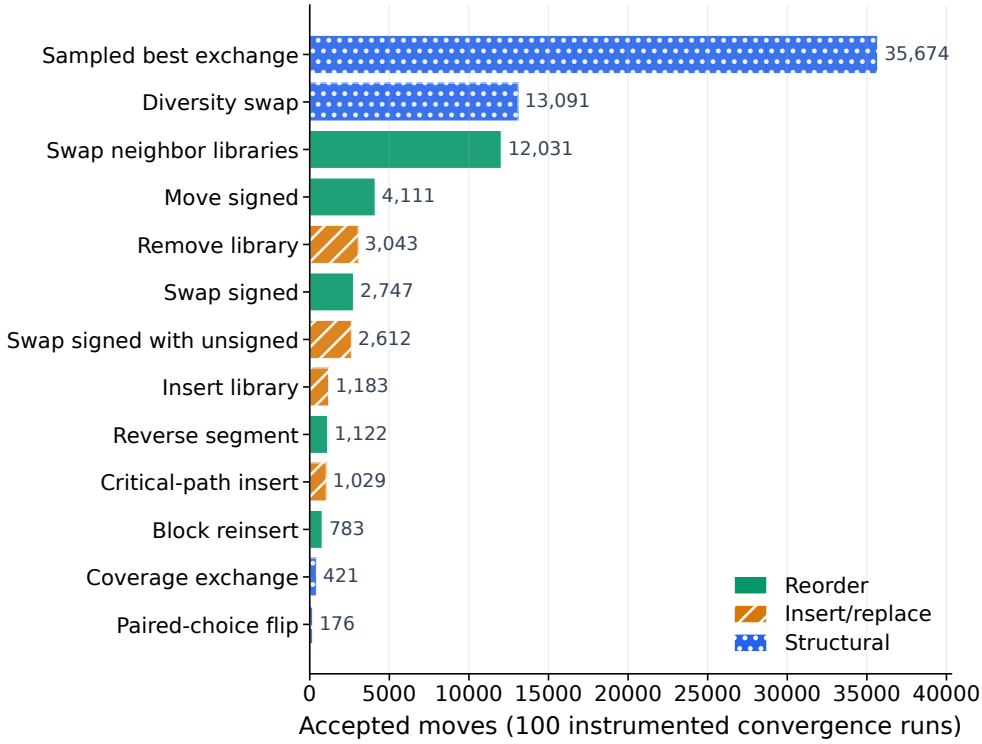


Fig. 5 Accepted moves per operator during the 100 instrumented convergence runs. Color and hatch pattern encode the operator group. Counts measure activity, not marginal contribution.

5.3 Effect of the initialization

The initialization experiment tests whether the quality of the initial solution is essential to the final result. It replaces the initial construction by the simple sign-up-time ordering (the *Sorted* rule of Table 1) and reruns the full ten-seed protocol. Fresh restarts still use the constructor portfolio of Sect. 3.3. With the full constructor, the ILS phase improves the constructed starting solutions by 0.557% on run means and by 0.582% measured with best-of-ten, aggregated over the 68 evaluation instances: the summed mean constructed total of 474,667,265 (Table 11) rises to final totals of 477,309,618 and 477,429,716, respectively.

The results (Table 11) show a strong corrective ability. The weak starts total 421,718,944 before search and 476,696,335 on average afterwards, a 13.037% improvement during search. Their best-of-ten total is only 0.042% below that of full ILS, and the paired test does not reach the 5% level ($p = 0.0678$). The larger 0.129% reduction in the mean total nevertheless shows the practical role of the constructor: the best of ten weak-start runs recovers most of the difference while their mean does not, so a good start mainly raises the typical run, and the complete algorithm, including its fresh restarts, can usually repair a weak start when given its full budget.

Table 11 Weak-initialization experiment on the 68 evaluation instances. Initial total is the sum of the mean constructed scores before ILS. Gaps compare the weak-start variant against full ILS under Eq. (21) (negative: full ILS superior). B/W/E counts the instances on which the score of full ILS is better than, worse than, or equal to that of the weak-start variant, and these counts and the p-value pair per-instance run means.

				Gap (%, <0 favors full ILS)			
Configuration	Initial total	Best total	Mean total	Best	Mean	B/W/E	p
Full ILS	474,667,265	**477,429,716**	**477,309,618**	–	–	–	–
Weak start	421,718,944	477,228,453	476,696,335	−0.042	−0.129	20/12/36	0.0678

Table 12 Evaluation instances with the largest relative improvement of full ILS over its constructed initial solution. Initial is the per-instance mean constructed score over the ten seeds. ILS best is the best of the ten runs.

Family	Instance	Initial	ILS best	Improvement (%)
Real-world-based	`B90k_L850_D21.txt`	828	869	4.952
Real-world-based	`B35k_L60_D120.txt`	12,544,648	12,864,753	2.552
Real-world-based	`B240k_L90_D180.txt`	42,359,386	43,085,291	1.714
Seed-based	`ce_090x_008t.txt`	20,368,337	20,713,720	1.696
Real-world-based	`B1000k_L115_D230.txt`	56,328,452	57,027,193	1.240
Seed-based	`e_090x_008t.txt`	8,809,568	8,912,388	1.167
Google Hash Code	`d_tough_choices.txt`	5,039,931	5,094,050	1.074
Seed-based	`e_060x_012t.txt`	5,613,967	5,649,393	0.631
Real-world-based	`B90_L6_D11.txt`	2,551	2,567	0.627
Real-world-based	`B2000k_L140_D280.txt`	60,713,936	61,011,720	0.490

To locate where the ILS phase adds most beyond construction, Table 12 lists the evaluation instances on which the full configuration improves most over its own constructed start. The largest relative gains concentrate in the real-world-based family and in seed-based cases with tight horizons, that is, exactly where the ordering decision leaves real room for repair. On `B90k_L850_D21.txt`, an extreme case with 850 libraries and only 21 days, full ILS raises the score by 4.952%.

5.4 Convergence and seed stability

Convergence was instrumented on ten purposively selected evaluation instances with ten seeds each (100 runs), recording time, phase, current and best score, and search events. The ten instances were chosen as diagnostic cases that showed visible ILS-phase improvement in preliminary runs. They are not a random or representative sample of the evaluation set, so the per-family figures in this subsection describe these selected cases and are not generalized to their full families. At the start of the ILS phase, construction had already supplied, on average over the 100 instrumented runs, 98.893% of the eventual final score (95.282% in the worst instrumented case). The mean time to reach 95% of the eventual improvement over the constructed solution (t_{95}, defined in the caption of Table 13) was 134.0 s. The family means in Table 13 show two regimes, visible also in Fig. 6: the selected real-world-based cases improve

Table 13 Convergence diagnostics from the 100 instrumented runs on the ten purposively selected diagnostic instances. Improvement is relative to the constructed solution. t_{95} is the ILS-phase time at which a run first reaches 95% of its eventual improvement over the constructed solution, and a run without improvement counts as $t_{95} = 0$. Records count every strict improvement of the incumbent during the ILS phase, including those found by a restart. Records and restarts are per-run means. Family rows summarize only the selected instances, not the full families.

Family	Instances	Improvement (%)	t_{95} (s)	Records	Restarts
Google Hash Code	3	0.509	111.8	50.0	13.2
Real-world-based	3	2.640	8.0	1.3	81.8
Seed-based	2	0.922	233.0	236.4	0.8
Synthetic	2	0.059	257.1	11.8	1.4
All	10	1.141	134.0	65.0	28.9

Table 14 Per-instance convergence diagnostics (ten runs each). CV is the coefficient of variation of the final score across seeds.

Instance	ILS mean	Improvement (%)	t_{95} (s)	CV (%)	Restarts
`d_tough_choices.txt`	5,092,633	1.046	260.1	0.0159	3.8
`e_so_many_books.txt`	5,239,942	0.332	71.1	0.0016	1.1
`f_libraries_of_the_world.txt`	5,348,248	0.149	4.1	0.0000	34.8
`B2000k_L140_D280.txt`	60,966,934	0.417	14.3	0.0258	24.7
`B35k_L60_D120.txt`	12,864,753	2.552	9.5	0.0000	39.9
`B90k_L850_D21.txt`	869	4.952	0.3	0.0000	180.7
`ce_060x_012t.txt`	13,013,931	0.311	85.8	0.0041	1.5
`ce_090x_008t.txt`	20,680,564	1.533	380.2	0.0529	0.0
`synthetic_20.txt`	29,691,848	0.086	258.0	0.0134	1.9
`synthetic_60.txt`	18,670,432	0.032	256.2	0.0202	0.9

almost immediately ($t_{95} = 8.0$ s) through many restarts, whereas the selected seed-based and synthetic cases continue to accumulate record improvements until near the 600 s limit. Table 14 breaks the diagnostics down by instance.

To assess seed stability over the complete evaluation set, we compute the coefficient of variation of the final score over the ten seeds of each instance (Fig. 7). The mean per-instance coefficient of variation is 0.0078% (median 0.0000%), so the variability is low. The family means are 0.0039% for the Google Hash Code, 0.0029% for the real-world-based, 0.0159% for the seed-based, and 0.0021% for the synthetic family. The maximum, 0.1117%, occurs on `e_060x_012t.txt`. Low variation makes the best-of-ten statistic insensitive to any single extreme run, consistent with the near-identical robustness rows of Table 17, although it does not remove the protocol asymmetry with the single-result comparison methods.

5.5 Comparison with the external portfolio

The main comparison measures the quality of the tuned ILS against the external portfolio of Sect. 4.4, first in aggregate and by family, then per instance. Table 15

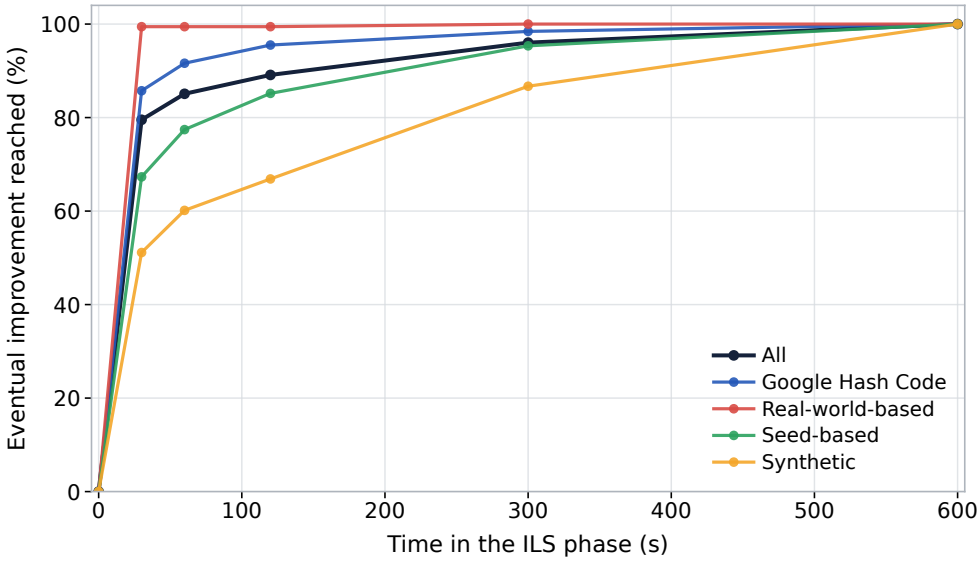


Fig. 6 Mean fraction of the eventual post-construction improvement reached at fixed ILS checkpoints, by family of the ten purposively selected diagnostic instances. Runs with no improvement have an undefined fraction and are excluded from the corresponding mean.

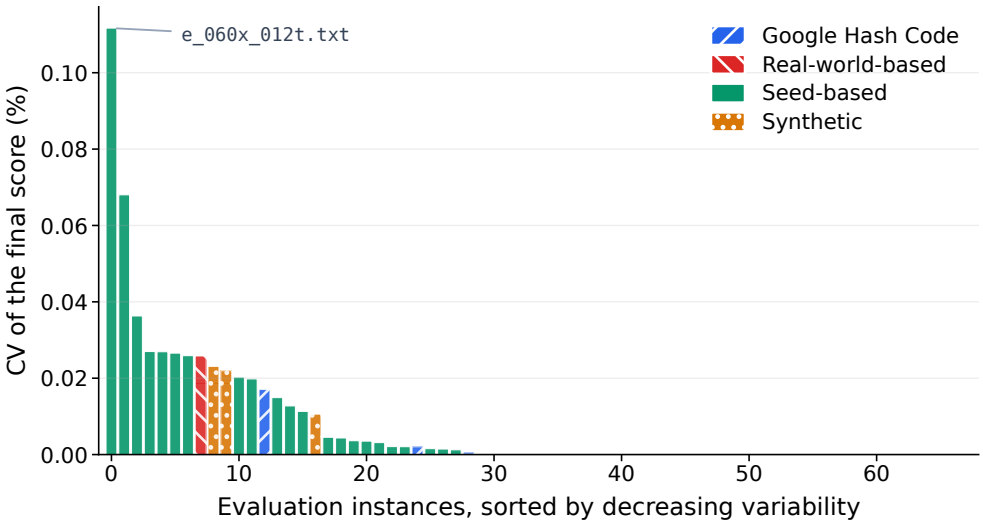


Fig. 7 Coefficient of variation of the final score over the ten seeds of each of the 68 evaluation instances, sorted by decreasing variability. Color and hatch pattern encode the family, and 39 instances reach identical scores on all ten seeds.

Table 15 Total score by evaluation family and method. ILS best sums the best of ten runs per instance. ILS mean sums the ten-run means. The portfolio row is the per-instance best of the six external methods. MILP1 and MILP2 are omitted as stand-alone rows because they do not cover all instances, but their valid scores enter the portfolio.

Method	Google Hash Code (5)	Real-world-based (9)	Seed-based (27)	Synthetic (27)	All (68)
ILS (best of ten)	27,195,847	**176,432,691**	**204,213,202**	**69,587,976**	**477,429,716**
ILS (run mean)	27,194,432	176,387,905	204,155,472	69,571,808	477,309,618
External portfolio	**27,211,759**	167,590,854	202,301,142	69,310,348	466,414,103
Errecto	26,959,871	167,590,854	194,339,348	69,288,893	458,178,966
Greedy	22,488,759	136,191,957	182,920,078	67,068,027	408,668,821
SAT	26,278,508	136,780,713	197,566,582	67,605,727	428,231,530
SA	20,535,469	136,385,015	180,425,480	67,628,046	404,974,010

Family run means are rounded independently. Their displayed values sum to 477,309,617, whereas the all-instance mean total, rounded only after aggregation, is 477,309,618.

reports total scores by family and method, and Fig. 8 visualizes the totals. The best-of-ten ILS total over the 68 evaluation instances is 477,429,716 and the sum of the ten-run means is 477,309,618. The external portfolio reaches 466,414,103, so by Eq. (20) the best-of-ten advantage is 2.362% (11,015,613 points). Measured with run means instead, the advantage is 2.336%. Most of this advantage is already present after the construction phase of the method. The summed mean constructed total of 474,667,265 (Table 11) is 1.769% above the external portfolio, and the ILS phase raises it by a further 0.557% on run means and 0.582% with best-of-ten (Sect. 5.3). Among the individual methods, Errecto is the best external solver in aggregate (458,178,966), followed by SAT (428,231,530), Greedy (408,668,821), and SA (404,974,010).

The advantage is not uniform across families. ILS finishes 0.058% below the portfolio on the five Google Hash Code instances, whose public solutions are heavily specialized. It is 5.276% ahead on the real-world-based family, 0.945% ahead on the seed-based family, and 0.401% ahead on the synthetic family. The largest single-instance gain follows the same pattern: on the two-million-book case `B2000k_L140_D280.txt`, the ILS best exceeds the portfolio by 13.706%. This one instance contributes 7,354,476 of the 11,015,613 points of net aggregate gain, about two thirds. The family differences and this concentration are why the paired evidence below is reported alongside the grand total.

5.6 Paired statistical evidence

Aggregate totals can be dominated by a few large instances, so the paired analysis asks on how many instances, and how consistently, ILS scores better than each external method. Table 16 reports the paired comparisons. Against the per-instance portfolio, the score of ILS is better on 24 instances, worse on 13, and equal on 31. The Holm-adjusted p-value is 2.176×10^{-4}, so the paired comparison supports H1. Against Errecto, ILS is never worse on the evaluation set and is better on 40 instances. The margins against Greedy, SAT, and SA are larger but less informative about the best available competitor. The median portfolio gap is zero because 31 instances are equal.

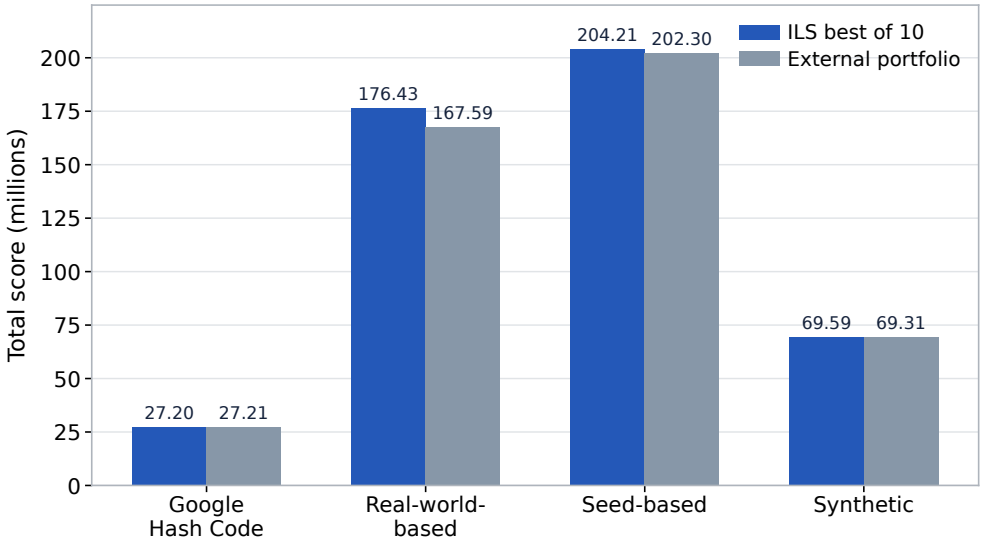


Fig. 8 Total score by evaluation instance family. Scores are shown in millions. The external bar is the per-instance portfolio, not a single algorithm.

Table 16 Paired comparisons of ILS best-of-ten against the external methods on the 68 evaluation instances. Gap columns follow Eq. (21) (negative favors ILS). Total is the aggregate gap of summed scores. Mean and Median summarize per-instance gaps. The p-values are Holm-adjusted within this family of five tests.

				Gap (%, < 0 favors ILS)			
Comparison	Better	Worse	Equal	Total	Mean	Median	Adjusted p
External portfolio	24	13	31	−2.362	−0.576	0.000	2.176×10^{-4}
Errecto	40	0	28	−4.202	−3.513	−0.320	3.638×10^{-12}
Greedy	47	1	20	−16.826	−13.835	−6.647	5.684×10^{-14}
SAT	48	1	19	−11.489	−5.156	−2.692	5.862×10^{-13}
SA	50	1	17	−17.891	−55.925	−3.699	3.109×10^{-14}

Significance comes from the direction and ranks of the 37 nonzero differences, not from ubiquitous improvement. On the instances they cover, the MILP variants behave asymmetrically: MILP2 never exceeds the ILS best on its 56 valid instances (ILS better on 40, equal on 16), whereas MILP1 exceeds it on 10 of its 41 valid instances (mostly seed-based variants of the official `c` and `d` structures), which is precisely where the portfolio draws several of the instances on which it scores better.

Because the aggregate totals weight large instances heavily, we also test whether the portfolio result survives scale normalization and a change of the ILS reporting statistic (Table 17). With best-of-ten, the mean per-instance gap is −0.576% and the signed-rank test on the nonzero normalized gaps gives $p = 1.705 \times 10^{-3}$. Replacing best-of-ten by the run mean leaves every entry essentially unchanged (mean gap −0.560%, $p =$

Table 17 Robustness of the portfolio comparison to the ILS reporting statistic and to score normalization. Gap total is the aggregate gap of the summed scores, as in Table 16, computed from the totals of Table 15. Gap total and the mean and median g_i follow Eq. (21) (negative favors ILS). p_{raw} tests raw paired scores and p_{norm} the nonzero normalized gaps, both two-sided and unadjusted.

	Gap (< 0 favors ILS)					
ILS statistic	Total	Mean g_i	Median g_i	B/W/E	p_{raw}	p_{norm}
Best of ten	-2.362%	-0.576%	0.000%	24/13/31	2.176×10^{-4}	1.705×10^{-3}
Run mean	-2.336%	-0.560%	0.000%	24/13/31	2.507×10^{-4}	1.913×10^{-3}

1.913×10^{-3}), and the better/worse/equal pattern is identical. The conclusion in favor of ILS therefore depends neither on the largest score scales nor on the best-of-ten protocol.

5.7 Optimality certificates from the upper bound

Because the simple bound of Eq. (15) is valid, a solution that attains it is optimal, so the best solutions are checked against the bound on every evaluation instance. The ILS best attains the bound on 21 of the 68 evaluation instances, all from the synthetic family. The external portfolio reaches the same value on 21 of them and on one further instance. That further case is instructive: on `d_tough_choices.txt`, SAT attains the bound value 5,109,000, proving that every available book can be scanned, while the ILS best reaches 5,094,050 (0.293% below). These checks show both the value and the limit of the bound: equality certifies a solution exactly, but the often large positive gap on the remaining instances cannot quantify the distance to the optimum. Complete instance-level values, including the bound, appear in Sect. 5.8.

5.8 Instance-level results

The instance-level view shows whether the aggregate, family, and paired conclusions hold on single instances or rest on a few large cases. Table 18 reports, for each of the 68 evaluation instances, the values underlying those analyses: the simple upper bound, the ILS best-of-ten and run mean, the external portfolio value, and the normalized gap of Eq. (21). At this level, the family pattern of Sects. 5.5–5.7 remains visible: the large real-world-based cases carry the largest absolute gains, the seed-based family mixes positive and negative gaps of a few percent at most, and the synthetic family is dominated by equal scores at the upper bound. Per-instance run means are rounded independently to integers, so the displayed mean column can sum to one point more than the total row, which is rounded only after aggregation. The separate scores of all six external constituents, together with the derived per-instance portfolio column and an auxiliary Greedy+MILP baseline that never enters the portfolio, are provided as Online Resource 2.

Table 18: Instance-level evaluation results, grouped by family (the `.txt` suffix of instance names is omitted). UB is the simple upper bound of Eq. (15); ILS best and ILS mean summarize ten runs; Portfolio is the per-instance best of the six external methods; the gap follows Eq. (21), negative values indicating ILS superiority. The gap of the total row is the aggregate gap of the summed scores, as in Table 16.

Instance	UB	ILS best	ILS mean	Portfolio	Gap (%, < 0 favors ILS)
Google Hash Code					
b_read_on	10,000,000	5,822,900	5,822,900	5,822,900	0.000
c_incunabula	23,372,244	5,690,468	5,690,439	5,690,888	0.007
d_tough_choices	5,109,000	5,094,050	5,092,841	5,109,000	0.293
e_so_many_books	12,457,261	5,240,181	5,240,005	5,240,723	0.010
f_libraries_of_the_world	39,885,586	5,348,248	5,348,248	5,348,248	0.000
Real-world-based					
B1000k_L115_D230	360,585,201	57,027,193	57,027,193	55,879,583	−2.054
B100_L9_D18	6,246	4,916	4,916	4,916	0.000
B1k_L18_D17	283,465	127,579	127,579	127,579	0.000
B2000k_L140_D280	578,379,827	61,011,720	60,966,934	53,657,244	−13.706
B240k_L90_D180	138,925,961	43,085,291	43,085,291	42,745,540	−0.795
B35k_L60_D120	20,578,732	12,864,753	12,864,753	12,864,753	0.000
B6k_L35_D70	2,936,308	2,307,803	2,307,803	2,307,803	0.000
B90_L6_D11	3,704	2,567	2,567	2,567	0.000
B90k_L850_D21	418,275	869	869	869	0.000
Seed-based					
b_075x_012t	3,374,450	185,334	185,334	185,341	0.004
bc_060x_008t	3,369,254	923,235	923,220	923,357	0.013
bc_075x_020t	5,168,217	1,299,768	1,299,742	1,299,813	0.003
bd_060x_012t	3,597,755	1,623,707	1,622,614	1,631,093	0.453
bd_090x_008t	6,073,672	2,375,036	2,374,322	2,293,686	−3.547
be_060x_020t	5,500,376	4,475,296	4,473,336	4,469,247	−0.135
be_090x_012t	9,261,438	7,353,771	7,351,926	7,122,631	−3.245
bf_075x_008t	7,011,170	4,494,063	4,493,613	4,449,237	−1.007
bf_090x_020t	7,969,327	6,872,910	6,872,910	6,844,683	−0.412
c_075x_012t	16,428,705	4,332,083	4,331,910	4,332,827	0.017
cd_060x_008t	11,362,607	4,428,885	4,428,199	4,460,167	0.701
cd_075x_020t	16,643,615	10,855,183	10,851,650	10,928,387	0.670
ce_060x_012t	18,615,160	13,014,383	13,013,811	12,787,403	−1.775
ce_090x_008t	28,729,695	20,713,720	20,692,587	20,127,366	−2.913
cf_060x_020t	16,143,221	13,442,088	13,441,428	13,382,904	−0.442
cf_090x_012t	28,288,174	21,425,988	21,425,636	21,338,035	−0.412
d_075x_008t	2,899,800	1,132,304	1,131,582	1,140,023	0.677
d_090x_020t	3,773,722	2,543,509	2,542,796	2,564,848	0.832
de_075x_012t	3,783,606	2,976,941	2,975,335	2,874,196	−3.575
df_060x_008t	2,933,572	2,007,570	2,007,479	2,002,526	−0.252
df_075x_020t	3,903,784	3,404,720	3,404,637	3,390,357	−0.424
e_060x_012t	7,723,605	5,649,393	5,631,492	5,535,451	−2.058
e_090x_008t	12,120,060	8,912,388	8,910,345	8,704,950	−2.383
ef_060x_020t	7,007,807	5,938,316	5,938,316	5,918,919	−0.328
ef_090x_012t	9,193,700	8,009,885	8,009,885	7,978,463	−0.394
f_075x_008t	24,117,937	18,243,051	18,243,051	18,186,218	−0.313
f_090x_020t	29,232,950	27,579,675	27,578,316	27,429,014	−0.549

Continued on next page

Table 18 (continued)

Instance	UB	ILS best	ILS mean	Portfolio	Gap (%, < 0 favors ILS)
Synthetic					
synthetic_10	51,834	17,463	17,463	17,463	0.000
synthetic_100	156,794	156,794	156,794	156,794	0.000
synthetic_105	8,558,652	8,490,974	8,490,974	8,490,974	0.000
synthetic_110	14,805,607	3,351,703	3,350,131	3,318,210	−1.009
synthetic_115	345,277	345,277	345,277	345,277	0.000
synthetic_120	187,715	187,715	187,715	187,715	0.000
synthetic_125	330,735	330,735	330,735	330,735	0.000
synthetic_130	226,911	226,911	226,911	226,911	0.000
synthetic_135	217,206	217,206	217,206	217,206	0.000
synthetic_15	49,747	49,747	49,747	49,747	0.000
synthetic_20	31,034,336	29,698,214	29,691,957	29,601,148	−0.328
synthetic_25	244,295	244,295	244,295	244,295	0.000
synthetic_30	2,282,117	2,282,117	2,282,117	2,282,117	0.000
synthetic_35	163,375	163,375	163,375	163,375	0.000
synthetic_40	236,992	236,992	236,992	236,992	0.000
synthetic_45	11,254,939	1,063,158	1,063,158	1,063,451	0.028
synthetic_5	28,639	28,639	28,639	28,639	0.000
synthetic_50	142,479	142,479	142,479	142,479	0.000
synthetic_55	65,074	65,074	65,074	65,074	0.000
synthetic_60	33,626,312	18,682,072	18,673,734	18,534,710	−0.795
synthetic_65	132,057	132,057	132,057	132,057	0.000
synthetic_70	249,881	249,881	249,881	249,881	0.000
synthetic_75	140,755	140,755	140,755	140,755	0.000
synthetic_80	137,454	137,454	137,454	137,454	0.000
synthetic_85	2,597,661	2,597,661	2,597,661	2,597,661	0.000
synthetic_90	177,823	177,823	177,823	177,823	0.000
synthetic_95	171,405	171,405	171,405	171,405	0.000
Total (68 instances)	1,594,785,261	**477,429,716**	477,309,618	466,414,103	−2.362

6 Discussion

The results obtained by the proposed solver for the Book Scanning Problem support both hypotheses of Sect. 4 and lead to three conclusions. First, the order representation is effective for this problem because a small local change simultaneously modifies activation dates and effective capacities, while the decoding portfolio repairs the book assignment at low cost. Second, local search is the dominant quality mechanism: the full ILS loop yields significantly higher run means than a one-pass local search (H2), but no single diversification component produces a robust isolated effect under the present budget. This is a statement about detectability within the tuned configuration, not a claim that these mechanisms are unnecessary in general (Sect. 5.1). Third, changing the composition of the active library set matters more consistently than rearranging the same libraries: the only operator group with a significant ablation effect is insertion/replacement, in agreement with its largest tuned weight.

The family pattern clarifies when the method helps. The Google Hash Code instances are few and have been examined intensively by contest participants, and

specialized methods retain a small advantage there. The real-world-based and seed-based families expose more opportunities for marginal-value correction, and there the insertion and replacement moves recover high-scoring or rare books that a constructor misses. Many synthetic instances are easy for several methods and end with equal scores, including the 21 cases certified optimal by the upper bound.

The aggregate and paired views answer different questions. The total score matches the original contest objective, in which points are added across instances, but it weights large cases heavily. The scale-normalized gap and its signed-rank test give each instance comparable influence. Both views favor ILS (H1), although the zero median gap and the 13 instances on which ILS is worse rule out any claim of uniform improvement.

Most limitations of this study concern how far its conclusions extend, and the design already contains several of them. The simple upper bound of Eq. (15) certifies 21 solutions as optimal. On the other instances it is loose, so positive gaps bound the distance to the optimum without measuring it, and stronger relaxations would sharpen that picture. The iRace race used only training instances. Its post-selection stage evaluated five elites once on the evaluation instances (Sect. 4.3), but the selected configuration was already the elite of the training race and was not changed, although the evaluation set was not fully blinded. Because ILS is stochastic, it receives ten seeds, whereas each comparison record is a single fixed result. Best-of-ten therefore measures multi-start potential, and the reported run means and the robustness analysis of Table 17 address this asymmetry. A fully symmetric design, in which every stochastic baseline receives the same seeds, would complete it. The external portfolio selects its best constituent per instance and preserves no comparable runtime records, so it is a quality reference and no efficiency comparison is claimed. MILP1 and MILP2 cover 41 and 56 of the 68 instances, so they are excluded from stand-alone tests while their valid scores strengthen the portfolio. The component ablations disable one mechanism at a time and quantify contributions within the final configuration (Sect. 5.1). They cannot detect interaction effects among perturbation, probabilistic acceptance, and restart, which may substitute for one another, so their null results hold for the tuned configuration, these instance families, and the 600 s budget and do not license removing a mechanism in general.

The remaining limitations concern the scope of the comparison and of the problem model. The study compares the reproducible implementations in the repository, not the complete public contest leaderboard, and accordingly claims no new best-known values. Where the 120 s + 600 s budget is more than an easy instance needs, the one-pass local-search variant finishes in about 16 s on average at a 0.262% aggregate best-score reduction. Finally, the benchmark is a contest abstraction of digitization, and an operational deployment would add failures, transport, scanner sharing, and dynamic arrivals.

Future work follows directly from these limitations: stronger upper bounds; more powerful book-allocation subroutines between the current greedy portfolio and a full minimum-cost flow (Rodrigues 2023b); instance-dependent operator and budget policies learned from the archived run data; a symmetric comparison design with common seeds and budgets for all stochastic methods; parallel multi-start ILS, which

the early-plateau instances suggest could use the same total budget more effectively than one long trajectory; and a comparison, on this instrumented benchmark, with approaches that, like FunSearch (Romera-Paredes et al. 2024), evolve priority functions automatically.

7 Conclusion

We presented an iterated local search for the Google Hash Code 2020 Book Scanning Problem that integrates multi-strategy construction, a portfolio of fast decoders, screening-accelerated and fully verified local search over thirteen neighborhoods, controlled perturbation, temperature-based acceptance, and stagnation restarts. Its main parameters were selected by an iRace training race on 256 instances. On the 68 evaluation instances from four families, the tuned configuration reached a best-of-ten total of 477,429,716 points, 2.362% above a per-instance portfolio of six external methods, with a better score on 24 instances, an equal score on 31, and a worse score on 13. The advantage survives Holm-corrected signed-rank testing, scale normalization, and replacement of best-of-ten by run means, and 21 solutions are certified optimal by a simple upper bound.

The analyses are as important as the headline score. They identify local search as the core mechanism, isolate library insertion and replacement as the one operator group with a reproducible contribution, show that the method recovers almost fully from a deliberately weak initialization, and document low seed-to-seed variability over the full evaluation set together with heterogeneous convergence behavior on selected diagnostic instances. This combination of competitive quality, explicit limitations, and auditable experimental artifacts (code, instances, and all run records are public) is the main contribution of the work.

Acknowledgements. The authors thank the maintainers of the public implementations and instance generators for the Book Scanning Problem used to construct the comparative benchmark.

Author contributions. D.A. developed the main ILS implementation, designed and executed the experiments, curated the results, and prepared the initial manuscript. K.S. supervised the research and contributed to the methodology and critical revision. A.A. contributed optimization-methodology expertise and critical revision. U.L. contributed benchmark datasets and the adapted Errecto comparison workflow. Y.B. developed BORT, a Google OR-Tools implementation for the Book Scanning Problem, and contributed to the mathematical-programming analysis. All authors reviewed and approved the manuscript.

Statements and Declarations

Funding. No specific funding was received for this study.

Data availability. The instance catalogs, normalized result tables, convergence summaries, and the data behind every figure are available in the project repository

(Alija 2026c). The Google Hash Code, real-world-based, seed-based, and synthetic instance sources are identified individually in Sect. 4.1.

Code availability. The Python implementation, experiment runner, iRace scenario, selected parameter set, solution validator, analysis scripts, and the exact commands for reproducing the experiments are available at https://github.com/dritonalija/ils_book_scanning.

Supplementary information. The electronic supplementary material contains all 680 evaluation ILS run records (Online Resource 1), the normalized external-method records, including the derived portfolio and an auxiliary Greedy+MILP baseline (Online Resource 2), the component-ablation records (Online Resource 3), the operator-group-ablation records (Online Resource 4, whose `variant` column identifies the ablated operator group of each run), the structural catalog of all 324 instances (Online Resource 5), the checkpoint data underlying Fig. 6 (Online Resource 6), and the per-run convergence diagnostics behind Tables 13 and 14 (Online Resource 7).

Competing interests. The authors declare that they have no competing interests.

Ethics approval. Not applicable.

Consent to participate. Not applicable.

Consent for publication. Not applicable.

ORCID iDs. Driton Alija: 0009-0009-1084-5422; Kadri Sylejmani: 0000-0002-4428-1161; Arben Ahmeti: 0000-0002-5981-7336; Uran Lajçi: 0009-0002-6214-5865; Yll Berisha: 0009-0003-9417-0085.

Use of generative AI. ChatGPT and Claude assisted with source-code review, manuscript organization, and English-language editing. They did not generate experimental data, select reported runs, or replace the scripted statistical analyses. The authors verified all tool-assisted material against the source code and archived results and take full responsibility for the manuscript.

References

Achterberg, T.: SCIP: Solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009) https://doi.org/10.1007/s12532-008-0001-1

Alija, D., Kryeziu, M., Hoda, E., Islami, L., Mustafa, L., Berisha, Y., Sylejmani, K., Ahmeti, A., Arbneshi, L.: Optimization of the book scanning problem via iterated local search and genetic algorithms. In: Proceedings of the 6th International Conference on Recent Trends and Applications in Computer Science and Information Technology (RTA-CSIT 2025). CEUR Workshop Proceedings, vol. 4044, pp. 127–135. CEUR-WS.org, Tirana, Albania (2025). https://ceur-ws.org/Vol-4044/paper14.pdf

Alija, D.: Adapted Google Hash Code 2020 Qualification Solution. https://github.com/dritonalija/Google-Hash-Code-Qualification-2020. Accessed 29 July 2026 (2026)

Alija, D.: Book Scanning Seed-Based Instance Dataset. https://github.com/dritonalija/book_scanning_dataset. Accessed 29 July 2026 (2026)

Alija, D.: Iterated Local Search for the Book Scanning Problem: Source Code and Experimental Artifacts. https://github.com/dritonalija/ils_book_scanning. Source code, instances, experiment scripts, and results. Accessed 29 July 2026 (2026)

Avenel, P.: Google Hashcode Archive (Problems & Scoreboards). https://github.com/pierreavn/google-hashcode-archive/blob/main/archive/2020/qualification/hashcode_2020_qualification_round_scoreboard.json. Archived final standings of the 10,725 teams of the Hash Code 2020 online qualification round. Accessed 12 September 2026 (2022)

Balas, E.: The prize collecting traveling salesman problem. Networks **19**(6), 621–636 (1989) https://doi.org/10.1002/net.3230190602

Berisha, Y.: BORT (Book Scanning OR-Tools): Book Scanning Problem Using OR-Tools. https://github.com/yllberisha/Bort. OR-Tools implementation of the Book Scanning mathematical model. Accessed 29 July 2026 (2025)

Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM **45**(4), 634–652 (1998) https://doi.org/10.1145/285055.285059

Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Global Optim. **6**(2), 109–133 (1995) https://doi.org/10.1007/BF01096763

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

Gunawan, A., Lau, H.C., Vansteenwegen, P.: Orienteering problem: A survey of recent variants, solution approaches and applications. Eur. J. Oper. Res. **255**(2), 315–332 (2016) https://doi.org/10.1016/j.ejor.2016.04.059

Gjoka, E., Morina, B.: Book Scanning Problem Input Generator. https://github.com/endritgjoka/hashcodeproblem-inputgenerator. Accessed 29 July 2026 (2026)

Google: Book Scanning: Problem Statement for the Online Qualification Round of Hash Code 2020. https://github.com/pierreavn/google-hashcode-archive/blob/main/archive/2020/qualification/hashcode_2020_qualification_round.pdf. Accessed 29 July 2026 (2020)

Holm, S.: A simple sequentially rejective multiple test procedure. Scand. J. Stat. **6**(2), 65–70 (1979)

Indzhev, E.: Book Scanning Solution: Google Hash Code 2020. https://github.com/indjev99/Optimizational-Problems-Solutions/blob/master/book_scanning.cpp. Accessed 29 July 2026 (2020)

Kirkpatrick, S., Gelatt, C.D. Jr., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983) https://doi.org/10.1126/science.220.4598.671

Khuller, S., Moss, A., Naor, J.S.: The budgeted maximum coverage problem. Inf. Process. Lett. **70**(1), 39–45 (1999) https://doi.org/10.1016/S0020-0190(99)00031-9

Kostka, B.: GHC 2020: Google Hash Code 2020 Repository. https://github.com/kostka-dev/ghc20. Original Errecto team solution. Accessed 29 July 2026 (2020)

Lajçi, U.: Book Scanning: Adapted Errecto Implementation. https://github.com/uran-lajci/Book.Scanning.Errecto. Adapted comparison implementation. Accessed 29 July 2026 (2026)

Lajçi, U.: Book Scanning Dataset. https://github.com/uran-lajci/Book.Scanning.Dataset. Accessed 29 July 2026 (2026)

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016) https://doi.org/10.1016/j.orp.2016.09.002

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

Luke, S.: Essentials of Metaheuristics, 2nd edn. Lulu, Raleigh, NC (2013). Online version 2.4, https://people.cs.gmu.edu/~sean/book/metaheuristics/. Accessed 29 July 2026

Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997) https://doi.org/10.1016/S0305-0548(97)00031-2

Numba Developers: Numba: A High Performance Python Compiler. https://numba.pydata.org/. Accessed 29 July 2026 (2026)

Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions—I. Math. Program. **14**(1), 265–294 (1978) https://doi.org/10.1007/BF01588971

Rodrigues, P.: Google Hash Code Models. https://github.com/pedromig/hashcode-models. Accessed 29 July 2026 (2023)

Rodrigues, P.M.D.: Principled modeling of the Google Hash Code problems for metaheuristics. Master's thesis, Universidade de Coimbra (2023). https://hdl.handle.net/10316/110471

Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. **40**(4), 455–472 (2006) https://doi.org/10.1287/trsc.1050.0135

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., Kohli, P., Fawzi, A.: Mathematical discoveries from program search with large language models. Nature **625**(7995), 468–475 (2024) https://doi.org/10.1038/s41586-023-06924-6

Sayed, N.: Google Hash Code 2020 Qualification Solution. https://github.com/nawidsayed/Google-Hash-Code-Qualification-2020. Accessed 29 July 2026 (2020)

Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing – SAT 2009. Lecture Notes in Computer Science, vol. 5584, pp. 244–257. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_24

Taub, L.: Google's Hash Code competition is back. https://blog.google/technology/developers/googles-hash-code-2020/. The Keyword, Google, 14 January 2020. Accessed 12 September 2026 (2020)

Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. Eur. J. Oper. Res. **209**(1), 1–10 (2011) https://doi.org/10.1016/j.ejor.2010.03.045

Veličković, P., Vitvitskyi, A., Markeeva, L., Ibarz, B., Buesing, L., Balog, M., Novikov, A.: Amplifying Human Performance in Combinatorial Competitive Programming (2024). https://doi.org/10.48550/arXiv.2411.19744

Wilcoxon, F.: Individual comparisons by ranking methods. Biom. Bull. **1**(6), 80–83 (1945) https://doi.org/10.2307/3001968

Yahiaoui, A.-E., Afifi, S., Allaoui, H.: Enhanced iterated local search for the technician routing and scheduling problem. Comput. Oper. Res. **160**, 106385 (2023) https://doi.org/10.1016/j.cor.2023.106385