https://jsras.rcc.edu.ly/
Vol.3 No.2 (2026), 235-252
Article history:
Received: 15Sep .2026
Accepted: 4.Oct . 2026
Published: 18 Nov .2026

Journal of Sustainable Research in Applied Sciences

JSRAS

Critical Review of Constraint Representation and Evaluation in Combinatorial Resource Allocation

Ahmed Jaha[1,*], a.jaha@it.misuratau.edu.ly; goha_99@yahoo.com, ORCID: 0009-0006-2422-6874; Anwar Alhenshiri[2], ORCID: 0009-0007-8825-7685
[1,2]Department of Computer Science, Faculty of Information Technology, Misratah University, Misratah, Libya,
*Corresponding author

Index Terms

Combinatorial resource allocation; Constraint representation; Feasibility enforcement; Metaheuristics; Hybrid optimization; Learning-based optimization.

Abstract

In combinatorial resource allocation, a representation change may alter model meaning, inference, or search. This critical methodological review examines these distinctions across exact optimization, constraint programming, constructive heuristics, metaheuristics, hyper-heuristics, hybrid methods, and learning-assisted methods. The synthesis draws on 84 sources and uses six analytical dimensions. Descriptive counts summarize publication years and source roles; 12 selected mechanism records are classified by principal representation, validity operation, guarantee scope, and evidence type. The contribution is a component-level evidence map that separates changed model semantics from stronger inference and redirected search. Equivalent formulations can support different deductions, while an immediately admissible choice may prevent completion. Matched queries preserve decisions; changed access order may alter them. A four-step procedure connects these distinctions to global propagation, learned branching, and two published timetabling studies. Checks cover 5405 matched graph–palette traces and 1099 graphs testing neighbor-order effects. Eight evaluation questions relate component changes to output validation, objective priorities, computational accounting, and reuse. The checks establish the stated finite cases; benchmark performance and the procedure's practical utility remain questions for further evaluation.

مراجعة نقدية لتمثيل القيود وتقييم طرق تخصيص الموارد التوافقي

أحمد جحا[1,*]، a.jaha@it.misuratau.edu.ly؛ goha_99@yahoo.com، ORCID: 0009-0006-2422-6874،
أنور الهنشيري[2]، ORCID: 0009-0007-8825-7685
[1,2] قسم علوم الحاسوب ، كلية تقنية المعلومات ، جامعة مصراتة ، مصراتة، ليبيا
* المؤلف المراسل

الكلمات المفتاحية

تخصيص الموارد التوافقي؛ تمثيل القيود؛ إنفاذ جواز الحلول؛ ما فوق الاستدلالية؛ التحسين الهجين؛ التحسين القائم على التعلم.

الملخص

قد يؤثر تغيير تمثيل القيود في تخصيص الموارد التوافقي في معنى النموذج أو قوة الاستدلال أو مسار البحث. تفحص هذه المراجعة النقدية للمناهج هذه الفروق في التحسين الدقيق، وبرمجة القيود، والطرق الاستدلالية البنائية، والطرق ما فوق الاستدلالية، والاستدلالية الفائقة، والطرق الهجينة والمدعومة بالتعلم. يستند التركيب إلى 84 مصدرًا وستة أبعاد تحليلية. وتلخص إحصاءات وصفية سنوات النشر وأدوار المصادر، مع تصنيف اثنتي عشرة آلية مختارة بحسب التمثيل الرئيس وعمليات التحقق ونطاق الضمان ونوع الدليل. يتمثل الإسهام في ربط تعديلات المكونات بالأدلة اللازمة لتفسيرها، مع فصل تغير معنى النموذج عن تقوية الاستدلال وتوجيه البحث. فقد تسمح صيغ مختلفة بالحلول نفسها وتدعم استنتاجات مختلفة، وقد يمنع قرار مقبول محليًا استكمال الحل، وتبقى القرارات ثابتة عند تطابق الاستعلامات وقواعد البناء، وقد تتغير عند اختلاف ترتيب الوصول. يربط إجراء من أربع خطوات هذه الفروق بنشر القيود والتفرع المتعلم ودراستين منشورتين للجداول الجامعية على Udine/ITC2007. وتتحقق الشيفرة من 5405 حالات تطابق للمصفوفة والقوائم، ومن أثر ترتيب الجيران في 1099 رسمًا. وتصل ثمانية أسئلة تقييم تعديلات المكونات بالتحقق من المخرجات وأولويات الأهداف وحساب التكلفة وإعادة الاستخدام. يقتصر التحقق الحسابي على الحالات المحددة؛ ويحتاج تقويم الأداء المعياري والفائدة العملية للإجراء إلى دراسات لاحقة.

1. INTRODUCTION

Combinatorial resource allocation (CRA) assigns limited resources under operational constraints and one or more objectives. Scheduling, timetabling, virtual machine (VM) placement, logistics, and packing differ in capacity, eligibility, timing, and objective priorities; these conditions determine feasibility and quality (Beloglazov et al., 2012; Coffman et al., 1997; Pinedo, 2022).

Exact optimization supplies bounds and certificates, while constraint programming (CP) combines domains, propagation, and search (Bixby, 2012; Rossi et al., 2006). Constructive heuristics extend partial solutions; improvement heuristics revise existing ones. Both can operate within metaheuristic or hybrid architectures (Blum

& Roli, 2003; Hooker, 2012). Learning has expanded these options, but hard-constraint guarantees depend on the mechanisms that check or preserve restrictions (Bengio et al., 2021; Mazyavkina et al., 2021).

The review compares methods through the flow of constraint information: representation stores it, search uses it to explore alternatives, and enforcement checks or preserves restrictions. Penalties, decoders, and repair may operate alongside an explicit model, so constraint coverage must be assessed across the implementation (Coello Coello, 2002; Lagaros et al., 2023).

Three research questions define the critical comparison:

RQ1: Which changes in constraint representation alter the feasible set, and which change deductions or decision information while preserving model semantics?

RQ2: When a constructive, reasoning, or learned component is replaced, which retained operations support output validity, and what is the scope of that support?

RQ3: What evidence permits an observed gain to be attributed to representation, inference, or search control under a fixed operational objective and explicit cost accounting?

The synthesis covers 84 sources and links component changes to the evidence needed to assess them. Three analytical propositions organize the comparison: P1, local admissibility need not preserve completion; P2, equivalent complete feasible sets need not support identical deductions; and P3, matched-query invariance (P3a) must be distinguished from changes in access order (P3b). Sections 5.3–5.5 examine these boundaries through source-based mechanisms and explicit examples. A four-step procedure connects the analysis to two mechanism applications and two published timetabling cases, followed by eight evaluation questions. The finite-check protocols and outcomes are reported with the examples.

2. Foundations and Related Work

2.1 Problem Classes and Computational Difficulty

CRA encompasses different mathematical structures. One-to-one assignment admits polynomial algorithms despite factorially many candidate permutations: The Hungarian method and later developments exploit its structure (Edmonds & Karp, 1972; Kuhn, 1955; Munkres, 1957; Tomizawa, 1971). Linear, quadratic, and multi-index assignment share terminology but require distinct formulations (Burkard et al., 2012). Counting assignments alone does not establish a complexity class (Papadimitriou, 1994).

Packing imposes container capacities; vector packing adds dimensions, whereas geometric packing requires spatial non-overlap (Christensen et al., 2017). The classical one-dimensional decision problem—whether all items fit into at most a specified number of capacity-limited bins—is strongly NP-complete (Garey & Johnson, 1979). This classification does not cover every allocation formulation.

Scheduling adds durations, precedence, and availability; a due date used to measure lateness need not be a hard deadline (Brucker, 2007; Pinedo, 2022). Coloring separates adjacent vertices, while clustering groups entities by a linkage or another criterion (Brélaz, 1979; Jensen & Toft, 1995; Johnson, 1967). Their use in allocation requires explicit capacity, timing, and resource interpretations. Interacting restrictions also matter: reversing a job-shop arc can create a directed cycle (Aarts & Lenstra, 2003). Difficulty depends on structure, rather than constraint count or graph density alone (Korte & Vygen, 2018).

Even within educational timetabling, the institution and its restrictions determine the formulation. Schaerf (1999) surveyed distinct school and university variants; Lewis (2008) examined metaheuristic approaches to university timetabling. A timetable should therefore be assessed against its stated variant before performance is compared across methods.

2.2 A Common Assignment Model

For comparison, X is a binary matrix assigning n entities to m resources: $x_{ij} = 1$ when entity i uses resource j and 0 otherwise. Each entity must use exactly one resource, and the sum of assigned demands w_i on resource j must not exceed its capacity W_j. The additive capacity restriction is standard in packing and knapsack problems (Korte & Vygen, 2018). Timing, eligibility, activation, multidimensional demand, and uncertainty require the relevant additional variables or restrictions. Sufficient capacity alone does not establish eligibility.

Hard constraints are mandatory; soft constraints express preferences. Capacity, assignment, compatibility, and precedence describe a requirement's subject, rather than whether it is hard or soft. Search may penalize violations temporarily, but every hard restriction must hold in the final output (Coello Coello, 2002; Rossi et al., 2006).

2.3 Objective Structure and Operational Priorities

Weighted sums produce a scalar objective; Pareto analysis identifies nondominated alternatives and leaves the preferred trade-off to a further decision (Deb, 2001). Lexicographic optimization gives earlier criteria strict priority (Talbi, 2009). If resource minimization precedes a secondary score, fewer active resources win, and the score breaks ties. Both values should be reported. This decision rule does not certify optimality: minimum resource use requires a matching lower bound or another valid proof.

Let Ω denote complete allocations satisfying all hard restrictions. Service requirements may exclude an otherwise compact allocation; a secondary improvement cannot override a declared primary objective.

2.4 Position within Existing Reviews

Existing reviews already provide substantial accounts of constraint handling, hybridization, and learning (Boschetti & Maniezzo, 2024; Dökeroğlu et al., 2024; Lagaros et al., 2023). The present comparison builds on that work. Table I compares five survey perspectives and the Machine Learning for Combinatorial Optimization (ML4CO) competition protocol with the analytical role adopted here. Its purpose is to link a particular component replacement to model semantics, retained validity operations, and the evidence needed to interpret the resulting performance claim.

Broader surveys provide further context. Houssein et al. (2024) classify metaheuristics across global and engineering optimization, while Peng et al. (2021) review graph learning for combinatorial optimization. Wei et al. (2026) organize deep reinforcement learning (DRL) around the environment, state–action–reward design, solver, and evaluation. These perspectives help locate a component within its architecture; the present comparison then asks which model and validity operations remain after that component is changed.

TABLE I RELATION TO SELECTED REVIEWS AND AN EVALUATION PROTOCOL

Source	Existing organizing perspective	Role of the present comparison
Blum and Roli (2003)	Intensification and diversification organize search components, with memory, guidance, and randomization.	Trace what a component replacement changes in the model, decision information, and validity checks. Component analysis and hybridization are established ideas.
Abdipoor et al. (2023)	Collaborative and integrative hybridization organize university timetabling methods and their benchmark or institutional evaluations.	Complement the taxonomy with explicit exchanged information, retained checks, and post-replacement guarantee scope; do not infer cross-domain superiority.
Bengio et al. (2021)	Learning roles, instance distributions, performance measures, surrogate objectives, and generalization structure the methodological discussion.	Connect these evaluation concerns to a stated model and the operations responsible for each output claim.
Cappart et al. (2023)	Graph learning is reviewed in primal solution generation, dual reasoning, and CP, together with cost and generalization issues.	Trace a selected learned component through its interface to the retained model and checks; do not treat prediction as a certificate.
Wei et al. (2026)	A challenge-driven DRL framework connects environment design, state–action–reward formulation, solvers, and benchmark evaluation.	Examine a specified replacement across learning and conventional architectures, separating model semantics, deductions, decision access, and retained validity operations.
Gasse et al. (2022), ML4CO	Three MILP tasks use controlled solver interfaces, quality-over-time integrals, and task-specific evaluation budgets.	Complement task metrics with model correspondence, component responsibility, and initialization or reuse costs; benchmark validation remains necessary.

The comparison concerns analytical roles rather than a claim that earlier reviews omitted validity, evaluation, or hybridization.

3. METHODOLOGY

3.1 Review Design

This critical synthesis of methods uses foundational texts, specialist surveys, and representative primary studies. Its bibliographic unit is a publication; its mechanism unit is a component operating within a stated model. Algorithm families organize the discussion but are not mutually exclusive experimental categories. Kitchenham and Charters (2007) distinguish planning, conduct, and reporting in systematic reviews. The present design uses purposive methodological sampling with explicit selection criteria, extraction fields, and descriptive coding. It examines how claims are supported within the selected cases rather than estimating field-wide prevalence or pooling heterogeneous performance results.

3.2 Source Selection and Scope

Purposive selection retained sources supporting problem definitions, a representation, search or enforcement mechanism, or an evaluation issue. Specialist reviews supplied context; primary studies described mechanisms, and a conceptual preprint informed interpretability evaluation. Additional history and application examples were omitted when they added no distinct mechanism or qualification. Inclusion supports the analytical purpose of the synthesis rather than an estimate of the prevalence of methods across the field.

Figure 1 summarizes source selection. Preliminary removal of 105 records reduced the initial 290 to 185; screening excluded 124 and retained 61 sources. Targeted source identification supplied 23 additional publications on mechanisms, methodological guidance, and foundational or specialist context, giving 84 distinct publications. The preliminary total combines duplicate and other removals; it is not a duplicate-only count. Additional sources remain a separate branch of the accounting rather than being added retrospectively to the initial retrieval total.

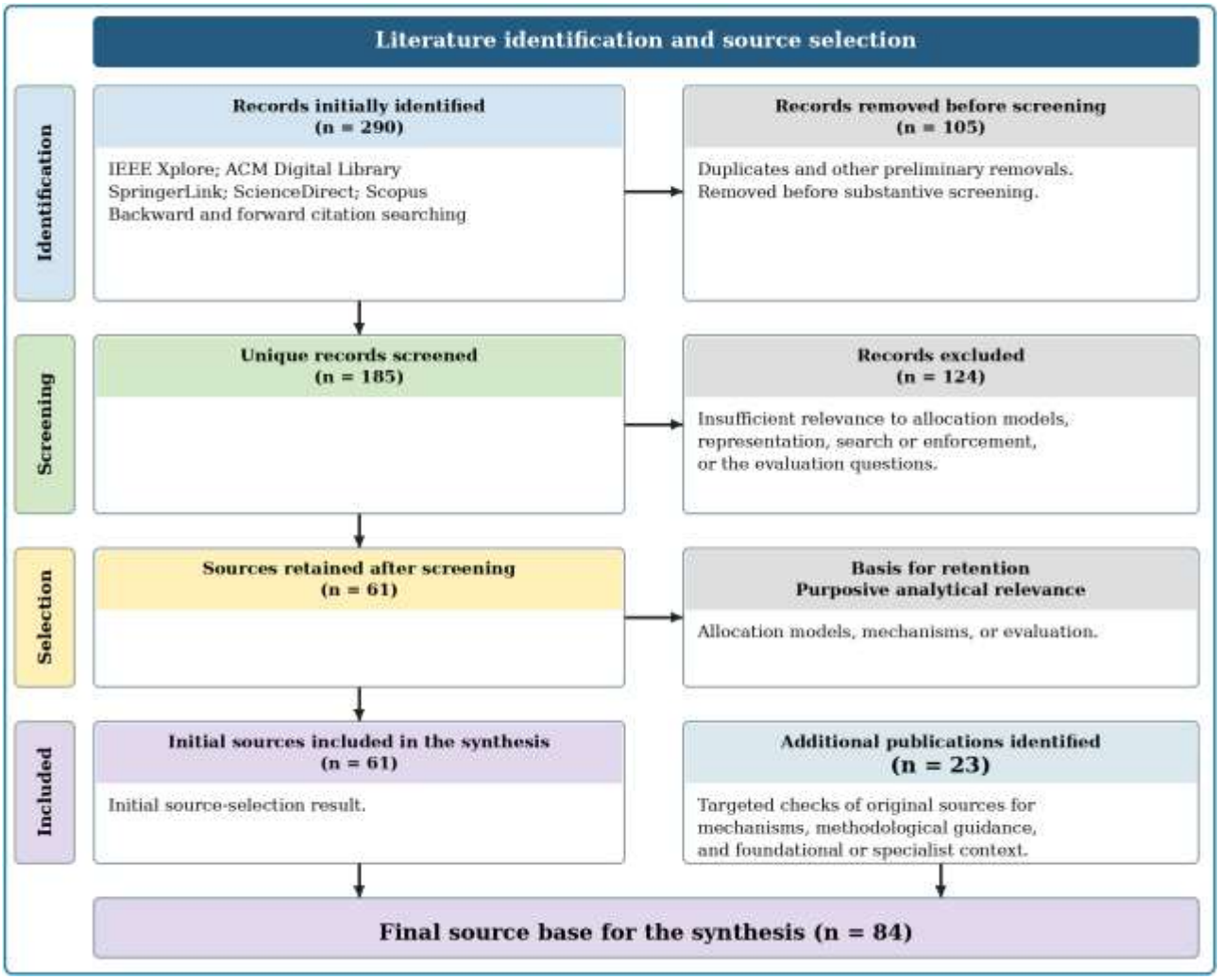


Fig. 1. Source identification and selection. The 61 initially retained sources and 23 additional publications form the 84-source synthesis.

3.2.1 Search Framework and Source Checks

The search framework covered IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus. The following Boolean framework grouped allocation problems, solution approaches, and constraint or complexity concepts:

("Combinatorial Optimization" OR "Resource Allocation" OR "Timetabling" OR "Scheduling" OR "Virtual Machine Placement" OR "Bin Packing")
AND ("Exact Optimization" OR "Integer Linear Programming" OR "Constraint Programming" OR "Constructive Heuristics" OR "Metaheuristics" OR "Hyper-Heuristics" OR "Reinforcement Learning" OR "Deep Reinforcement Learning" OR "Graph Neural Networks" OR "Neural Combinatorial Optimization" OR "Learning to Optimize" OR "Learning-Augmented Optimization")
AND ("Hard Constraints" OR "Soft Constraints" OR "Constraint Satisfaction" OR "Constraint Handling" OR "NP-Hard" OR "Complexity")

Backward citation searching examined foundational works and specialist reviews; forward searching considered methodological extensions and relevant alternative formulations. The initial search-and-download phase ran from 26 April to 15 May 2026 and included reading for preliminary relevance assessment. Preliminary removal followed on 16–31 May, screening and exclusion in June–July, and further source development in August. Publication checks on 14–15 September, claim-level checks on 15–28 September, and bibliographic reconciliation on 29 September distinguished publication identity from evidentiary support. Targeted title, author, subject, and identifier checks continued through 12 November 2026.

Extraction distinguished publication identity and cited version, principal role, mechanism, inspected passage, access level, and the claim supported by that passage. Bibliographic reconciliation checked correspondence between citations and reference entries; claim-level assessment examined whether the inspected material supported the stated mechanism or qualification. Twenty-nine focused support judgments were completed for selected claims. These checks do not imply full-text verification of every claim in all 84 publications. The Boolean expression describes the conceptual search

framework rather than a complete database-by-database retrieval log.

3.2.2 Bibliographic Description and Mechanism Sample

Table II describes all 84 publications by year, principal role, and cited use of seven algorithm families. Each source receives one principal role: foundation, secondary synthesis, primary method or analysis, evaluation background, or conceptual background. Family tags may overlap. Years follow the cited issue or edition rather than an earlier online release; Holland's reprint is counted as 1992. These descriptors summarize the selected literature rather than study quality or field-wide prevalence.

TABLE II PUBLICATION YEARS, PRINCIPAL ROLES, AND CITED ALGORITHM-FAMILY USE

Descriptor	**Category**	**Sources**	Share of 84
Publication year	Before 2019	55	65.5%
Publication year	2019	2	2.4%
Publication year	2020	1	1.2%
Publication year	2021	5	6.0%
Publication year	2022	3	3.6%
Publication year	2023	3	3.6%
Publication year	2024	7	8.3%
Publication year	2025	4	4.8%
Publication year	2026	4	4.8%
Principal role	Foundation	20	23.8%
Principal role	Secondary synthesis	25	29.8%
Principal role	Primary method or analysis	32	38.1%
Principal role	Evaluation background	6	7.1%
Principal role	Conceptual background	1	1.2%
Cited family use	Exact optimization	20	23.8%
Cited family use	Constraint programming	7	8.3%
Cited family use	Constructive heuristics	9	10.7%
Cited family use	Metaheuristics	18	21.4%
Cited family use	Hyper-heuristics	3	3.6%
Cited family use	Hybrid methods	11	13.1%
Cited family use	Learning-assisted optimization	18	21.4%

Year and role blocks each total 84. Family-use counts overlap; all percentages in this table use 84 publications as denominator. Roles and tags describe cited use.

Of the 84 sources, 29 (34.5%) date from 2019–2026 and 18 (21.4%) from 2023–2026. The 32 primary-method or analysis sources are not 32 independent experiments. They yield 31 mechanism records, with source access distinguished during extraction; Sörensen's methodological analysis has no algorithm record. Table IV uses a separate denominator of 12 selected cases, while ML4CO supplies an evaluation protocol.

3.3 Analytical Framework

Six dimensions guided analysis: representation; search control; feasibility enforcement and guarantee scope; computational cost; scalability; and determinism and reproducibility. Each selected mechanism was examined through its model, input state, decision, output, validity operations, guarantee scope, and evidence type. Local admissibility, extendibility, completion, final feasibility, and optimality were distinguished (Section 5.3). A guarantee was recorded with its model and component scope, rather than inferred from the algorithm family.

3.4 Synthesis Strategy

The synthesis mapped search and enforcement operations to architectures and interpreted claims within

each source's model, configuration, instances, and stopping conditions. Table IV reports 12 selected mechanism records spanning propagation, construction, adaptive search, hybridization, and learned control. This is a focused sample for examining the analytical propositions, not a frequency estimate for all 84 sources. Published results, source-based architectural interpretations, and examples developed for this review are identified separately. Sections 5.4–5.5 apply and delimit the comparison procedure; Section 6 connects it to proposed evaluation designs.

4. Optimization Paradigms in Combinatorial Resource Allocation

Table III summarizes overlapping architectural roles. Constructive and improvement heuristics can serve within metaheuristic, hyper-heuristic, hybrid, or learning-assisted methods.

TABLE III REPRESENTATIVE ARCHITECTURES AND THEIR FEASIBILITY RESPONSIBILITIES

Architecture	Information and search role	Feasibility or certificate	Representative sources
Exact optimization	Algebraic models or recursive states; bounds, systematic search, or subproblem reuse	Checked solution; optimality under a valid recurrence or completed proof	Nemhauser and Wolsey (1988); Bixby (2012); Martello and Toth (1990)
Constraint programming	Domains, global constraints, propagation, and search	Sound filtering; complete solutions require all modeled constraints to hold	Apt (2003); Régin (1994)
Constructive heuristics	Priorities with capacity, eligibility, or temporal state	Validity under implemented checks; completion requires success	Blum and Roli (2003); Kolisch (1996)
Metaheuristics	Neighborhood or population control with an evaluator	Depends on decoding, repair, variation, and final acceptance	Coello Coello (2002); Deb (2000)
Hyper-heuristics	Selection or generation of low-level heuristics	Inherited from checked low-level mechanisms and composition	Burke et al. (2013); Dökeroğlu et al. (2024)
Hybrid and matheuristic methods	Coordinated heuristic, CP, or mathematical components	Depends on subproblem scope and sound interfaces	Hooker (2012); Blum et al. (2011)
Learning-based methods	Learned construction, improvement, or solver guidance	Depends on supported masks, checks, or retained solver	Gasse et al. (2019); Khalil et al. (2017); Kool et al. (2019)

4.1 Exact Optimization and Constraint Programming

A feasible incumbent and an optimality certificate are different solver outcomes; a relaxation solution may violate original restrictions (Bixby, 2012; Nemhauser & Wolsey, 1988). Certification requires valid bounds and the necessary proof. The distinction between a relaxation bound and discrete feasibility is already explicit in Land and Doig's (1960) formulation and search procedure. Equivalent integer formulations can yield different relaxations, presolve deductions, and costs. Time discretization can also change the scheduling problem itself (Pinedo, 2022; Schrijver, 1986).

Formulation strength and preprocessing provide concrete instances of this distinction. Achterberg et al. (2020) describe presolve as reducing model size and strengthening the formulation. Kis and Horváth (2022) construct ideal, non-extended formulations for disjunctive constraints whose embedding admits a specified network representation. Such results concern the formulation and its relaxation under stated assumptions; they do not establish a universal runtime advantage for a storage structure.

Dynamic programming reuses subproblem values. For integer weights, the capacity-indexed 0–1 knapsack recurrence stores the greatest profit attainable from the items considered at each capacity limit, comparing exclusion with feasible inclusion (Martello & Toth, 1990). Correctness requires sufficient state; timing or eligibility may demand another decomposition. Constraint programming combines domains, propagation, and search to solve a constraint satisfaction problem (CSP), or to optimize an objective over its satisfying assignments (Apt, 2003). Compared with pairwise decomposition, Régin's (1994) matching-based all-different filtering can strengthen deductions while preserving the same complete assignments.

4.2 Constructive Heuristics and Metaheuristic Search

DSATUR orders vertices by the number of distinct colors on their neighbors (Brélaz, 1979). This constructive rule differs from Brélaz's separate exact procedure and later exact DSATUR algorithms (Lewis, 2021; San Segundo, 2012). Alone, it certifies neither minimum coloring nor omitted allocation restrictions. Kolisch's (1996) serial and parallel schedule-generation schemes separate activity priorities from precedence and resource checks, allowing outer search to revise priorities while retaining the generator.

Clustering can supply groups to an allocation constructor. Agglomerative clustering merges groups by a linkage criterion (Johnson, 1967; Murtagh & Contreras, 2012). Davidson and Ravi (2009) distinguished feasible clusterings from merge paths becoming irreducible too early under some instance-level constraints. Resource capacities and the meaning of a cluster must be specified separately. A cluster may represent a resource or be assigned to one. Repair, backtracking, or alternative orders may revise commitments; a tested portfolio provides no proof of global optimality.

Metaheuristics differ in candidate generation and acceptance (Blum & Roli, 2003; Sörensen, 2015). Genetic algorithms combine population selection and variation (Holland, 1975/1992); simulated annealing permits

temperature-dependent acceptance of worsening moves (Kirkpatrick et al., 1983). Tabu search uses memory, admissibility, and aspiration, without requiring randomness (Glover, 1989). Particle swarm optimization originated in continuous space, so discrete allocation needs a mapping or discrete operators (Kennedy & Eberhart, 1995). Ant colony optimization constructs candidates through pheromone-guided choices (Dorigo & Stützle, 2004).

Their feasibility depends on the representation, operators, evaluator, and any decoder or repair. Penalties alone need not guarantee a valid output. Deb's (2000) rule favors feasible candidates, ranking them by objective and infeasible candidates by violation; it does not ensure that search finds feasibility within its budget. Decoding and repair require sound checks and an explicit failure policy (Coello Coello, 2002; Lagaros et al., 2023).

4.3 Hyper-Heuristics and Hybrid Methods

Hyper-heuristics select or generate low-level heuristics, which may construct solutions or perturb existing candidates; guarantees depend on those mechanisms and their composition (Burke et al., 2013; Dökeroğlu et al., 2024). Algorithm selection relates instance features to performance (Rice, 1976). Selection can occur during calibration, per instance, or during a run. Reuse avoids repeating portfolio search, but calibration and reselection costs still belong in evaluation.

Clustering may organize candidate solutions rather than allocation resources. In the Q-learning-based hyper-heuristic with a clustering strategy (QHH/CS) of Yang et al. (2025), k-means groups candidate permutations after t-distributed stochastic neighbor embedding (t-SNE), while Q-learning selects local search actions. For the permutation flow-shop scheduling problem (PFSP), schedules remain represented by job permutations and evaluated through completion-time recurrences. Clustering supports search organization; it does not itself establish resource-feasibility conditions.

Hybrids distribute search and reasoning across components (Blum et al., 2011; Hooker, 2012; Puchinger & Raidl, 2005). Matheuristics incorporate mathematical programming (Boschetti & Maniezzo, 2024). An exact neighborhood result is conditional on fixed decisions and boundary restrictions; it does not establish global optimality. Interfaces must specify exchanged priorities, assignments, bounds, and repair requests, including communication costs.

Large neighborhood search removes and reconstructs part of a solution. Shaw (1998) combined constraint programming and local search; Ropke and Pisinger (2006) adapted removal and insertion weights from recent performance. Their model permits a penalized request bank, but their reported experiments require all requests to be planned in the accepted output. Valid routes alone therefore do not establish complete service. The final acceptance condition must accompany any comparison of adaptive search.

4.4 Optimization with Machine Learning

Learning can construct or improve solutions, select methods, guide search, or assist formulation. Graphs may encode operational conflicts, variable–constraint structure, or policy features (Cappart et al., 2023; Darvariu et al., 2024). In mixed-integer linear programming (MILP), Gasse et al. (2019) learned branching from a bipartite variable–constraint graph while retaining the solver's model and validity operations.

More recently, Ben Mhamed et al. (2026) used proximal policy optimization (PPO) to select branching variables from tree and candidate features. SCIP retained validity operations. Their evaluation disabled primal heuristics and supplied the known optimal value as an initial cutoff, isolating tree-search efficiency. Those conditions limit extrapolation to solving from scratch, where finding primal bounds also incurs work.

Khalil et al. (2017) used fitted Q-learning and updated structure2vec embeddings for greedy graph construction in vertex cover, maximum cut, and traveling-salesman formulations. Problem-specific helper and termination procedures maintained the required structure; CRA transfer would need its own capacity, eligibility, and timing checks. For capacitated routing without split deliveries, Kool et al. (2019) masked customers already served or whose demand exceeded residual capacity. The split-delivery variant uses different rules. Such masks cover the restrictions encoded in them; policy, problem variant, training distribution, and deployment cost remain separate evaluation factors (Chung et al., 2025; Martins et al., 2025).

Across routing variants, Zhou et al. (2024) combined a mixture-of-experts policy with variant-dependent masks. Open routes change the return-distance and depot-time tests used for closed routes. Reusing the policy architecture therefore does not make the underlying constraint semantics interchangeable; the checks must match the target variant.

A recent CP integration has learned value selection while retaining propagation and solver branching. Marty et al. (2024) combined restart-based reinforcement learning with a tripartite representation and used the policy within iterative limited discrepancy search (ILDS). Their experiments fixed variable selection to minDomain. This intervention concerns value ordering within CP, with the model and propagation engine still governing validity.

Large language models may generate formulations, code, or candidate solutions (Da Ros et al., 2026). Each output needs validation: the formulation against requirements and units, code against the formulation, and allocation against an independent checker. A solver certificate concerns its submitted model; correspondence with the intended operational problem remains a separate issue.

ORLM generates mathematical models and executable solver code (Huang et al., 2025). Its execution-accuracy test compares a computed optimal value with a supplied ground-truth value. In this review's interpretation, agreement on that test supports the tested output but does

not establish equivalence of feasible sets across inputs. Validation of the intended model remains necessary.

4.5 Selected Mechanisms and Evidence

Table IV examines 12 mechanisms across propagation, construction, adaptive search, mathematical submodels, and learned control. Scheduling and routing allocate time or capacity under model-specific restrictions. General MILP, CP, and modeling studies are included to examine component interfaces that can support allocation formulations, rather than as evidence of performance in every CRA domain. Transfer requires the correspondence described in Section 7.5. Table V summarizes representation, validity, guarantee scope, and evidence for these selected cases.

TABLE IV SELECTED MECHANISM RECORDS AND THE SCOPE OF THEIR EVIDENCE

Source and model	Representation and search	Validity and guarantee scope	Evidence
Régin (1994) CSP	Domains and constraint relations: Value graph and matching-based all-different filtering.	Filtering preserves satisfying assignments of that constraint; solving the complete CSP still requires its remaining constraints.	Theory and published algorithm evaluation.
Gasse et al. (2019) MILP	Algebraic model: Variable–constraint graph; learned branching.	The retained solver provides validity operations; learned choice supplies no independent certificate.	Benchmark comparison and component analysis.
Ropke and Pisinger (2006) Pickup and delivery	Route state: Request bank with adaptive removal and insertion.	Route checks and final acceptance differ; the reported experiments require all requests planned.	Model, algorithm, and benchmark experiments.
Kool et al. (2019) Capacitated routing	Route state: Attention policy with residual demand and capacity; visit masks.	Masks support encoded routing restrictions; rules differ for split deliveries.	Benchmark experiments under stated decoding budgets.
Yang et al. (2025) PFSP	Permutation state: t-SNE and k-means group candidate job permutations; Q-learning selects moves.	Permutation structure and completion-time evaluation remain separate from clustering; no finite-budget optimum certificate follows.	Analysis, ablation, and a further routing application.
Beloglazov et al. (2012) VM placement	Placement state: Resource availability and a power model; checked placement and migration.	Current-resource checks support placement; future service and energy require workload evaluation.	CloudSim simulation with generated utilization.
Burke et al. (2010) Timetabling	Algebraic model: Reduced and full models; Surface and restricted dives.	Extendibility relies on the model; a restricted-neighborhood optimum need not be global.	Published Udine/ITC2007 application.
Marty et al. (2024) CP	Domains and constraint relations: Tripartite graph; learned value ordering within ILDS.	Retained CP propagation and branching govern validity; the learned policy supplies no separate optimum certificate.	Experiments and discussion of transfer and computational cost.
Zhou et al. (2024) Multi-variant routing	Route state: Mixture-of-experts policy; variant-dependent masks.	Encoded routing checks differ across variants; architecture reuse does not establish model equivalence.	Benchmark and transfer experiments.
Huang et al. (2025) Automated modeling	Algebraic model: Language model generates formulation and solver code.	Execution accuracy compares objective values; it does not prove feasible-set equivalence.	NL4OPT, MAMO, and IndustryOR tests.
Lin et al. (2024) Thermal-aware VM placement	Placement state: VM–host mapping; differential evolution, particle swarm optimization, and a thermal model.	A candidate-host filter alone does not certify aggregate migration feasibility.	CloudSim with real workload traces; energy and violation metrics.
Ben Mhamed et al. (2026) MILP	Algebraic model: Tree/candidate features; PPO branching.	SCIP retains validity; expert setup uses a known optimum cutoff and disables primal heuristics.	Benchmark comparison under the stated solver setup.

Twelve selected mechanism records are shown. ML4CO is considered separately as an evaluation protocol in Section 7.3. The rows do not count independent experiments or represent all 84 publications.

TABLE V PRINCIPAL REPRESENTATIONS, VALIDITY OPERATIONS, GUARANTEE SCOPES, AND EVIDENCE IN 12 CASES

Descriptor	Principal category	Cases	Share of 12
Principal representation	Algebraic model	4	33.3%
Principal representation	Domains and constraint relations	2	16.7%
Principal representation	Route, permutation, or placement state	6	50.0%
Validity operation	Constraint filtering	1	8.3%
Validity operation	Retained solver or CP engine	4	33.3%
Validity operation	Checked search or placement evaluation	4	33.3%
Validity operation	Variant-specific construction masks	2	16.7%
Validity operation	Generated-model solver execution	1	8.3%
Principal guarantee scope	Single-constraint preservation	1	8.3%
Principal guarantee scope	Retained model and solver	3	25.0%
Principal guarantee scope	Restricted-neighborhood model	1	8.3%
Principal guarantee scope	Encoded construction restrictions	2	16.7%
Principal guarantee scope	Problem-specific checks and acceptance	4	33.3%
Principal guarantee scope	Generated formulation	1	8.3%
Evidence type	Theory with algorithm evaluation	1	8.3%
Evidence type	Benchmark or test-set evaluation	9	75.0%
Evidence type	Workload simulation	2	16.7%

Each block totals 12 cases. The principal representation is the model or candidate state against which the discussed validity operation acts. Algebraic cases are Gasse et al., Ben Mhamed et al., Burke et al., and Huang et al.; domain-and-relation cases are Régin and Marty et al.; the other six use routes, permutations, or placement states. Auxiliary policy graphs, embeddings, and features are distinguished from these principal representations in Table IV. One principal operation, scope, and evidence category is also assigned per case. These classifications describe the selected mechanisms; they do not rank methods or exclude additional representations within an implementation.

5. COMPARATIVE ANALYSIS OF RESOURCE ALLOCATION PARADIGMS

Figure 2 connects the four comparison steps to representation, search, and enforcement. The model defines the admissible output; component interfaces determine which information reaches each decision and check. P1–P3 identify distinct boundaries for interpreting a replacement.

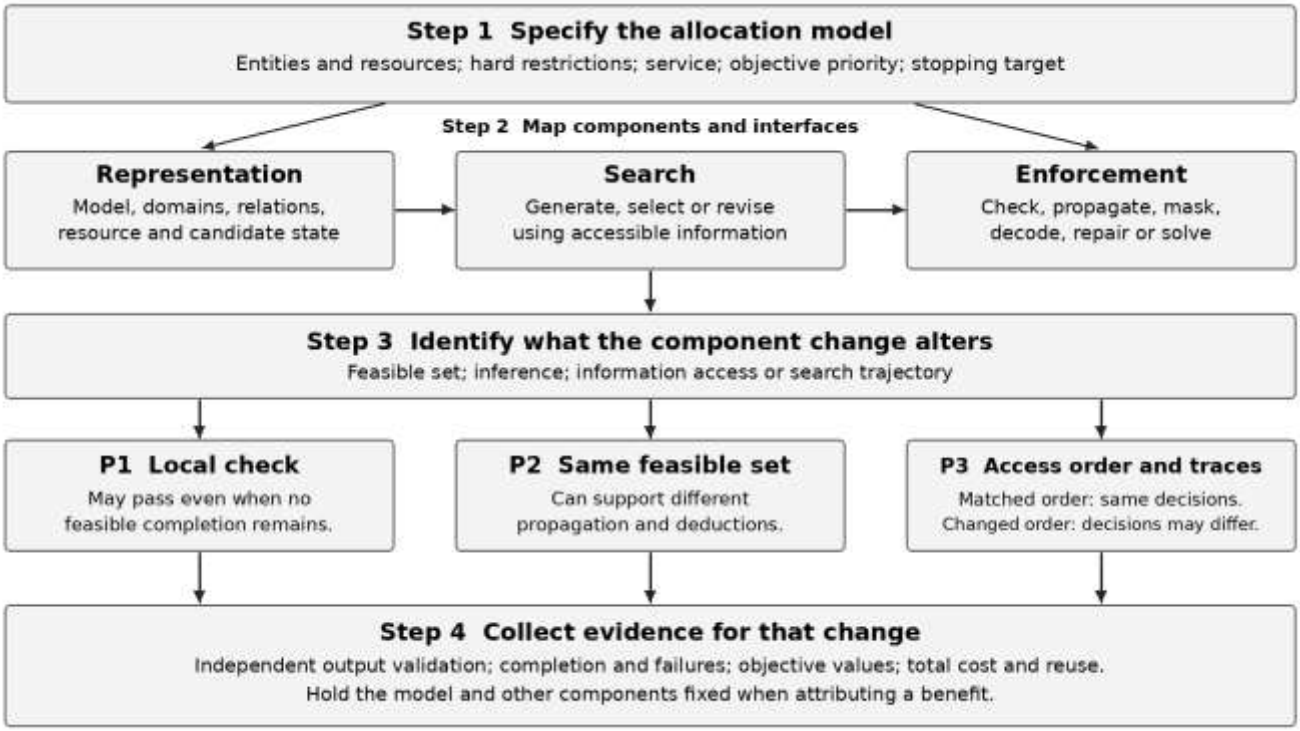


Fig. 2. Four-step component comparison. P1 separates local checks from completion, P2 separates model equivalence from inference, and P3 distinguishes matched-query invariance from access-order effects.

5.1 Computational Characteristics

A comparison must specify its stopping target: first feasibility, a quality threshold, best improvement before a deadline, or certified optimality. Preprocessing, generation, checking, decoding, repair, and deployment preparation all contribute to cost (Bengio et al., 2021; Bixby, 2012). Iterations are not uniform units of work.

Determinism requires fixed inputs, state, rules, and tie-breaking; independent reproduction also needs data, software, and execution details (López-Ibáñez et al., 2021). Tuning and final evaluation should be separated, using held-out instances for generalization (Eiben & Smit, 2011). Henderson et al. (2018) showed experimental sensitivity in continuous-control reinforcement learning; the transferable lesson is to document configuration and data partitioning.

Repeated seeds on one instance measure run-to-run variation, not independent instances. Paired instance comparisons, uncertainty, effect sizes, timeouts, infeasible outputs, and censored runtimes require appropriate treatment. Demšar's (2006) classifier-comparison procedures require their assumptions to be checked before application to allocation. Deterministic decisions do not imply identical runtimes across environments.

5.2 Constraint Representation

Models, domains, conflict graphs, eligibility relations, resource profiles, and learned features expose different information. Penalties and repair act on that information. Search can manipulate priorities while a decoder checks the full model, or learning can coexist with explicit solver constraints (Coello Coello, 2002; Gasse et al., 2019). A representation is useful when its effect on decisions or checks justifies storage, access, and maintenance costs.

Three simultaneous tasks of demand 0.4 each fit pairwise on a capacity-1 resource, yet together require 1.2. Exact rational arithmetic gives pair load 4/5 and triple load 6/5. A graph encoding only pairwise capacity conflicts misses this violation; a running load detects it. The limitation concerns that restricted encoding, rather than graphs generally.

Eligibility creates another bottleneck: three simultaneous activities restricted to the same two rooms cannot all be assigned when each room hosts at most one activity. Enumeration of all eight activity–room assignments yields no injective assignment, although any pair can be assigned. Matching or global all-different reasoning detects this through the activity–resource relation (Régin, 1994). Conflict layers can preserve causes and update locations that their union loses. Equivalent complete feasible sets need not provide identical deductions or access costs.

5.3 Feasibility and Search

Local admissibility means passing the checks applied to a partial allocation. Extendibility requires a complete feasible allocation retaining every fixed decision. With Ω as in Section 2.3, partial assignment S fixes entries D(S) at values s_{ij}. Its feasible completions are:

$$\Omega(S) = \{X \in \Omega : x_{ij} = s_{ij}, \forall (i,j) \in D(S)\} \quad (1)$$

State S is extendible exactly when Ω(S) is nonempty. P1 follows from a two-entity example. Entities A and B each demand one unit, and two resources each have capacity one. A may use either resource; B only the first. Assigning A to the first passes immediate checks but blocks B. Enumerating the four complete assignments leaves exactly one feasible allocation: A on the second resource and B on the first. None retains A on the first. Local admissibility has therefore not preserved completion.

Completion assigns every mandatory entity; final feasibility also satisfies every hard restriction. Quality and optimality need further evidence. Table VI separates enforcement guarantees from unresolved questions. These operations can be combined across algorithm families.

TABLE VI ENFORCEMENT OPERATIONS AND THE LIMITS OF THEIR GUARANTEES

Operation	Information used	What it can establish	What remains unresolved
Penalty evaluation	Objective and violation measures	Preference under declared coefficients	Hard feasibility of the selected output
Feasibility-oriented selection	Validity, objective, and violations	Ordering among evaluated candidates	Generation of a useful feasible candidate
Checked construction or masking	Partial allocation and tested restrictions	Local admissibility under the tested restrictions	Extendibility, completion, and objective quality
Decoder	Encoding and allocation model	Mapping to an allocation; feasibility if checks are sound	Coverage, failure policy, and cost
Repair	Violations and permitted modifications	A modified candidate; feasibility if all checks pass	Success, disruption, and other interactions
Constraint propagation	Domains and structural relations	Sound removal of inconsistent values	Residual search and complete assignment
Exact subproblem	Restricted model and fixed decisions	Feasible completion or certificate within that scope	Boundary interactions and global optimality
Final validation	Complete output, original restrictions, and declared tolerances	Whether the checked output satisfies the model	Search quality and adequacy of the model itself

Evaluation must retain objective priorities and metric denominators. Consolidation can reduce host count, but energy also depends on power states, loads, and operating duration (Beloglazov et al., 2012). Consider two lectures of ten students, rooms of capacities ten and thirty, and two available periods. Using both rooms in one period leaves ten unused seats per assigned lecture on average. Reusing the smaller room over both periods gives zero but lengthens the timetable. Both averages exclude empty room-periods. Preference depends on the objective, not the fit measure alone.

Report failure rates alongside quality among successful outputs. A best-known feasible minimization value is an upper bound, not necessarily the optimum.

5.4 Applying the Comparison Framework

The comparison procedure has four steps. (1) Specify the model, mandatory service, hard and soft restrictions, validator, objective priority, metric denominators, and stopping target. (2) Map the changed and retained components and their interfaces: stored state, exposed information, queries, decisions, updates, and validity checks, using Tables III, IV, and VI; auxiliary policy features remain distinct from the enforced model. (3) Determine whether replacement changes the feasible set, deductions, or search path, and trace the retained guarantee and failure policy through P1–P3. (4) Collect the evidence specified by E1–E8 under an explicit common budget, including failures, independent validation, and initialization, checking, repair, and refresh costs counted once. Equal-step and equal-time comparisons answer different questions. The following applications show the different conclusions supported by these steps.

Propagation illustrates P2. For three variables with domains {1, 2}, {1, 2}, and {1, 2, 3}, pairwise arc consistency removes no value, whereas global all-different propagation leaves only 3 in the third domain. Both formulations admit the same two complete assignments, (1, 2, 3) and (2, 1, 3), as in Régin's (1994) motivating example. The finite check enumerates all 12 domain tuples under both complete-assignment definitions. A separate binary check repeatedly removes a value without a distinct supporting value in another variable's domain until a fixed point; no domain changes. Values supported by the globally feasible tuples reduce the third domain to {3}. This verifies the deductions in the example without benchmarking a propagator implementation. The model is unchanged; evaluation must still distinguish pruning from search effort, memory, and total time.

Learned branching: within the retained MILP solver, learning changes which fractional variable is selected for branching and supplies no separate feasibility certificate. Gasse et al. (2019) evaluated this intervention with root-only cuts and disabled restarts. Their comparison distinguished search-tree size from solving time, which includes feature extraction and policy inference. Attribution must therefore account for solver configuration as well as the policy. The relevant outcome may be first feasibility, incumbent quality, dual progress, or time to proof; these are different targets.

Published timetabling application: Burke et al. (2010) support the same four checks on Udine/ITC2007. First, room occupancy is mandatory, while room-capacity shortfall is penalized. Second, Surface addresses period-related penalties; dives assign rooms or revise periods within fixed days. Third, simultaneous-room bounds

support completion under this model, whereas PeriodFixed and DayFixed restrict different neighborhoods. Fourth, the publicly available author version reports objective 36, lower bound 35, and gap 2.8% for comp04 at 40 normalized CPU units in Figure 8. The gap is (36 − 35)/36, equivalent to (25/9)% or 2.8% to one decimal place. These are published results, not measurements from a new experiment.

The application separates a modeling decision, an extendibility argument, and the scope of a certificate. Making room capacity mandatory or adding room eligibility would change the model and require renewed analysis. The reported gap does not justify claiming global optimality. Unlike the small examples, this application links the procedure to an independently published timetabling evaluation.

Recent local-search application: Ceschia et al. (2026) represent each lecture by a period–room slot. Auxiliary availability lists and course–period conflict counts guide LectureKick generation; explicit preconditions preserve hard feasibility, while sampling biases and simulated annealing control search. Their component analysis separates move-selection and cooling parameters. On comp validation instances, the reported ablation paths change mean cost by −2.37 and +1.68, respectively. These are path-dependent comparisons between parameter configurations, not independent causal effects: the authors note parameter coupling and limited validation coverage. Their 30-run comparison finds significant gains over the local-search baselines, but no significant overall difference in the broader comparison of leading methods. This application illustrates attribution and evidence scope; it neither isolates storage changes nor validates the present comparison procedure.

5.5 Matched Queries and Access-Order Effects

P3a concerns matched queries. A change in storage need not change construction. The graph-coloring example has vertices A, B, and C, edges A–B and B–C, and colors 1 and 2. An adjacency matrix and adjacency lists store the same graph. Both implementations process A, then B, then C, choosing the smallest color absent from the already colored neighbors.

For A, both representations return an empty forbidden-color set, so color 1 is selected. For B, both return {1}, giving color 2. For C, both return {2}, giving color 1. The assignment is therefore (1, 2, 1) in either representation. More generally, identical query responses and a deterministic decision rule preserve identical partial assignments by induction: the same state and response determine the same next choice.

This equivalence assumes identical ordering, tie-breaking, query responses, and matched construction steps; it does not imply equal memory or access costs. Under a wall-clock limit, different costs may permit different numbers of steps. Separate matrix and list constructors were checked on every labeled simple undirected graph with n = 1, …, 5 vertices and every palette {1, …, k}, k = 1, …, n. Each graph was generated by a binary choice for every unordered vertex pair; the constructors processed vertices in ascending order. Their traces recorded the current vertex, forbidden-color set, and chosen color or failure. All 5405 graph–palette cases produced identical traces: 3478 complete colorings passed an independent check of assignment length, palette membership, and all original edges; 1927 constructions stopped identically without an available color. Such a stop does not prove that the graph is uncolorable. The finite checks verify the implementations and examples, while the conditional induction establishes the general matched-query result.

P3b tests the boundary when access order changes. Breadth-first traversal uses a FIFO queue, roots in ascending vertex order, and vertices marked when enqueued. On removal from the queue, each vertex receives the smallest available color from {1, …, n}. Matrix rows expose neighbors in ascending order; adjacency lists expose the same edges in descending order. Across all 1099 labeled simple graphs on one to five vertices, 615 yield different assignments and 18 use different numbers of colors. Sorting the lists ascending restores identical traversal orders and assignments in all 1099 cases. Every complete coloring passes the same independent edge check. For edges AB, AC, AD, AE, BC, BE, CD, ascending traversal A–B–C–D–E gives (1, 2, 3, 2, 3); descending-neighbor traversal A–E–D–C–B gives (1, 4, 3, 2, 2), indexed by A–E. Both colorings are valid.

The changed neighbor order alters the traversal supplied to the greedy rule. P3b therefore exposes a decision-interface change; it does not attribute a quality gain to storage alone. Together, P3a and P3b distinguish matched-query invariance from order-sensitive access. With a fixed model, validator, and deterministic constructor, trace comparison can identify decision changes; if traces agree, measured query and update costs still need separate evaluation. The checks used Python 3.9-compatible standard-library operations, exact fractions where applicable, and no random seeds, external solver, or benchmark download. Their enumerated inputs, decision rules, validation conditions, and outcomes are specified above. These finite counts are boundary checks, not benchmark performance estimates.

6. Research Questions and Evidence Requirements

6.1 Proposed Evaluation Designs

Table VII translates the analysis into eight evaluation questions, E1–E8, distinct from RQ1–RQ3. Attribution to representation requires fixed constraint meaning, objective, ordering, search policy, and budget, with other implementation changes controlled. Completion, feasibility, and quality remain separate outcomes. A residual-capacity vector may supply sufficient aggregate information cheaply; extra detail is valuable only when it changes useful decisions or reduces implementation cost. These questions are proposed tests, not estimates of how frequently published work overlooks an issue.

TABLE VII EVALUATION QUESTIONS AND CORRESPONDING EVIDENCE REQUIREMENTS

Evaluation question	Literature context	What to hold fixed	Evidence to collect
E1: Which retained information changes useful decisions?	Global constraints and construction: Régin (1994); Kolisch (1996)	Feasible model, objective, search policy, and budget	Query, update, memory, completion, and quality measures
E2: Which checks support the claimed output?	Candidate selection and construction masks: Deb (2000); Kool et al. (2019)	Restrictions and required completion conditions	Boundary cases and independent output validation
E3: Which guarantees survive a component change?	Integration and learned branching: Hooker (2012); Gasse et al. (2019)	Component interfaces and unchanged model semantics	Interface checks, retained guarantees, and total cost
E4: When is initial investment worthwhile?	Selection, tuning, and learning: Rice (1976); Eiben and Smit (2011); Bengio et al. (2021)	Request distribution, workload horizon, and allowed reuse	Data, calibration, training, deployment, and retraining cost
E5: Which gains satisfy the operational decision rule?	Packing and consolidation: Coffman et al. (1997); Beloglazov et al. (2012)	Primary objective, service limits, and termination target	Success rates, active-resource count, metric denominators, and cost
E6: Which conclusions reproduce or generalize?	Reproducibility: López-Ibáñez et al. (2021)	Declared fixed factors; vary the factor being tested	Replay artifacts, independent seeds where relevant, distinct instances, and controlled workload changes
E7: Which components transfer across models or domains?	Problem semantics and graph roles: Christensen et al. (2017); Cappart et al. (2023)	Explicit constraint and objective mapping; identify adaptations	Validity under the target model, adaptation cost, and performance on distinct target instances
E8: Which explanations support an allocation decision?	Interpretability and explicit checks: Doshi-Velez and Kim (2017); Régin (1994); Da Ros et al. (2026)	User task, model, and required explanation content	Correct constraint attribution, consistency with the output, and usefulness for revision

6.2 Conceptual Dependence on the Six Dimensions

The six dimensions describe interacting aspects of an implementation. Table VIII varies one stated factor at a time and identifies the evaluation questions whose evidence would change. This is a conceptual boundary check, not a numerical sensitivity experiment or a completeness proof for E1–E8.

TABLE VIII CONCEPTUAL CHANGES IN DIMENSIONS AND THEIR EVIDENCE REQUIREMENTS

Dimension	Controlled change	Questions and implications
Representation	Replace a matrix by adjacency lists, preserving query responses, ordering, and tie-breaking.	E1 and E6: identical decisions can coexist with different memory and access costs; Section 5.5 supplies the trace.
Search control	Replace a branch or value selector while retaining the model and validity operations.	E2–E3 and E6: search effort or quality may change; retained guarantees must be traced through the interface.
Enforcement and guarantee scope	Strengthen filtering while preserving complete feasible assignments.	E2–E3: compare deductions and residual search; unchanged semantics do not imply unchanged inference.
Computational cost	Include initialization and refresh while holding request outputs and the reuse horizon fixed.	E4–E5: the accounting comparison can change even when the allocations do not; Equation (2) identifies the included terms.
Scalability	Vary instance size or capacity pressure while retaining constraint meaning and objective priority.	E5–E7: evidence must match the change; capacity pressure may alter feasibility and is not merely a runtime variable.
Determinism and reproducibility	Vary execution order or seeds where applicable, retaining inputs and model.	E6 and E8: test repeatability and explanation consistency; stochastic variation and distribution shift are different tests.

The mapping is many-to-many. Changing enforcement may also change search cost; changing the workload may alter both the feasible set of a particular instance and the value of reuse. E7 additionally requires an explicit mapping when the target model changes, and E8 requires a user task when explanations are evaluated. No single dimension or question substitutes for the others. The examples show where evidence requirements change without asserting that all implementations respond in the same way.

7. Discussion and Future Research Directions

Practical value depends on how representation, search, and enforcement interact under the chosen model. Assessment should identify the contribution of a particular implementation and the evidence supporting it.

7.1 Information and Component Evaluation

An evaluator may retain a violation vector while search receives only a scalar penalty. Exposing that vector could change decisions. A controlled test would hold the model and objective fixed, measuring query and update costs, memory, feasibility, and quality. Complementary tests can vary search control while retaining enforcement, separating information benefits from search changes and additional computation.

7.2 Explainable and Trustworthy Optimization

Model transparency, decision explanation, sensitivity analysis, and feasibility certification answer different questions. Doshi-Velez and Kim's (2017) conceptual preprint distinguishes evaluation with real-task users, simplified human tasks, and formal proxies. Allocation explanations should identify the relevant demand, limit, assignment, or eligibility relation. Testing whether users detect errors or make valid revisions assesses usefulness; an independent validator still checks the result.

7.3 Scalable Evaluation

Training and calibration may be amortized over reused requests; workload change may require refresh. Conventional methods likewise reuse preprocessing, cached state, and warm starts (Bengio et al., 2021). For $N > 0$, average computational cost is:

$$T_{avg}(N) = \frac{T_{init} + \sum_{k=1}^{N} T_{request,k} + T_{refresh}(N)}{N} \quad (2)$$

Initialization covers data generation, preprocessing, training, or calibration; request terms include search, checking, and repair; refresh covers later updates. Each cost is counted once in the same additive time measure. Equation (2) is an accounting identity, not evidence of speedup or individual latency. Report latency, peak memory, and quality separately; initial portfolio selection need not be charged in full to every later request. For example, initialization cost 1000, cost 2 per request, and no refresh give average costs 12 for 100 requests and 4 for 500 requests in common abstract units. This exact illustration concerns amortization, not measured speedup.

Generalization tests should match the claimed changes in instance size, resource pressure, distribution, or constraint types. Validity checks may still prevent infeasible outputs when quality deteriorates. The Machine Learning for Combinatorial Optimization competition (ML4CO) separates primal solutions, branching for dual progress, and solver configuration, using primal, dual, and primal-dual gap integrals for the corresponding targets (Gasse et al., 2022). In its primal task, SCIP checks proposed solutions. This protocol specifies tasks, interfaces, and evaluation budgets; the questions here connect a component change to retained guarantees and the costs included in the claim. Shared hardware and bounded evaluation time do not by themselves account for earlier training.

Representation costs also affect conventional methods. Compare initialization, memory, lookup, and update costs across entity count, horizon, capacity pressure, conflict density, resource dimensions, and update frequency. Parallelism can reduce elapsed time while increasing processor work; asynchronous updates may alter search unless order is controlled. No universal superiority of a data structure is implied.

The cloud example also illustrates why sustainability needs an operational measure. Beloglazov et al. (2012) checked available resources before comparing incremental power in Modified Best Fit Decreasing (MBFD), then evaluated energy, service-level agreement (SLA) violations, and migrations through CloudSim simulation with generated CPU utilization. These results should be read as simulation evidence. Host consolidation alone establishes neither a measured energy saving nor service compliance under future demand.

A thermal-aware placement study counts both IT and cooling energy (Lin et al., 2024). Its candidate-host filter requires spare resources for at least one migrated VM; that condition alone cannot certify the complete migration map. Extended CloudSim experiments used PlanetLab and Azure workload traces and measured energy, SLA violations, and thermal violations. These are simulation findings, with service and thermal outcomes reported alongside energy.

7.4 Dynamic Allocation under Uncertainty

New requests, failures, capacity changes, and eligibility revisions may require migration or reallocation. Distinguish rebuilding, warm starts, repair, and online decisions; report response time, reassignment, and disruption with feasibility and quality. Stability concerns changed conditions; determinism concerns repeated unchanged input. For uncertain demand, state whether inputs are point estimates, scenarios, or uncertainty sets, including service-risk and recourse assumptions. Nominal validity does not cover every possible workload.

7.5 Transfer across Models and Domains

Transfer requires explicit correspondence between entities, resources, variables, restrictions, and objective priorities. Vector capacity checks do not establish geometric non-overlap (Christensen et al., 2017). Identify the reused component, changed checks, adaptation or retraining costs, and results on independent target instances. Successful reuse does not alone establish model equivalence or domain generalization. Benchmark names should be accompanied by the variant, preprocessing, validator, and scoring rule.

7.6 Limitations

Purposive selection is subject to visibility, accessibility, and selection biases. Publication descriptors concern 84 sources, whereas the representation, guarantee, and evidence counts concern 12 selected cases. Heterogeneous models and benchmarks prevent a pooled performance ranking, and an implementation can span categories. The two timetabling applications interpret published findings. Executable enumeration checks verify the small examples and constructor traces, but they do not measure benchmark speed or validate the procedure's practical utility. Table VIII examines conceptual dependence rather than numerical sensitivity. Dynamic allocation, uncertainty, and explanation remain evaluation issues within this scope.

8. CONCLUSION

The 84-source synthesis compares allocation methods through retained information, guided decisions, and implemented validity operations. RQ1 is addressed by distinguishing a changed feasible set from stronger deductions or changed decision access. Global propagation preserves complete assignments while improving inference; the matrix/list examples distinguish unchanged decisions under matched queries from changes caused by neighbor-access order. RQ2 is answered by identifying the retained validator, constructor, propagator, or solver and tracing its guarantee to the actual model and output conditions.

For RQ3, the four-step procedure and eight evaluation questions connect a component change to output validation, objective priority, failure reporting, and computational accounting. The published timetabling cases distinguish model-specific validity conditions, restricted-neighborhood guarantees, and coupled search-control effects. The decomposition case separates soft capacity shortfall from mandatory room occupancy and a restricted-neighborhood optimum from a global certificate. The representation definitions, source-based mechanism descriptions, and finite-check protocols make the comparison inspectable within the paper. The resulting contribution is an evidence map for component comparisons; further applications can test whether it improves attribution of performance changes and experimental design.

REFERENCES

Aarts, E. H. L., & Lenstra, J. K. (Eds.). (2003). *Local search in combinatorial optimization*. Princeton University Press. https://doi.org/10.2307/j.ctv346t9c

Abdipoor, S., Yaakob, R., Goh, S. L., & Abdullah, S. (2023). Meta-heuristic approaches for the university course timetabling problem. *Intelligent Systems with Applications, 19*, Article 200253. https://doi.org/10.1016/j.iswa.2023.200253

Achterberg, T., Bixby, R. E., Gu, Z., Rothberg, E., & Weninger, D. (2020). Presolve reductions in mixed integer programming. *INFORMS Journal on Computing, 32(2)*, 473–506. https://doi.org/10.1287/ijoc.2018.0857

Apt, K. R. (2003). *Principles of constraint programming*. Cambridge University Press. https://doi.org/10.1017/CBO9780511615320

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems, 28*(5), 755–768. https://doi.org/10.1016/j.future.2011.04.017

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research, 290*(2), 405–421. https://doi.org/10.1016/j.ejor.2020.07.063

Ben Mhamed, A., Kamal-Idrissi, A., & El Fallah Seghrouchni, A. (2026). Learning branching policies for MILPs with proximal policy optimization. *Proceedings of the AAAI Conference on Artificial Intelligence, 40*(29), 24379–24387. https://doi.org/10.1609/aaai.v40i29.39619

Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, Optimization Stories, 107–121. https://doi.org/10.4171/DMS/6/16

Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing, 11*(6), 4135–4151. https://doi.org/10.1016/j.asoc.2011.02.032

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308. https://doi.org/10.1145/937503.937505

Boschetti, M. A., & Maniezzo, V. (2024). Contemporary approaches in matheuristics an updated survey. *Annals of Operations Research, 343*(2), 663–700. https://doi.org/10.1007/s10479-024-06302-z

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM, 22*(4), 251–256. https://doi.org/10.1145/359094.359101

Brucker, P. (2007). *Scheduling algorithms* (5th ed.). Springer. https://doi.org/10.1007/978-3-540-69516-5

Burkard, R., Dell'Amico, M., & Martello, S. (2012). *Assignment problems* (Rev. reprint). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611972238

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society, 64*(12), 1695–1724. https://doi.org/10.1057/jors.2013.71

Burke, E. K., Mareček, J., Parkes, A. J., & Rudová, H. (2010). Decomposition, reformulation, and diving in university course timetabling. *Computers & Operations Research, 37*(3), 582–597. https://doi.org/10.1016/j.cor.2009.02.023

Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., & Veličković, P. (2023). Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research, 24*(130), 1–61. https://jmlr.org/papers/v24/21-0449.html

Ceschia, S., Da Ros, F., Di Gaspero, L., & Schaerf, A. (2026). Revisiting local search for Curriculum-Based Course Timetabling. *Computers & Operations Research, 195*, Article 107621. https://doi.org/10.1016/j.cor.2026.107621

Christensen, H. I., Khan, A., Pokutta, S., & Tetali, P. (2017). Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review, 24*, 63–79. https://doi.org/10.1016/j.cosrev.2016.12.001

Chung, K. T., Lee, C. K. M., & Tsang, Y. P. (2025). Neural combinatorial optimization with reinforcement learning in industrial engineering: A survey. *Artificial Intelligence Review, 58*(5), Article 130. https://doi.org/10.1007/s10462-024-11045-1

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering, 191*(11–12), 1245–1287. https://doi.org/10.1016/S0045-7825(01)00323-1

Coffman, E. G., Garey, M. R., & Johnson, D. S. (1997). Approximation algorithms for bin packing: A survey. In D. S. Hochbaum (Ed.), *Approximation algorithms for NP-hard problems* (pp. 46–93). PWS Publishing.

Da Ros, F., Soprano, M., Di Gaspero, L., & Roitero, K. (2026). Large language models for combinatorial optimization: A systematic review. *ACM Computing Surveys, 58*(11), Article 272. https://doi.org/10.1145/3801961

Darvariu, V.-A., Hailes, S., & Musolesi, M. (2024). Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=HduK51xNtS

Davidson, I., & Ravi, S. S. (2009). Using instance-level constraints in agglomerative hierarchical clustering: Theoretical and empirical results. *Data Mining and Knowledge Discovery, 18*(2), 257–282. https://doi.org/10.1007/s10618-008-0103-4

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering, 186*(2–4), 311–338. https://doi.org/10.1016/S0045-7825(99)00389-8

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. Wiley.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30. https://www.jmlr.org/papers/v7/demsar06a.html

Dökeroğlu, T., Küçükyılmaz, T., & Talbi, E.-G. (2024). Hyper-heuristics: A survey and taxonomy. *Computers & Industrial Engineering, 187*, Article 109815. https://doi.org/10.1016/j.cie.2023.109815

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. MIT Press. https://doi.org/10.7551/mitpress/1290.001.0001

Doshi-Velez, F., & Kim, B. (2017). *Towards a rigorous science of interpretable machine learning* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1702.08608

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM, 19*(2), 248–264. https://doi.org/10.1145/321694.321699

Eiben, A. E., & Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation, 1*(1), 19–31. https://doi.org/10.1016/j.swevo.2011.02.001

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman.

Gasse, M., Bowly, S., Cappart, Q., Charfreitag, J., Charlin, L., Chételat, D., Chmiela, A., Dumouchelle, J., Gleixner, A., Kazachkov, A. M., Khalil, E., Lichocki, P., Lodi, A., Lubin, M., Maddison, C. J., Morris, C., Papageorgiou, D. J., Parjadis, A., Pokutta, S., … Kun, M. (2022). The machine learning for combinatorial optimization competition (ML4CO): Results and insights. In D. Kiela, M. Ciccone, & B. Caputo (Eds.), *Proceedings of the NeurIPS 2021 competitions and demonstrations track* (Proceedings of Machine Learning Research, Vol. 176, pp. 220–231). PMLR. https://proceedings.mlr.press/v176/gasse22a.html

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., & Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems, 32*, 15554–15566. https://proceedings.neurips.cc/paper/2019/hash/d14c2267d848abeb81fd590f371d39bd-Abstract.html

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing, 1*(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence, 32*(1), 3207–3214. https://doi.org/10.1609/aaai.v32i1.11694

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press. (Original work published 1975)

Hooker, J. N. (2012). *Integrated methods for optimization* (2nd ed.). Springer. https://doi.org/10.1007/978-1-4614-1900-6

Houssein, E. H., Saeed, M. K., Hu, G., & Al-Sayed, M. M. (2024). Metaheuristics for solving global and engineering optimization problems: Review, applications, open issues and challenges. *Archives of Computational Methods in Engineering, 31*(8), 4485–4519. https://doi.org/10.1007/s11831-024-10168-6

Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., & Wang, Z. (2025). ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research, 73*(6), 2986–3009. https://doi.org/10.1287/opre.2024.1233

Jensen, T. R., & Toft, B. (1995). *Graph coloring problems*. Wiley. https://doi.org/10.1002/9781118032497

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika, 32*(3), 241–254. https://doi.org/10.1007/BF02289588

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Networks* (Vol. 4, pp. 1942–1948). IEEE. https://doi.org/10.1109/ICNN.1995.488968

Khalil, E. B., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems, 30*, 6348–6358. https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html

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

Kis, T., & Horváth, M. (2022). Ideal, non-extended formulations for disjunctive constraints admitting a network representation. *Mathematical Programming, 194*, 831–869. https://doi.org/10.1007/s10107-021-01652-z

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (Version 2.3, EBSE Technical Report EBSE-2007-01). Keele University and Durham University.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research, 90*(2), 320–333. https://doi.org/10.1016/0377-2217(95)00357-6

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! *International Conference on Learning Representations*. https://openreview.net/forum?id=ByxBFsRqYm

Korte, B., & Vygen, J. (2018). *Combinatorial optimization: Theory and algorithms* (6th ed.). Springer. https://doi.org/10.1007/978-3-662-56039-6

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1–2), 83–97. https://doi.org/10.1002/nav.3800020109

Lagaros, N. D., Kournoutos, M., Kallioras, N. A., & Nordas, A. N. (2023). Constraint handling techniques for metaheuristics: A state-of-the-art review and new variants. *Optimization and Engineering, 24*(4), 2251–2298. https://doi.org/10.1007/s11081-022-09782-9

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica, 28*(3), 497–520. https://doi.org/10.2307/1910129

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum, 30*(1), 167–190. https://doi.org/10.1007/s00291-007-0097-0

Lewis, R. M. R. (2021). *Guide to graph colouring: Algorithms and applications* (2nd ed.). Springer. https://doi.org/10.1007/978-3-030-81054-2

Lin, J., Lin, W., Wu, W., Lin, W., & Li, K. (2024). Energy-aware virtual machine placement based on a holistic thermal model for cloud data centers. *Future Generation Computer Systems, 161*, 302–314. https://doi.org/10.1016/j.future.2024.07.020

López-Ibáñez, M., Branke, J., & Paquete, L. (2021). Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization, 1*(4), Article 14. https://doi.org/10.1145/3466624

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Wiley.

Martins, M. S. E., Sousa, J. M. C., & Vieira, S. (2025). A systematic review on reinforcement learning for industrial combinatorial optimization problems. *Applied Sciences, 15*(3), Article 1211. https://doi.org/10.3390/app15031211

Marty, T., Boisvert, L., François, T., Tessier, P., Gautier, L., Rousseau, L.-M., & Cappart, Q. (2024). Learning and fine-tuning a generic value-selection heuristic inside a constraint programming solver. *Constraints, 29*(3–4), 234–260. https://doi.org/10.1007/s10601-024-09377-4

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research, 134*, Article 105400. https://doi.org/10.1016/j.cor.2021.105400

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics, 5*(1), 32–38. https://doi.org/10.1137/0105003

Murtagh, F., & Contreras, P. (2012). Algorithms for hierarchical clustering: An overview. *WIREs Data Mining and Knowledge Discovery, 2*(1), 86–97. https://doi.org/10.1002/widm.53

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. Wiley. https://doi.org/10.1002/9781118627372

Papadimitriou, C. H. (1994). *Computational complexity*. Addison-Wesley.

Peng, Y., Choi, B., & Xu, J. (2021). Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Science and Engineering, 6*(2), 119–141. https://doi.org/10.1007/s41019-021-00155-3

Pinedo, M. L. (2022). *Scheduling: Theory, algorithms, and systems* (6th ed.). Springer. https://doi.org/10.1007/978-3-031-05921-6

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira & J. R. Álvarez (Eds.), *Artificial intelligence and knowledge engineering applications: A bioinspired approach* (Lecture Notes in Computer Science, Vol. 3562, pp. 41–53). Springer. https://doi.org/10.1007/11499305_5

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 362–367). AAAI Press. https://cdn.aaai.org/AAAI/1994/AAAI94-055.pdf

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers, 15*, 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

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

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier.

San Segundo, P. (2012). A new DSATUR-based algorithm for exact vertex coloring. *Computers & Operations Research, 39*(7), 1724–1733. https://doi.org/10.1016/j.cor.2011.10.008

Schaerf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review, 13*(2), 87–127. https://doi.org/10.1023/A:1006576209967

Schrijver, A. (1986). *Theory of linear and integer programming*. Wiley.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher & J.-F. Puget (Eds.), *Principles and practice of constraint programming—CP98* (Lecture Notes in Computer Science, Vol. 1520, pp. 417–431). Springer. https://doi.org/10.1007/3-540-49481-2_30

Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research, 22*(1), 3–18. https://doi.org/10.1111/itor.12001

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Wiley. https://doi.org/10.1002/9780470496916

Tomizawa, N. (1971). On some techniques useful for solution of transportation network problems. *Networks, 1*(2), 173–194. https://doi.org/10.1002/net.3230010206

Wei, S., Huang, C., Wang, Z., Wang, Y., Kong, D., Mi, H., & Sun, Z. (2026). Deep reinforcement learning for combinatorial optimization problems: A challenge-driven methodology and systematic review. *Mathematics, 14*(14), Article 2538. https://doi.org/10.3390/math14142538

Yang, Y.-Y., Qian, B., Li, Z., Hu, R., & Wang, L. (2025). Q-learning based hyper-heuristic with clustering strategy for combinatorial optimization: A case study on permutation flow-shop scheduling problem. *Computers & Operations Research, 173*, Article 106833. https://doi.org/10.1016/j.cor.2024.106833

Zhou, J., Cao, Z., Wu, Y., Song, W., Ma, Y., Zhang, J., & Xu, C. (2024). MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, & F. Berkenkamp (Eds.), *Proceedings of the 41st International Conference on Machine Learning* (Proceedings of Machine Learning Research, Vol. 235, pp. 61804–61824). PMLR. https://proceedings.mlr.press/v235/zhou24c.html