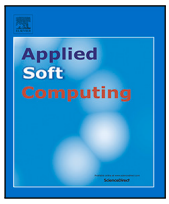
Applied Soft Computing Journal 180 (2025) 113368

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

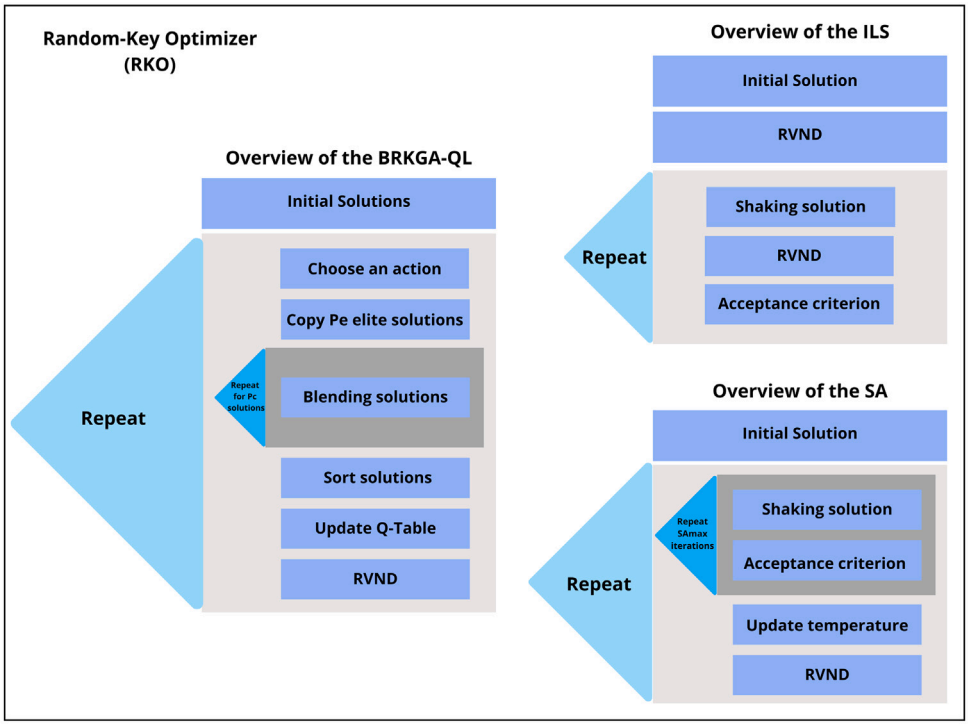


Random-key algorithms for optimizing integrated Operating Room Scheduling

Bruno Salezze Vieira [a,b,c,*], Eduardo Machado Silva [a,b], Antônio Augusto Chaves [a]

[a] Federal University of São Paulo (UNIFESP), São José dos Campos, Brazil
[b] Aeronautics Institute of Technology (ITA), São José dos Campos, Brazil
[c] Southampton Business School, University of Southampton, Southampton, United Kingdom

GRAPHICAL ABSTRACT



ARTICLE INFO

Keywords:
Surgery scheduling
Metaheuristic
Reinforcement learning
Random-key optimizer

ABSTRACT

Efficient surgery room scheduling is essential for hospital efficiency, patient satisfaction, and resource utilization. This study addresses the challenge as a combinatorial optimization problem that incorporates multi-room scheduling, equipment scheduling, and complex availability constraints for rooms, patients, and surgeons, facilitating rescheduling and enhancing operational flexibility. To solve such a problem, we introduce multiple algorithms based on a Random-Key Optimizer (RKO), coupled with relaxed formulations to compute lower

* Corresponding author at: Southampton Business School, University of Southampton, Southampton, United Kingdom.
E-mail addresses: bsvieira@unifesp.br (B.S. Vieira), machado.silva@unesp.br (E.M. Silva), antonio.chaves@unifesp.br (A.A. Chaves).

https://doi.org/10.1016/j.asoc.2025.113368
Received 23 September 2024; Received in revised form 29 April 2025; Accepted 19 May 2025
Available online 11 June 2025

bounds efficiently, rigorously tested on literature and new, real-world-based instances. The RKO approach decouples the problem from the solving algorithms through an encoding/decoding layer, making it possible to use the same solving algorithms to multiple room scheduling problems case studies from multiple hospitals, given the particularities of each place, even other optimization problems. Among the possible RKO algorithms, we design the heuristics Biased Random-Key Genetic Algorithm with Q-Learning, Simulated Annealing, and Iterated Local Search for use within an RKO framework, employing a single decoder function. The proposed heuristics, complemented by the lower-bound formulations, provided optimal gaps for evaluating the effectiveness of the heuristic results. Our results demonstrate significant lower- and upper-bound improvements for the literature instances, notably in proving one optimal result. Our strong statistical analysis shows the effectiveness of our implemented heuristic search mechanisms. Furthermore, the best-proposed heuristic efficiently generates schedules for the newly introduced instances, even in highly constrained scenarios. This research offers valuable insights and practical solutions for improving surgery scheduling processes, delivering tangible benefits to hospitals by optimizing resource allocation, reducing patient wait times, and enhancing overall operational efficiency.

1. Introduction

Surgeries are crucial in hospital management, impacting patient outcomes, hospital efficiency, and overall healthcare system resilience. Effective surgical scheduling optimizes resource utilization, reduces service delivery costs, shortens patient wait times, and increases hospital admissions. However, operating room (OR) scheduling remains a persistent challenge due to increasing demand, constrained hospital resources, and the complexity of coordinating multiple interdependent factors. The integrated surgery scheduling problem emerges as a critical optimization challenge, requiring the efficient allocation of surgical procedures while considering various operational constraints and objectives [1,2].

Healthcare resiliency issues, such as workforce shortages, supply chain disruptions, and infrastructure limitations, further exacerbate the complexities of surgery scheduling. The availability of key resources – including operating rooms, pre- and post-operative care beds, Intensive Care Units (ICUs), and Post-Surgery Care Units (PSCUs) – is often constrained by staffing limitations and fluctuating patient demand [e.g., 3]. Additionally, disruptions in medical supply chains and the increasing burden of chronic illnesses contribute to scheduling inefficiencies, leading to prolonged wait times and suboptimal resource utilization. The rise of cybersecurity threats and interoperability challenges in hospital IT systems also complicate real-time scheduling adjustments [e.g., 4].

To address these challenges, integrated surgery scheduling problems aims to optimize the allocation of surgeries by considering multiple constraints, including room availability, required medical equipment, surgeon schedules, and patient care pathways. The primary goal is to optimize surgical scheduling to improve resource utilization, reduce patient waiting times, and enhance hospital resilience. Effective scheduling strategies enable hospitals to adapt to disruptions, ensure service continuity, and enhance overall patient care [e.g., 2,5–7].

From the patient's perspective, Fig. 1 visually represents the surgery process. It begins with the patient arriving at the hospital and progressing through the following steps: (*i*) Resting room: Upon arrival, the patient is admitted and taken to a resting bed in the assessment area. Diagnostic tests and observations are conducted to assess the patient's medical condition; (*ii*) Operating Room (OR): Subsequently, the patient is taken to the OR to undergo the surgical procedure; (*iii*) Post-Surgery Care Unit (PSCU): After surgery, the patient is transferred to the PSCU for critical observation and anaesthesia recovery. This step ensures the patient's safe transition from the surgical procedure; (*iv*) Intensive Care Unit (ICU) (if applicable): Depending on the surgery's risk level or the patient's condition, there might be a scheduled transfer to the ICU for specialized care and monitoring; (*v*) Recovery Room: Following the critical observation and ICU (if required), the patient is taken back to the same room from step (*i*) for a recovery period. During this phase, the patient receives focused care and support to aid their healing process. Once the medical team determines the patient is stable and ready to leave the hospital, they are discharged (*vi*) with relevant post-surgery instructions and prescriptions, if necessary. A 3D model of the described sequence, even though without the PSCU components, can be found in [8].

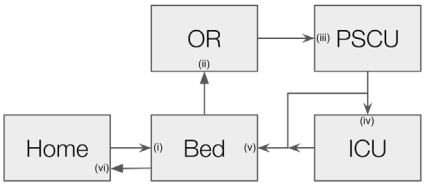


Fig. 1. Patient surgery flow.

Considering this process, the problem we refer to as the Integrated Operating Room Scheduling Problem (IORSP) involves managing a set of surgical procedures, surgeons, operating rooms, and necessary equipment. A solution is to assign each patient to a suitable sequence of rooms based on predefined time slots at different stages within the previously outlined workflow. This allocation aims to schedule pending surgeries to minimize the total execution time, a metric referred to as makespan. This optimization problem takes into account a range of critical factors and constraints, which include:

- Surgeon availability: The scheduling process must account for the availability and preferences of surgeons. Surgeons often have different specialities and expertise, and their schedules must be synchronized with the allocated ORs. In this case study, each surgery already has a surgeon selected for it;
- Operating room allocation: The problem entails allocating surgical procedures to the available operating rooms (ORs), which are only accessible during business hours, to minimize idle time and maximize utilization. Additionally, certain surgeries may necessitate specific equipment or specialized facilities unique to certain ORs, which adds complexity to the allocation process;
- Surgery duration: Each surgical procedure has an estimated duration, which must be considered when scheduling surgeries sequentially. Accurate estimation of surgery duration is crucial for avoiding delays and conflicts. The patient's time in all surgery steps must be accounted for when scheduling the multiple rooms;
- Equipment availability: The availability of necessary equipment for different kinds of surgeries, such as ophthalmic and brain microscopes or video/endoscopy racks, is considered during the scheduling process. Ensuring that all required resources are available at the right time is essential for smooth operations;
- Time constraints: The scheduling problem also considers time constraints, such as surgeon availability, business hours, moving time between rooms, and room maintenance post-use.

Building upon the research of Xiang et al. [9], Burdett and Kozan [10], and Vali et al. [11], we establish that the Integrated Operating Room Scheduling Problem (IORSP) can be effectively conceptualized and addressed as a variant of the Flexible Job Shop Problem

(FJSP) [12]. This research has been guided by the collaboration with a non-profit hospital that has unities in several cities in Brazil, basing our approach on a real-world case. This paper reviews existing literature alongside a comparative analysis with analogous scenarios and methodologies.

In this paper, we propose three metaheuristics using the Random-Key Optimizer (RKO) concept [13] to solve the IORSP: Biased Random Key Genetic Algorithm with Q-Learning (BRKGA-QL) [14], Simulated Annealing (SA) [15], and Iterated Local Search (ILS) [16]. The BRKGA-QL has demonstrated efficacy without necessitating parameter fine-tuning, which is crucial for real-case hospital applications. Furthermore, we introduce two mathematical models for relaxed cases, enabling more efficient computation of lower bounds compared to a complete formulation [17].

To our knowledge, this is the first work among ORSP literature to (*i*) propose multiple solution methods that are completely decoupled from the original problem and (*ii*) propose relaxed formulations to compute lower bounds efficiently. For the RKO literature, (*iii*) a proposed local search has been completely decoupled from the original problem, operating solely in the random-key solution space. With it, we tested available literature instances, successfully adapting our methods to solve them and comparing our results with existing literature methods. Finally, we designed, tested, and made a set of 20 instances based on our case study, which is available online. We can highlight our combined scientific contributions as:

- Proposal of two lower bound formulations that have small implementation complexity and yield better lower bounds than previous literature approaches;
- Investigation of flexible time constraints for each restricted schedule, where each room and surgeon's working hours can be delimited as available or not by the minute (one minute was our time discretization approach, but the user can change it);
- Introduction of a novel concept of a random-key optimizer with three metaheuristics with local searches using the same decoder process;
- Development of a parameter-less metaheuristic that performs better than tuned ones and literature algorithms for similar hospital cases.
- Our in-depth statistical performance analysis shows the effectiveness of the implemented heuristic mechanics.
- Validation of our methods against a similar literature case and demonstration of better results than previous ones, also being able to prove optimal solutions;
- Complete modelling of a real-case scenario and the production of 20 procedurally generated instances with the combined knowledge of other works and making them available in a public repository;
- Demonstration of possible rescheduling using the same input data approach.

The remainder of the paper is structured as follows. In Section 2, we provide a literature review of the most similar works. In Section 3, we define the problem description. Section 4 details our proposed metaheuristics using the random-key concept, decoder, and reinforcement learning component. In Section 5, we present our experimental data, results, and analyses. In Section 6, we further discuss the results and conclude the paper. Our mathematical formulations to compute lower bounds are shown in Appendices A and B. The computational results of these models are in Appendix C.

2. Literature review

In this section, we provide a literature review on the ORSP and a review of the RKO literature. We perform a systematic review of the literature using the string ((“Operat* Room Schedul*”) AND hospital AND optimi*) in the scopus base.

2.1. ORSP literature

The modelling of surgery scheduling problems began early in the literature, with [18] developing a job shop-based decision support system that integrates mathematical programming, a knowledge base, and database technologies to optimize nurse scheduling, surgeon block allocation, and surgeries, maximizing OR utilization and minimizing overtime costs. Next, the work of Dexter et al. [19] studied bin packing algorithms and fuzzy constraints in operating room management.

Reviewing the literature, the diversity of hospital management worldwide results in real-world case studies that rarely match each other, highlighting the absence of a singular dominant trend and reflecting the variety of challenges in ORSP research [1,20]. Still, similarities can be identified. We categorize this subsection into papers that address OR scheduling using exact methods (mathematical models solved by optimization software such as Gurobi or CPLEX) and heuristic approaches. We also highlight recent studies exploring other approaches, such as multi-objective and stochastic methods.

2.1.1. Exact approaches

Roshanaei et al. [21] introduced Branch-and-Check methods for operating room planning and scheduling, incorporating surgeon specialities to optimize patient-surgeon assignments and priorities. Their exact methods significantly outperformed traditional integer programming formulations on both benchmark instances and procedurally generated test cases.

Yazdi et al. [22] models the Surgery Scheduling Problem as a Multi-Project, Multi-Mode Resource-Constrained Project Scheduling Problem. Their approach treats each surgery as a project composed of sequential activities such as surgery, cleaning, and recovery, with varying precedence constraints. The model incorporates resource allocation flexibility through multiple activity modes and optimizes surgery scheduling using a mixed-integer programming (MIP) framework. It aims to minimize makespan while inserting emergency surgeries using the Break-in-Moments technique.

Roshanaei et al. [23] proposed a Balanced Distributed Operating Room Scheduling model, optimizing OR allocation across multiple hospitals. Their approach minimizes macro (hospital-level) and micro (OR-level) imbalances using mixed-integer nonlinear programming, solved via reformulation-linearization and Benders decomposition. Their method focuses on optimizing workload balance in a multi-hospital network, which is evaluated using real hospital data.

Augustin et al. [24] proposed a data-driven approach for OR scheduling that integrates ICU bed availability into the surgical case assignment process. Their model optimizes patient selection and scheduling, maximizing OR utilization, while considering ICU congestion and cancellation risks, using a MIP formulation with probabilistic constraints. However, their model does not explicitly consider patient priorities, nor does it incorporate detailed resource scheduling for nurses, anaesthetists, and equipment. Their method was evaluated on the practical case of the teaching hospital Sainte-Justine in Montreal, Canada.

2.1.2. Heuristic approaches

Among the works most similar to our approach, we have the works of Fei et al. [25], which proposed a tactical operating room planning model based on an open scheduling strategy. Their approach assigns surgical cases to multifunctional ORs while optimizing utilization and minimizing overtime costs using a column-generation-based heuristic. Their model assumes that all resources except surgeons are always available, without considering dynamic rescheduling. Fei et al. [26] extended their previous work by incorporating a two-phase approach for weekly surgery scheduling. They first solve the planning problem using a column-generation heuristic and then optimize the daily schedule with a hybrid genetic algorithm, considering recovery room

constraints. Liu et al. [27] modelled OR allocation with five days of planning considering working hours and surgeon scheduling and solved using dynamic programming heuristics for procedurally generated instances and a Belgian university hospital case study. Xiang et al. [9] treated the problem as an FJSP variant, scheduling pre and post-surgery rooms, nurses, and anaesthetists individually, similarly to our approach. They solved it with an Ant Colony metaheuristic.

Molina-Pariente et al. [28] proposed a case that may allocate an assistant surgeon to shorter surgery times; they propose a mixed-integer linear programming formulation and an iterative constructive method to optimize schedules. Aringhieri et al. [29] modelled OR and recovery bed scheduling at San Martino University Hospital over a one-week horizon, integrating weekend stays without surgeries and multiple surgical specialities. Their approach employs a two-stage tabu search metaheuristic that first assigns OR blocks to specialities and then schedules patients, optimizing societal costs. However, it does not explicitly model simultaneous resource allocation. Landa et al. [30] address the OR planning problem, integrating assigning surgery dates and OR blocks and sequencing surgeries in each OR under uncertain surgery durations, aiming to maximize OR utilization and minimize overtime costs, and solved it using a hybrid simulation-based algorithm exploring neighbourhood search.

Dellaert and Jeunet [31] modelled a case of a Dutch cardiothoracic centre for a 4-week planning horizon, considering the allocation of beds, ORs, and ICUs and maximizing the number of scheduled surgeries. The authors proposed a mathematical model and a VNS metaheuristic that found better results than the model solved with CPLEX in all cases. Durán et al. [32] modelled a case of a Chilean public hospital and developed two models and two algorithms for scheduling interventions on top of an existing OR schedule over a defined period that satisfies patient priority criteria. Burdett and Kozan [10] modelled a case of a university and college-affiliated teaching hospital in Brisbane, Australia, as an FJSP considering bed, OR, PSCU, and ICU allocation. They presented and made available 24 generated instances, and to solve it, they proposed some initial solution heuristics and a Hybrid Simulated Annealing as its main algorithm.

Akbarzadeh et al. [5] proposed a heuristic approach for OR scheduling that integrates nurse re-rostering and nurse-patient assignments to improve resource utilization and maximize OR profit. Their model jointly optimizes surgical case planning and scheduling while adjusting nurse shifts based on actual patient demand and minimizing costs. However, although their model explicitly incorporates nurse re-rostering and assignment, it does not consider operating room equipment or anaesthetist assignments. A two-phase heuristic that uses the linear problem solution generated via column generation to construct a feasible solution is presented. Computational experiments have been conducted with artificial data generated in a controlled and structured manner and real-life data from the Sina Hospital (Tehran, Iran). Zhu et al. [33] solved a case study of an affiliated hospital of the University of Science and Technology of China, maximizing OR utilization on a one-week planning horizon, OR to surgery to surgeon assignment, and patient priorities. The authors presented a mathematical formulation and two hybrid metaheuristics to obtain better solutions. Thomas Schneider et al. [34] investigated the scheduling of surgical groups while considering ORs and downstream beds with varying availability over a 15-day planning horizon. Based on the Leiden University Medical Center, they proposed a single-step integer model that maximizes OR utilization while minimizing bed usage variation through weighted objectives. They also developed a simulated annealing approach, demonstrating that their method reduces the number of required beds compared to previous results.

Lin and Li [35] solved a case of surgery to OR assignment, five days planning horizon and minimizing the operation costs and overtime usage, provided an improved mathematical model of Fei et al. [25] and a metaheuristic Artificial Bee Colony that obtained better solutions for larger instances. Park et al. [36] modelled a case of a Korean university hospital considering surgeons' preferences and cooperative operations, in which surgery can have multiple surgeons cooperatively and sequentially to reduce its execution time, and the co-oped surgeries have a lower total execution time. This feature helps to minimize the total overtime and the number of ORs used. A mathematical model and a metaheuristic were proposed to solve it.

2.1.3. Variants of the problem

When considering OR scheduling under uncertainty conditions, we have the works of several researchers. Siqueira et al. [37] solved a stochastic model for a long-term plan from a case study of the Brazilian National Institute of Traumatology and Orthopedics, which considers OR and recovery ward allocations with downstream constraints, using simulations and optimal action-taking. Khaniyev et al. [38] proposed a next-day OR scheduling model that accounts for uncertain surgery durations. Their model assumes a predefined sequence of surgeries and optimizes the weighted sum of idle time, patient waiting time, and overtime costs. Their approach focuses on fixed-sequence scheduling in a single OR containing all the necessary surgery resources. They developed an exact recursive analysis method and proposed three heuristics: Expectation (using expected durations), Myopic (independent surgery scheduling), and Veteran (allowing early starts). The experiments were conducted on randomly generated instances. Gür et al. [8] proposed a two-stage OR scheduling model integrating Constraint Programming and Goal Programming to balance surgical team workload and maximize OR utilization. Their method focuses on structured workforce balancing and minimizing idle time, with some assumptions such as a sufficient number of personnel and full availability of necessary resources to carry out the operations. Addis et al. [39] proposed a cardinality-constrained robust optimization model for OR scheduling, addressing uncertain surgery durations. Their model incorporates patient waiting times, urgency levels, and delay penalties in the objective function, ensuring operational resilience. The model was tested on real hospital data from San Martino Public Hospital, Italy. For a deeper understanding of patient scheduling under uncertainty, we recommend the papers of Addis et al. [39]; Rahimi and Gandomi [40], which explore robust optimization techniques and mathematical modelling for OR scheduling under uncertainty conditions.

We can highlight several works when considering multi-objective papers for ORSP. Marques and Captivo [41] modelled a case of Lisbon's public hospital, allocating only ORs in a one-week planning horizon. The bi-objective problem maximizes the number of surgeries and room occupations with surgeon specialities and is solved with a bi-objective evolutionary metaheuristic. Hamid et al. [42] modelled a public hospital case in Tehran as a multi-objective problem. They schedule ORs considering patient priorities and surgical team members' decision-making styles to improve the surgical teams' compatibility level, minimizing the total cost, overtime OR utilization, and maximizing team consistency. They presented a mathematical model and implemented two metaheuristics to obtain better solutions: a Non-dominated Sorting Genetic Algorithm and a Multi-Objective Particle Swarm Optimization. Mazloumian et al. [3] proposed a multi-objective integrated model for OR scheduling that simultaneously addresses the Master Surgical Schedule Problem and the Surgical Case Assignment Problem. They developed a deterministic integer programming model as a baseline and two robust optimization models, both designed to handle uncertainty in surgery durations and emergency arrivals. The objective function is the minimization of patient waiting times, surgery postponements, and OR utilization deviations. Equipment availability is not explicitly modelled in their work, as OR block times are assumed to include full personnel resources and necessary medical devices. Lotfi and Behnamian [4] proposed a multi-objective scheduling model for distributed OR planning in hospital networks, considering emergency arrivals, inter-hospital transportation, and shared resources, including ORs, surgeons, nurses, and medical equipment. However, it does not explicitly model allocating multiple specific resources per surgery, as they

Table 1
Case study comparisons with similar literature works.

Paper	E	H	Objective function	SSA	MRS	TS	PP	SE	PH	OT	ORS	MO	SC
Fei et al. [25,26]; Liu et al. [27]	X	X	Costs			X			X	X			
Aringhieri et al. [29]; Landa et al. [30]	X	X	Costs			X		X	X				
Molina-Pariente et al. [28]	X	X	#Surgeries			X	X	X	X				
Marques and Captivo [41]		X	Usage			X	X	X	X			X	
Xiang et al. [9]		X	Makespan	X	X	X		X		X	X		
Durán et al. [32]		X	Usage	X			X	X	X	X			
Dellaert and Jeunet [31]	X	X	Usage		X				X				
Siqueira et al. [37]	X		#Surgeries			X		X			X		X
Burdett and Kozan [10]	X	X	Makespan		X	X							
Hamid et al. [42]	X	X	Usage and MSWT			X	X		X	X		X	
Roshanaei et al. [21]	X		Usage	X		X		X	X				
Roshanaei et al. [23]	X		Usage		X				X				
Khaniyev et al. [38]	X	X	Costs			X			X	X			X
Zhu et al. [33]		X	Costs	X				X	X	X			
Thomas Schneider et al. [34]	X	X	Usage			X		X	X	X			
Yazdi et al. [22]	X		#Surgeries	X	X	X			X				
Akbarzadeh et al. [5]	X	X	Costs			X			X		X		
Park et al. [36]	X	X	Active ORs and OT			X				X	X	X	
Augustin et al. [24]	X		Usage	X	X	X	X	X	X				X
Lotfi and Behnamian [4]	X	X	Makespan and Costs		X	X	X	X	X	X		X	X
Mazloumian et al. [3]	X		#Surgeries and Usage	X		X		X	X	X		X	X
Gür et al. [8]	X		Usage	X	X	X		X	X		X		X
Ma et al. [6]	X	X	Costs			X	X	X	X		X	X	
Addis et al. [39]	X		#Surgeries			X	X		X				X
Azab et al. [7]	X		Costs and Usage	X	X	X		X	X	X		X	X
This Work	X	X	Makespan		X	X	X		X		X		

are treated as aggregated hospital resources. Their approach employs a multi-objective learning Variable Neighbourhood Search to minimize surgery completion time, patient allocation costs, and OR overtime, balancing efficiency and cost. Ma et al. [6] proposed a knowledge-based multi-objective evolutionary algorithm for home health care routing and scheduling with multiple centres. Their MIP model minimizes total service cost and tardiness, considering caregiver skill levels, workload balancing, and patient time constraints. The algorithm integrates genetic operators with knowledge-based local search to improve solution quality. Computational experiments compare the proposed algorithm with five benchmark algorithms and a mathematical programming solver. Azab et al. [7] proposed a bi-objective stochastic model for OR scheduling that considers time constraints, surgeon preferences, and collaborative surgeries. Their model uses stochastic optimization with the Sample Average Approximation method to handle uncertain surgery durations, aiming to balance operating costs and surgeon satisfaction. However, resource allocation is indirectly addressed by assuming ORs with fixed nursing teams and equipment.

2.1.4. Summary of the literature review

Table 1 presents the similarities identified in our literature review, categorizing studies based on whether they were solved using an exact (E) or heuristic (H) method, their objective function focus, and key problem attributes. These attributes include surgery-to-surgeon assignment (SSA), multi-room scheduling (MRS), time slot allocation (TS), patient prioritization (PP), surgeon specialities (SE), limited planning horizon (PH), overtime costs (OT), and other resource scheduling (ORS) such as nurses, anaesthetists, or equipment. Additionally, the table highlights whether the approach is multi-objective (MO) or stochastic (SC). We observe that the least explored aspects in the literature are papers focusing on ORS, followed by PP, SSA and MRS.

For more comprehensive and recent reviews of ORSP models and mixed approaches, we advise the works of Al Amin et al. [1]; Aktaş et al. [2]. Al Amin et al. [1] provides a comprehensive review of Operating Room Scheduling (ORS) research from 2000 to 2023, analysing key factors, optimization techniques, and solution approaches under deterministic and uncertain conditions. The study highlights real-world constraints, such as resource limitations, staff availability, and patient variability, which significantly impact scheduling. Aktaş et al. [2] provides a comprehensive review of operating room and surgical team scheduling, emphasizing the importance of efficient resource utilization, patient safety, and staff workload balance. The study highlights key scheduling factors, such as personnel availability, equipment readiness, and sterilization conditions. By analysing 29 research articles, the review categorizes constraints, scheduling strategies, uncertainties, and solution methods, stressing the benefits of integrating surgical team scheduling into OR planning to enhance overall efficiency.

2.2. RKO literature

To address the gaps identified in the literature review, this study proposes the application of the Random-Key Optimizer (RKO) method to solve an integrated ORSP, considering multi-room scheduling, time slot allocation, patient priorities, and resource scheduling, including nurses, anaesthetists, and equipment.

A RKO is an optimization heuristic that operates in the continuous random-key space $[0, 1)^n$, where n represents the size of the random-key vector used to encode the problem solution. The pioneer of this approach is the work of Bean [43], which utilizes a genetic algorithm to perform searches in the solution space for various optimization problems. In [44,45], a variation of the RKGA called Biased RKGA (BRKGA) is proposed and evaluated for a wide range of optimization problems. Later, [46–48] introduced a continuous Particle Swarm Optimization (PSO) algorithm combined with a random-key encoding scheme to determine a permutation. Similarly, [49] proposed a Harmony Search (HS) heuristic with random-key encoding to solve a job-shop scheduling problem, where the application of HS operators to harmonies encoded by random keys resulted in feasible scheduling solutions. Pessoa and Andrade [50] presented four constructive heuristics for the flowshop scheduling problem, along with heuristics based on the BRKGA, Iterated Local Search (ILS), and Iterated Greedy Search (IGS), which explore feasible neighbourhoods represented as sequences. In another study, [51] proposed a BRKGA and an ILS approach with a decoder that translates random-key vectors into schedules for the machine-dependent scheduling problem. This decoder, a straightforward construction method based on job permutations, was also adapted for use in Tabu Search (TS) and Simulated Annealing (SA). Andrade et al. [52] proposed the Implicit Path-Relinking (IPR) that also explores the separation between problem and solution spaces, constructing the path entirely within the unit hypercube and leveraging the decoder

for solution evaluation. Recently, the RKO framework was employed by Schuetz et al. [13] for robot motion planning, using the BRKGA and an extension of SA to search the random-key space. In [53], the RKO is implemented using SA, ILS, and Variable Neighbourhood Search (VNS) for the tree hub location problem. Chaves et al. [54] proposed an RKO considering the GRASP metaheuristic and evaluates it for the travelling salesman problem, tree hub location problem, Steiner triple covering problem, node capacitated graph partitioning problem, and job sequencing and tool switching problem. A comprehensive and up-to-date review of the BRKGA can be found in [55,56].

3. Problem definition

Given the literature review and problem characteristics, we model the IORSP as an FJSP variant. The FJSP is usually described as having a set of jobs K to be processed by the machines of set R, each job consists of a sequenced set T_k of tasks, and each task t has to be performed sequentially to complete the job. For each job k, $k \in K$, the execution of task t, $t \in T_k$, requires one machine r out of a set of given machines R_t, $R_t \subset R$, that execute task t. For task t running on machine r, $r \in R_t$, the task time is γ_t^d, the setup time is γ_t^m and cool-down time is γ_t^c. When a task t is complete, but the allocated machine $r \in R_{Next(t)}$ is not immediately available, the start of task $Next(t)$ is delayed. This delay is called blocking, with a blocking limit per task ϕ_t.

Moreover, the FJSP typically follows some assumptions as we do, as the machines and jobs are always available. Once started, an operation cannot be interrupted. There is no precedence among the tasks of different jobs; each task can be processed by only one machine at a time. As the objective function, FJSP usually minimizes the makespan, that is, the amount of time required to complete all jobs or to maximize the total of completed jobs given a planning horizon.

We can translate the FJSP nomenclature to our problem, considering jobs as surgeries and machines as subjects (rooms, types of equipment, and surgeons). Each surgery consists of a sequence of tasks. Our problem has some particularities, like a non-empty initial schedule or simultaneous machines for the same task, as operations (tasks) require a surgeon, which may require some equipment and OR, i.e., different types of machines (subjects) allocated simultaneously. All γ_t^d, γ_t^m and γ_t^c are fixed for any particular subject, so they are called duration (γ_t^d), moving time (γ_t^m) and cleaning time (γ_t^c). The name blocking is a homonym in our work, and we have a fixed blocking limit Φ for all tasks of 15 minutes. Each subject (machine) has its initial availability schedule (structure we call availability slots). As an FJSP variant, the surgery is a job that requires heterogeneous machines working on the same task during the same period to complete it. In our approach, we only tackled minimizing the makespan. We note that maximizing the number of surgeries tends to prioritize the shorter surgeries.

Table 2 shows our modelled structures for our mathematical notations. We can divide them between static and dynamic structures. Among our static structures, the input data consists of a set of equipment E_f for each equipment type f, sets R^{bed} and R^{or} representing the beds and operation rooms, respectively, a set of surgeries K, and each surgery k to be scheduled has a set of tasks T_k. Each task k has a set of required people L_t (in our case study, this set primarily includes patients for all tasks and the assigned surgeon for each surgery), set of required equipment types F_t, set of compatible rooms R_t, expected duration time γ_t^d, moving time γ_t^m and cleaning time γ_t^c. There is an initial set of availability slots for each one of our resources (person, equipment, or room).

Fig. 2 illustrates the concept of availability slots (AS), a timeline for any subject with available and unavailable intervals. The numbers and arrows on the illustration point to the corresponding hour an interval starts or ends. In this example $W = \{0, 15, 24, 39, 72, 87, 120\}$, the subject is available for the first 15 h, becomes unavailable for 9 h (15 to 24), is available again for 15 h (24 to 39), becomes unavailable for the next 33 h (39 to 72), and so on. This AS example could be the availability of a surgeon, operating room, or other subjects. Mathematically, we define W as an ordered set of time instants arranged in increasing order. The initial element signifies the first available time instant, while the subsequent elements represent an alternation of unavailable and available time instants. This data structure is used as an initial timeline availability and is dynamically adjusted per solution.

Table 2
Problem data notations.

Sets	
E_f	Equipment of type f
R^{or}	All OR rooms
R_{bed}	All bed rooms
K	Surgeries
T_k	Tasks of surgery k
L_t	People allocated for task t
F_t	Equipment types required for task t
R_t	Compatible rooms for task t
Attributes	
γ_t^d	Duration time of task t
γ_t^m	Moving time of task t
γ_t^c	Cleaning time of task t

Furthermore, our modelling allows tackling the re-scheduling of surgeries. Surgery re-scheduling is crucial for effectively managing the IORSP due to the hospital environment's dynamic and unpredictable nature. It allows for flexibility and adaptability in emergencies, resource availability fluctuations, and changes in patient conditions. Re-scheduling ensures that urgent surgeries can be accommodated without significant disruptions, minimizes delays and wait times, and optimizes the use of resources. Additionally, it helps cope with cancellations and no-shows by filling gaps efficiently, maintaining high productivity levels, and ensuring timely surgical care. To implement surgery re-scheduling efficiently, we make use of the sets of availability slots with the previously fixed schedules for rooms, patients, equipment and surgeons as shown in detail in Section 5.5.

Appendices A and B present the relaxed models we propose and implement to compute lower bounds to minimize each instance's makespan. The resulting lower bounds, presented in Appendix C, are used to analyse the performance of the heuristics, where smaller gaps indicate a stronger performance of both approaches. Both models solve a relaxed case of the IORSP and exploit the fact that, for the case study and the literature case, the bed is reserved during the entire patient's stay in the hospital. It considers a best-case scenario execution for all surgeries and only allocates surgeries to rooms but does not sequence any task. All patients are treated without delays (blocking), disregarding room or surgeon-restricted schedules, sequencing of tasks, and equipment availability. Removing these constraints, the relaxed problem solved by both models is an allocation problem.

4. Random-key optimizer

The random-key representation was first proposed by Bean [43] for an extension of the genetic algorithm called the Random-key Genetic Algorithm (RKGA). This representation encodes a solution with random numbers in the interval $[0, 1)$. The main idea is that the RKGA searches the random-key space as a surrogate for the original solution space. Points in the random-key space are mapped to points in the original solution space by a deterministic algorithm called the decoder. An advantage of this encoding is its robustness to problem structure, as it separates the solver (solution procedure) from the problem being solved. The connection between the solver and the problem is established through the decoder.

A Random-key Optimizer (RKO) is an optimization heuristic that solves specific problems by exploring the continuous random-key space $[0, 1)^n$, where n is the size of the random-key vector that encodes the

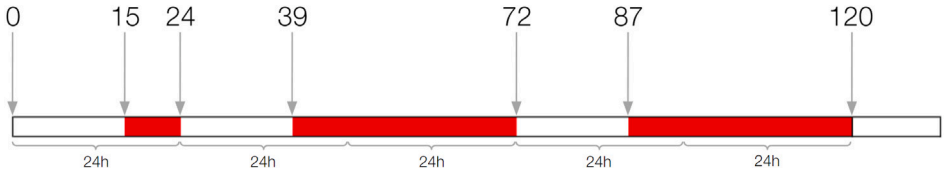


Fig. 2. Example of a set of availability slots (AS) for a subject.

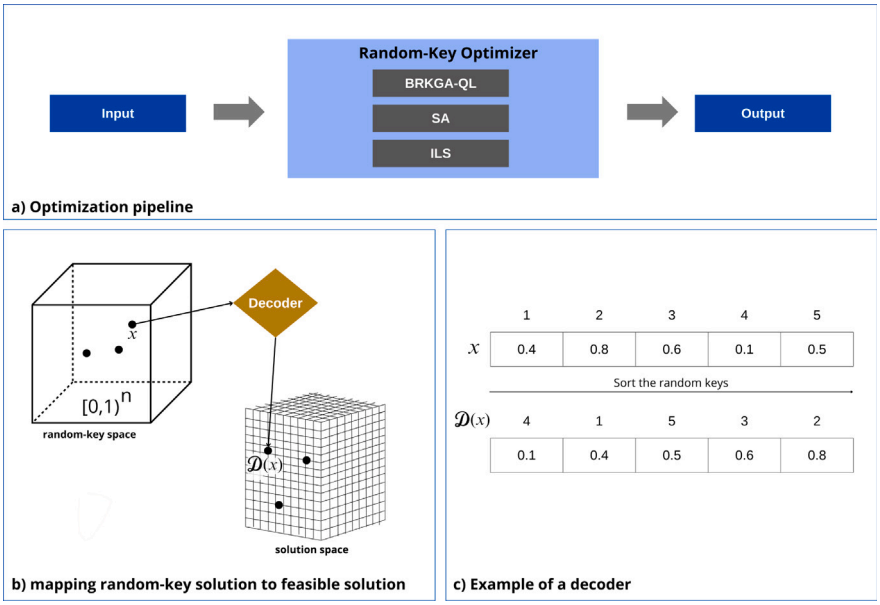


Fig. 3. Random-Key Optimizer concept. Based on [13].

problem solution. Fig. 3 presents the RKO concept where $\mathscr{D}$ represents the specific decoder algorithm.

The optimization process is shown in Fig. 3(a). This process takes as input an instance of a combinatorial optimization problem, runs the search process of each metaheuristic sequentially or parallelly, considering the decoder, and returns the best solution found by the metaheuristics. In this paper, we consider three metaheuristics: BRKGA-QL, SA, and ILS. However, other metaheuristics can be adapted in this framework. Fig. 3(b) illustrates the mapping schema that connects the random-key representation to the solution space through a problem-specific decoder. After this transformation, the quality of the solutions can be evaluated. Finally, Fig. 3(c) presents an example of a decoder process for a scheduling-based problem. In this case, each vector position represents a task or characteristic of the optimization problem. The decoder works by sorting the random keys, where the sorted indices correspond to a potential solution obtained by arranging the tasks in the specific order dictated by the sorted keys.

In the remainder of this section, we introduce the RKO framework. We begin by discussing its key components (Section 4.1), followed by the decoder proposed to the IORSP (Section 4.2). An overview of the metaheuristics utilized in the framework (BRKGA-QL, Simulated Annealing, and Iterated Local Search) are presented in Section 4.3, which operate on the random-key vectors within the proposed RKO framework.

4.1. RKO components

The RKO components are procedures embedded within the framework's random-key space that support the search process's metaheuristics balance between diversification and intensification. These components include shaking, blending, and local search performed by the Randomized Variable Neighbourhood Descent (RVND) method on the random-key vectors. Each component is described in detail in the following subsections, and the pseudocodes are in the Appendix D.

4.1.1. Shaking

The shaking method was inspired by the approach proposed by Andrade et al. [57]. The method modifies random-key values by applying random modifications considering four distinct neighbourhood moves. A perturbation rate β is employed. This value is randomly generated within a specified interval $[\beta_{min}, \beta_{max}]$, which should be defined according to the specific metaheuristic approach being used. The four movements are:

- *Swap*: Swap the positions of two randomly selected random keys i and j.
- *Swap Neighbour*: Swap the position of a randomly selected random key i with its neighbouring key $i+1$.
- *Mirror*: Change the value of a randomly selected random key i with its complementary value.
- *Random*: Assigns a new random value within the interval $[0, 1)$ to a randomly selected random key i.

Algorithm 6 described the shaking procedure. First, a shaking rate β is randomly generated within the interval $[\beta_{min}, \beta_{max}]$, determining the number of perturbations to be applied, specifically $\beta \times n$, where n is the length of the random-key vector A. For each perturbation, a random shaking move is selected from four options: a random move, a mirror move, a swap move, or a swap neighbour move. The selected move is then applied to the vector A. After all perturbations are performed, the modified vector is returned as the output. This vector is then decoded during the metaheuristics search process.

4.1.2. Blending

The blending method extends the uniform crossover (UX) concept proposed by Davis [58] by introducing stochastic elements to create a

new random-key vector. The algorithm combines two solutions, A^a and A^b, to generate a new solution A^c. For each position i in the vector, a random decision is made based on a probability ρ to inherit the corresponding key from either A^a or A^b. The algorithm introduces a parameter $factor$, which modulates the contribution of A^b. Specifically, when $factor = 1$, the original key from A^b is used, and when $factor = -1$, the complement $(1.0 - A_i^b)$ is considered. Additionally, with a small probability μ, the algorithm generates a new random value within the interval $[0, 1)$, further diversifying the resulting vector A^c. The algorithm's pseudocode is presented in Algorithm 7.

The parameter $factor$ is specifically used in our proposed Nelder–Mead heuristic (see Section 4.1.7) to generate solutions in a direction opposite to a 'worse' solution. In the classical Nelder–Mead method, new points are typically generated based on the difference between two solutions. However, applying this approach directly in the random-key space could lead to invalid solutions, such as negative random keys. Additionally, combining random keys without proper adjustments could result in solutions that deviate significantly from the base solutions. Therefore, we introduced the $factor$ parameter within the blending method to address these issues. This modification ensures controlled exploration while maintaining feasibility within the random-key representation.

4.1.3. Randomized Variable Neighbourhood Descent

The Variable Neighbourhood Descent (VND) was proposed by Mladenović and Hansen [59] and extended later for various optimization problems. The VND consists of a finite set of pre-selected neighbourhood structures denoted by N_k for $k = 1, \ldots, k_{max}$, where $N_k(A)$ represents the set of solutions in the kth neighbourhood of a random-key vector A. While standard local search heuristics typically employ a single neighbourhood structure, VND utilizes multiple structures to enhance the search process. Key considerations for applying VND include determining which neighbourhood structures to use and their sequence and selecting an appropriate search strategy for switching between neighbourhoods. Later, [60] proposed the RVND. RVND randomly selects the neighbourhood heuristic order to be applied in each iteration. RVND efficiently explores diverse solution spaces and can be applied to random-key spaces. Users can implement classic heuristics for the specific problem and encode the locally optimal solution into the random-key vector after the search process. Alternatively, users can implement random-key neighbourhoods independent of the specific problem, using the decoder to converge towards better solutions iteratively.

Algorithm 8 displays the RVND pseudo-code. Given an initial solution A, the algorithm begins by initializing a Neighbourhood List (NL). While the NL is not empty, a neighbourhood $\mathcal{N}^i$ is selected randomly from it, and the best neighbour A' within $\mathcal{N}^i$ is identified. If the objective function value $\delta_{D(A')}$ improves upon the current solution $\delta_{D(A)}$, the current solution A is updated to A', and the NL is reset. If no improvement is found, the selected neighbourhood $\mathcal{N}^i$ is removed from the NL. The process repeats until all neighbourhoods have been explored without finding a better solution. The algorithm then returns the best solution found.

Next, we introduce four problem-independent local search heuristics designed to operate within the random-key space. These heuristics employ distinct neighbourhood structures for the RVND algorithm, specifically used to identify the best neighbours. The neighbourhood structures include Swap LS, Mirror LS, Farey LS, and Nelder–Mead LS.

4.1.4. Swap Local Search

The Swap local Search focuses on interchanging two values within the random-key vector. The local search procedure considering this structure is outlined in Algorithm 9. The algorithm begins by defining a vector RK with random order for the random-key indices and initializes the best solution found, A^{best}, to the current solution A. It then iterates over all pairs of indices i and j (with $j > i$) in the random-key vector. For each pair, it swaps the value of the random keys at indices RK_i and RK_j in A. If the resulting solution has a better objective function value than A^{best}, it updates A^{best} to the new solution. If not, it reverts A to the previous best solution. The process continues until all pairs have been considered. The algorithm returns A^{best} as the best random-key vector found in the neighbourhood.

4.1.5. Mirror Local Search

The Mirror Local Search perturbs the random-key values, changing the current value in position j to $(1 - A[j])$. Algorithm 10 illustrates this procedure. Initially, it defines a vector RK with a random order for the random-key indices and sets the best solution found, A^{best}, to the current solution A. The algorithm then iterates over all indices i in the random-key vector A. For each index, it inverts the value of the random key at RK_i. After each inversion, if the new solution has a better objective function value than A^{best}, it updates A^{best} to the new solution. If not, it reverts A to A^{best}. This process continues until all indices have been processed. Finally, the algorithm returns A^{best} as the best random-key vector found in the neighbourhood.

4.1.6. Farey Local Search

The Farey Local Search modifies the value of each random key by randomly selecting values between consecutive terms of the Farey sequence [61]. The Farey sequence of order η consists of all completely reduced fractions between 0 and 1, with denominators less than or equal to η, arranged in increasing order. For our purposes, we use the Farey sequence of order 7:

$$F_7 = \left\{ \frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1} \right\}$$

In each iteration of the heuristic, the random keys are processed in a random order. Algorithm 11 illustrates this procedure. It begins by defining a vector RK with a random order for the random-key indices and initializes the best solution found, A^{best}, to the current solution A. The algorithm then iterates over each index i in the random-key vector. For each index i, it iterates over the Farey sequence F of fractions, setting the value of the random key RK_i in A to a random value uniformly generated between F_j and F_{j+1}, where F_j and F_{j+1} are consecutive fractions in the Farey sequence. After updating the random key RK_i, if the new solution has a better objective function value than A^{best}, it updates A^{best} to the new solution. If not, it reverts A to A^{best}. The algorithm continues this process until all indices have been processed. Finally, the algorithm returns A^{best} as the best random-key vector found in the neighbourhood.

4.1.7. Nelder–Mead Local Search

The Nelder–Mead Local Search, introduced by Nelder and Mead [62], is a numerical technique for finding the minimum of an objective function in a multidimensional space. This direct search approach relies on function comparisons and is commonly used in derivative-free nonlinear optimization. The method starts with at least three solutions and can perform five moves: reflection, expansion, inside contraction, outside contraction, and shrinking. In this study, we always apply the Nelder–Mead Local Search with three solutions: A_1, A_2, and A_3, where one is the current solution derived from the metaheuristic, while the others are randomly chosen from a pool of elite solutions found during the search process. These solutions are ordered by objective function value (A_1 is the best and A_3 is the worst). Fig. 4 illustrates a simplex polyhedron and the five moves.

Algorithm 12 presents the pseudo-code for the Nelder–Mead Local Search adapted for discrete optimization problems. We employ the blending method (Section 4.1.2) to generate new solutions, using $\rho = 0.5$ and $\mu = 0.02$. The algorithm begins with an initial simplex of three solutions (A_1, A_2, A_3). The simplex is sorted based on the objective function values, and the simplex's centroid (A_0) is computed between A_1 and A_2 ($A_0 = \text{Blending}(A_1, A_2, 1)$). The main loop iterates until

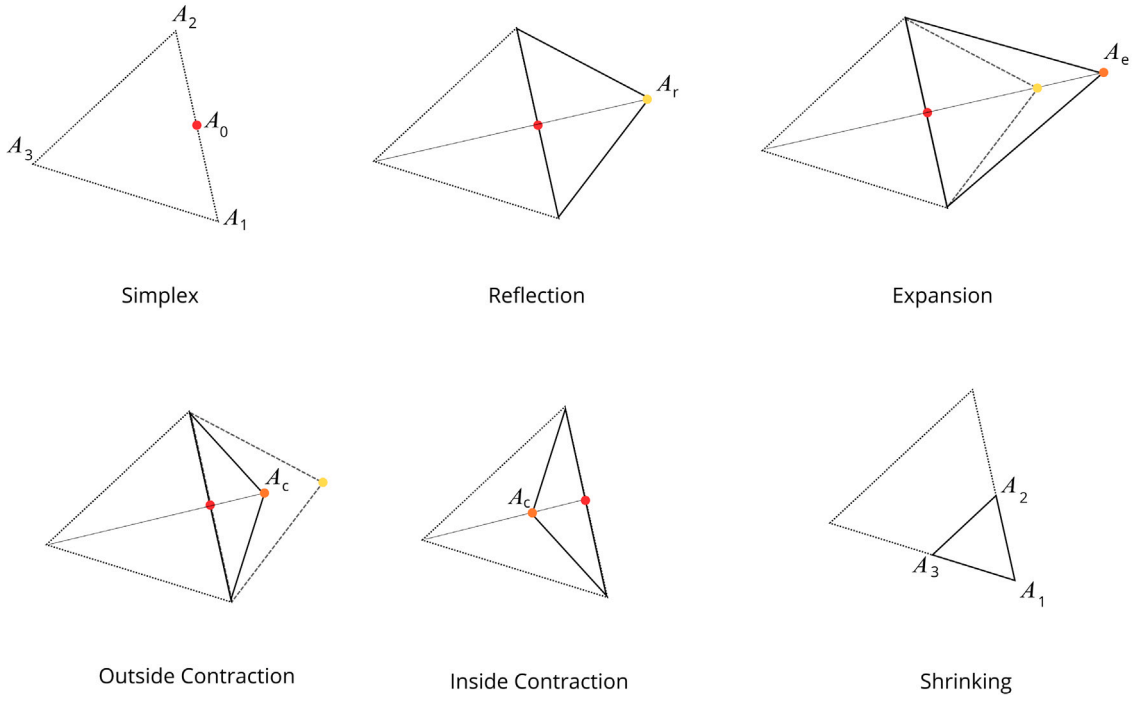


Fig. 4. Illustrative example of the simplex polyhedron and the five moves of the Nelder–Mead Local Search.
Source: Chaves et al. [54].

a termination condition is met. The algorithm performs a series of moves on the simplex during each iteration to explore the search space. A reflection solution (A_r = Blending($A_0, A_3, -1$)) is computed. If the objective function value at A_r is better than the current best solution (A_1), the algorithm computes an expansion solution (A_e = Blending($A_r, A_0, -1$)). If the objective function value at A_e is better than at A_r, A_3 is replaced by A_e; otherwise, A_3 is replaced by A_r. If neither the reflection nor the expansion improves the solution, the algorithm contracts towards the solution A_r or A_3. For an outside contraction (when A_r is better than A_3), the contraction solution is A_c = Blending($A_r, A_0, 1$). For an inside contraction (when A_r is not better than A_3), the contraction solution is A_c = Blending($A_0, A_3, 1$). If the contraction step does not improve, the entire simplex is shrunk towards the best solution A_1 (A_i = Blending($A_1, A_i, 1$), $i = 2, 3$). The algorithm terminates when the maximum number of iterations equals $n \times e^{-2}$.

4.2. Encoding and decoding

Solutions to the IORSP are encoded with a vector A of $n = |K|$ random keys, where K is the set of surgeries. Then, each random key corresponds to a unique surgery. Our decoder implementation follows a straightforward approach: (*i*) sorts the surgeries based on their random-key values, (*ii*) sorts the surgeries based on their priority values, keeping the random-key order for equal priorities; (*iii*) schedules each surgery task sequentially at the earliest available time slots. This implementation always produces feasible schedules given that the planning horizon is not planning horizon constrained while minimizing the makespan since, for any surgery, there will always be a future timeslot in some compatible room with the necessary equipment available. The same approach also always produces feasible solutions to the version of the problem that maximizes the number of scheduled surgeries within a planning horizon.

Table 3 shows the decoder's dynamic structures and operators. We have a solution problem s_A that contains, for each surgery $k \in K$, a surgery allocation data D_k, and each surgery allocation data contains an allocated room r_k, set of equipment e_t, starting time σ_t and blocking time ψ_t. Thus, the scheduled surgeries makespan $\delta_{\mathscr{D}(A)}$ is when the last patient completes its last task.

Algorithm 1 details our implementation. The solution starts with no surgeries scheduled (Line 1) and, consequently, $\delta_{\mathscr{D}(A)} = 0$, then the surgeries are sorted given the sequence of random keys (Line 2). Then, given the final sequence of surgeries, each one is scheduled by our Greed Insertion algorithm (Line 4).

Algorithm 1: Decoder

```
Data: Vector A of random keys
Result: Fitness (makespan) value, δ_{D(A)}
1 s_A ← ∅; δ_{D(A)} ← 0;
2 SortByRK(A);
3 for a ∈ A do
4     GreedInsertion(s_A, K_a, δ_{D(A)});
5 return δ_{D(A)}
```

Algorithm 2 details the Greed Insertion algorithm. It also has the simple concept of scheduling each surgery's task as soon as possible, given all available rooms, types of equipment, and persons. It receives a current problem solution s_A and a surgery k to be scheduled and starts by setting variable σ^m (minimal start time) to zero and *Success* to false (Line 1). The main loop (Lines 2–22), for each task (Line 4), searches all compatible rooms R_t and selects it the one r_t with the earliest moment available σ_c (Line 6), with the assistance of operator *Search* as noted on Table 3. The same logic is applied to select the required equipment types F_t, if any is required (Lines 9–12) and for the required people P_t (Lines 13–15).

Variable σ_t indicates the selected start time for task k. It starts as σ^{min} (Line 5), and it is updated with the maximum value of the selected room (Line 7), each selected equipment (Line 12) and each involved person's schedule (Line 15). The blocking value ϕ_t is computed as the delay (Line 16), if it exceeds the limit (Line 17), the new σ^{min} is set by delaying the first task (Line 18) and restarting k's scheduling (Line 19), otherwise σ^{min} is updated to allocate the following task (Line 21). After a successful surgery allocation, the surgery allocation data D_k is added

Table 3
Decoder notations.

Solution variables	
s_A	Solution problem construct from random-key vector A
D_k	Surgery's k allocation data
r_t	Room for task t
e_t	Set of equipment for task t
σ_t	Starting time for task t
ψ_t	Blocking time for task t
$\delta_{\mathscr{D}(A)}$	Makespan of random-key vector A
Operators	
$Next(t)$	Returns the next task to be executed after t on the same surgery
$First(k)$	Returns the first task to be executed on surgery k
$Last(k)$	Returns the last task to be executed on surgery k
$Search(R_t, k, s_A, \sigma^{min})$	Returns the earliest available resource r and the corresponding candidate moment σ_r^c at which task k can be executed in solution s_A, after moment σ^{min}.

to solution s_A (Line 23) and hence, the makespan $\delta_{\mathscr{D}(A)}$ is updated (Line 24).

Algorithm 2: Greed Insertion

Data: Solution s_A, surgery k, and the current makespan $\delta_{\mathscr{D}(A)}$
Result: A Solution s_A with surgery k scheduled.

1 $\sigma^m \leftarrow 0; Success \leftarrow False;$
2 **while** $\neg Success$ **do**
3 $\quad Success \leftarrow True;\ D_k \leftarrow \emptyset;$
4 $\quad$ **for** *Task* $t \in T_k$ **do**
5 $\quad\quad \sigma_t \leftarrow \sigma^{min};$
6 $\quad\quad (r_t, \sigma^c) \leftarrow Search(R_t, t, s_A, \sigma_t);$
7 $\quad\quad \sigma_t \leftarrow max(\sigma_t, \sigma^c);$
8 $\quad\quad e_t \leftarrow \emptyset;$
9 $\quad\quad$ **for** $f \in F_t$ **do**
10 $\quad\quad\quad (e, \sigma^c) \leftarrow Search(E_f, t, s_A, \sigma_t);$
11 $\quad\quad\quad e_t \leftarrow e_t \bigcup \{e\};$
12 $\quad\quad\quad \sigma_t \leftarrow max(\sigma_t, \sigma^c);$
13 $\quad\quad$ **for** $p \in P_t$ **do**
14 $\quad\quad\quad (p, \sigma^c) \leftarrow Search(\{p\}, t, s_A, \sigma_t);$
15 $\quad\quad\quad \sigma_t \leftarrow max(\sigma_t, \sigma^c);$
16 $\quad\quad \phi_t \leftarrow \sigma_t - \sigma^{min};$
17 $\quad\quad$ **if** $\phi_t > \Phi$ **then**
18 $\quad\quad\quad \sigma^{min} \leftarrow \sigma_{First(k)} + \phi_t;$
19 $\quad\quad\quad Success \leftarrow False;\ Break;$
20 $\quad\quad$ **else**
21 $\quad\quad\quad \sigma^m \leftarrow \sigma_t + \gamma_t^d + \gamma_t^m;$
22 $\quad\quad D_k \leftarrow D_k \bigcup \{(r_t, e_t, \sigma_t, \phi_t\};$
23 $s_A \leftarrow s_A \bigcup \{D_k\};$
24 $\delta_{\mathscr{D}(A)} \leftarrow max(\delta_{\mathscr{D}(A)}, \sigma_{Last(k)} + \gamma^d_{Last(k)});$

Fig. 5 illustrates an example of the Algorithm 1 on a simplified scenario with five surgeries, no priorities, two beds, two ORs, one PSRU, and all subjects with clear initial schedules. Notably, Surgeries 2 and 3 lack PSRU tasks; not all tasks follow the same room sequencing, and all surgeries allocate first and last the same bed as described in Section 1.

The decoder receives a vector of random keys $A = \{0.2, 0.6, 0.1, 0.4, 0.3\}$. Subsequently, each surgery is scheduled in the sequence 3, 1, 5, 4, and 2. Firstly, Surgery 3 involves 3 tasks, with the first task allocating Bed 1 at the first time slot, OR 1 at the second time slot, and the patient returning to Bed 1 at the third time slot. Surgery 1 comprises 4 tasks, with the first allocating Bed 1 at the first time slot, OR 2 at the second and third time slots, the PSRU at the fourth, and the patient returning to Bed 2 at the fifth. Similarly, Surgeries 5, 4, and 2 also have their respective tasks sequentially allocated at the first available moment.

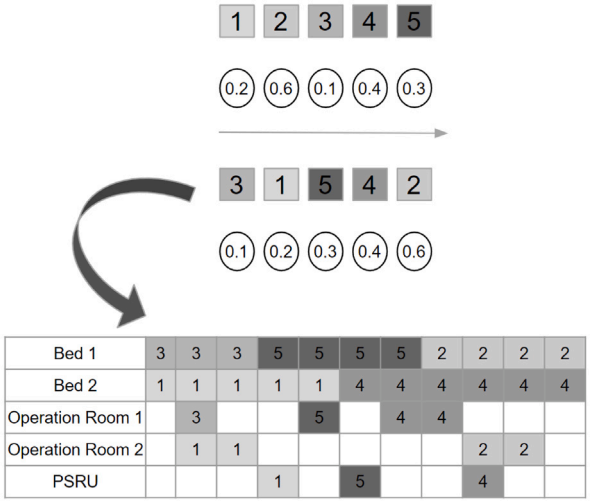


Fig. 5. Example of our decoder with a Greed Insertion strategy.

4.3. Metaheuristics

The components presented are employed in the three metaheuristics developed in this study: BRKGA-QL, SA, and ILS. BRKGA-QL utilizes the blending method as a crossover operator and applies the shaking and RVND methods to intensify the search in promising regions. SA conducts its search using the shaking method and performs local searches with RVND after the temperature cooling process. Finally, ILS bases its search strategy on the shaking and RVND methods. All metaheuristics explore the random-key solution space, and the generated solutions are mapped into feasible IORSP solutions through the decoder method. The following subsections present the methodology of each metaheuristic in detail.

4.3.1. Biased Random-Key Genetic Algorithm with Q-learning

The Biased Random-Key Genetic Algorithm (BRKGA) [45] extends RKGA by introducing bias in selection and crossover. It starts with a randomly generated population P, in which each individual is evaluated based on fitness and classified into elite (P_e) and non-elite individuals ($P - P_e$). A new population is formed by retaining P_e, introducing mutants (P_m), and generating offspring (P_c) via parametrized uniform crossover (PUX) [63] with probability ρ_e guiding gene inheritance. Each individual is then decoded into a solution.

In this paper's BRKGA variant, the new population consists only of P_e and P_c, replacing PUX with a blending crossover (Section 4.1.2) that incorporates mutation (Algorithm 7, lines 2–3). Evolution continues until a stopping criterion is met, with parameters including population size $|P|$, elite proportion p_e, and mutation probability μ. BRKGA-QL [14] differs by replacing mutants with direct mutations, where each gene mutates with probability μ or follows ρ_e for inheritance. It also employs a Q-Learning agent to dynamically adjust parameters (p, p_e, μ, ρ_e) and integrates a local search module (RVND, Section 4.1.3) for solution refinement via clustering and intensification.

Table 4
Possible parameter states.

Parameter	States
$\|P\|$	233, 377, 610, 987, 1597, 2584
p_e	0.10, 0.15, 0.20, 0.25, 0.30
μ	0.01, 0.02, 0.03, 0.04, 0.05
ρ_e	0.55, 0.60, 0.65, 0.70, 0.75, 0.80

The parameter selection process represents an additional optimization challenge, typically addressed through a parameter tuning procedure for later static parameter executions. Such procedure often entails selecting parameter ranges manually or through automation in most existing works [64]. In contrast, the Q-Learning process offers an automated parameter-tuning mechanism that runs parallel to the main problem-solving algorithm.

The Q-Learning algorithm [65] has its domain modelled as a Markov Decision Process [66]. It consists of the learning process of the agent inserted in an environment in terms of actions a, states s, and rewards r. Eq. (1) shows the calculations for updating the weights of state–action pairs. Function Q delineates the value associated with the state–action pair (s, a) at the Q-table $i+1$, symbolizing the quality of actions taken to anticipate future returns or rewards. This update is contingent on two parameters: the learning factor, denoted as λ^{lf}, and the diversification factor, denoted as λ^{df}.

$$Q^{i+1}(s,a) := Q^{i}(s,a) + \lambda^{lf}[R^{i}(s,a) + \lambda^{df} \times max_{a'} Q^{i}(s',a') - Q^{i}(s,a)] \quad (1)$$

Moreover, analogous to metaheuristic common practices, the Q-Learning execution tends to take the best-known actions for each state (intensification) to obtain the expected maximum return. On the other hand, it is also necessary to choose different actions to explore other policies (diversification). The ϵ-Greedy policy consists of a good strategy to balance intensification and diversification, choosing the action with the highest value in Q with probability $1-\epsilon$ or selecting an action at random with probability ϵ. The Q-table represents the learning process with the value of each state–action pair.

In our implementation, at the end of each BRKGA-QL generation j, a reward $R^{i}(s,a)$ is incremented for the corresponding state–action pair (s,a) and Q-table i, as showed by Eq. (2), where δ_{b_j} is the best fitness in the current generation and $\delta_{b_{j-1}}$ the previous one.

$$R^{i}(s,a) \leftarrow R^{i}(s,a) + \begin{cases} \frac{(\delta_{b_{j-1}}/\delta_{b_j})-1}{|P|}, & \text{if } \delta_{b_j} < \delta_{b_{j-1}} \\ 0, & otherwise \end{cases} \quad (2)$$

Furthermore, the reward function serves as an intermediary solution, striking a balance between enhancing the current best fitness and the binary rewards as proposed by Karafotias et al. [67]. Notably, this function exhibits a higher sensitivity to scale than the binary reward, considering the magnitude of improvement and the population size. Table 4 presents the potential states for each parameter, with λ^{df} fixed as 0.8 and λ^{lf} starts with 1 and decreases linearly until 0.1 at the time limit. The Q-table initially has all values set to 0. It undergoes updates every k BRKGA-QL generation using Eq. (1), followed by the reset of corresponding rewards.

Finally, Algorithm 3 presents the summarized implementation of the BRKGA-QL. It is comprised of a usual BRKGA implementation [45] coupled with a local search (Lines 16–19) and the added QL component (Lines 1, 5, 6 and 15).

Algorithm 3: BRKGA-QL

```
   Input: Time limit TL
   Output: Best solution found A^best
1  Step 1: Initialize Q-Table values;
2  Step 2: Randomly generate the population P;
3  Step 3: Evaluate and sort P by fitness. Store the best individual in
     A^best;
4  while TL is not reached do
5      Step 4: Set Q-Learning parameters (ε, lf, df);
6      Step 5: Choose an action for each parameter (p, p_e, μ, ρ_e) from the
         Q-Table using the ε-greedy policy;
7      Step 6: Evolutionary process;
8      Classify P as elite or non-elite individuals;
9      Create elite set P_e using p_e as a guide;
10     Create the offspring set P_c through the blending procedure, using
         ρ_e, μ, and factor = 1 as guides;
11     P ← P_e ∪ P_c;
12     Evaluate and sort P by fitness;
13     if the best individual improved then
14         Store the best individual in A^best;
15         Step 7: Set reward (R^i) and update Q-Table;
16     if exploration or stagnation is detected then
17         Step 8: Local Search;
18         Identify communities in P_e with the clustering method;
19         Apply RVND in the best individuals of these communities;
20         Apply Shaking in other individuals;
21 return A^best;
```

4.3.2. Simulated annealing

Simulated Annealing (SA) is a widely used global optimization method inspired by statistical physics [15,68] and is supported by theoretical guarantees of global convergence [69]. SA starts with an initial solution (denoted as a vector of random keys) $A \in S$, where S represents the random-key solution space. Then, a neighbourhood solution A' is generated through a perturbation algorithm. The search procedure in SA is based on the Metropolis acceptance criterion of Metropolis et al. [70], which models how a thermodynamic system moves from the current solution (state) to a candidate solution in which the objective function (energy content) is being minimized. If $\delta_{\mathcal{D}(A')} - \delta_{\mathcal{D}(A)} \leq 0$, then A' is accepted as the current solution, else the candidate solution is accepted based on the acceptance probability:

$$P(A') = \exp\left(-\frac{\Delta E}{T}\right), \quad (3)$$

where $\Delta E = \delta_{\mathcal{D}(A')} - \delta_{\mathcal{D}(A)}$ and T defines the current temperature. The key idea is to prevent the algorithm from becoming trapped in local optima by allowing uphill moves. It is important to note that uphill moves are more likely to occur at higher temperatures.

Algorithm 4 presents the SA pseudo-code used in this work. The procedure starts by initializing with a solution represented by a random-key vector A. The algorithm runs a loop until a stopping criterion is met. Within this loop, it generates a neighbouring solution (A') from the current solution (A) and calculates the difference in objective function values (energy difference, ΔE). If the new solution has a lower energy ($\Delta E \leq 0$), it is accepted as the current solution and potentially updates the best-found solution. If the new solution has higher energy ($\Delta E > 0$), it may still be accepted based on a probability determined by the Metropolis criterion via Eq. (3), which allows for occasional uphill moves to escape local optima. The temperature (T) is then updated by multiplying it with a cooling rate (α). An RNVD is subsequently called to refine the current solution further. This process continues, gradually reducing the temperature and thus the probability of accepting worse solutions until the stopping criterion is satisfied. The algorithm finally returns the best solution found during the search.

4.3.3. Iterated local search

The Iterated Local Search (ILS) is a straightforward yet powerful heuristic to solve various optimization problems. The foundational idea of ILS was introduced by Baxter [71] and further elaborated by Lourenço et al. [16]. The essence of ILS lies in iteratively constructing a sequence of solutions using a specific heuristic, which typically leads to significantly better solutions than repeated random trials of the same heuristic. This optimization technique attempts to escape

Algorithm 4: Simulated Annealing

Data: Random-key vector A, initial temperature T_0, cooling rate α, β_{min}, β_{max}, SA_{max}, time limit TL
Result: Best found solution

$A^{best} \leftarrow A$, $T \leftarrow T_0$;
while *TL is not reached* **do**
  $iter \leftarrow 0$;
  while $iter < SA_{max}$ **do**
    $A' \leftarrow \text{Shaking}(A, \beta_{min}, \beta_{max})$;
    Calculate the energy difference $\Delta E \leftarrow \delta_{\mathcal{D}(A')} - \delta_{\mathcal{D}(A)}$;
    if $\Delta E \leq 0$ **then**
      $A \leftarrow A'$;
      if $\delta_{\mathcal{D}(A)} < \delta_{\mathcal{D}(A^{best})}$ **then**
        $A^{best} \leftarrow A$;
    else
      Calculate the acceptance probability $P \leftarrow \exp\left(-\frac{\Delta E}{T}\right)$;
      Generate a random number $r \in [0, 1]$;
      if $r < P$ **then**
        $A \leftarrow A'$;
    $iter++$;
  Update temperature $T \leftarrow \alpha \times T$;
  $A \leftarrow \text{RVND}(A)$;
return A^{best}

local optima by applying local search and perturbation in an iterative manner.

The ILS algorithm, as detailed in Algorithm 5, begins with a random-key vector A as input. Initially, the vector A undergoes an RVND process, resulting in the solution set as A^{best}. The algorithm then enters a loop until the stopping criterion is met. A copy of A^{best}, denoted as A', is created and perturbed within the loop. This perturbed solution is then refined using the RVND function. If the objective function value $\delta_{\mathcal{D}(A')}$ of the newly obtained solution A' is better than that of the current best solution A^{best}, represented as $\delta_{\mathcal{D}(A^{best})}$, then A^{best} is updated to A'. Once the stopping criterion is satisfied, the algorithm returns A^{best} as the best-found solution.

Algorithm 5: Iterated Local Search

Data: Random-key vector A, β_{min}, β_{max}, time limit TL
Result: Best found solution

$A \leftarrow \text{RVND}(A)$;
$A^{best} \leftarrow A$;
while *TL is not reached* **do**
  $A' \leftarrow A^{best}$;
  $A' \leftarrow \text{Shaking}(A', \beta_{min}, \beta_{max})$;
  $A' \leftarrow \text{RVND}(A')$;
  if $\delta_{\mathcal{D}(A')} < \delta_{\mathcal{D}(A^{best})}$ **then**
    $A^{best} \leftarrow A'$;
return A^{best};

5. Computational experiments and analysis

Our proposed metaheuristics were coded in C++ and compiled with GCC. The proposed lower bound models were coded on Python and solved with Gurobi 10.01 [72]. The computer used in all experiments was a Dual Xenon Silver 4114 20c/40t 2.2 GHz processor with 96 GB of DDR4 RAM and running CentOS 8.0 x64. Each proposed model was solved for each instance with a time limit of 3600 s, and the metaheuristics were run 30 times for each instance with a time limit proportional to the number of surgeries. The literature instances and results were retrieved from [10] (48 instances in total: 24 with business hours and 24 without), and our case study instances were generated based on gathered data from multiple sources (20 instances with business hours and surgeon and room availabilities and 20 instances without it). This process is detailed in Section 5.4. Table 5 presents the general characteristics of each instance, with its name, number of surgeries and number of rooms per room type in the sequence a patient usually has, and the total CPU time limit (in seconds) of each heuristic run.

We analysed the computational results in terms of quality and robustness. We present the average CPU time and the relative percentile deviation (RPD) for the best-known solution (X_{min}) as shown in Eq. (4). The RPD values are relative deviations from reference ones. We calculated the RPD in each run and presented the average (ARPD) and best (BRPD) values. A statistical analysis is made per set of instances, considering the RPDs of all runs. First, we apply the Friedman test with a significance level of $\alpha = 0.05$ to determine whether the methods have statistically significant differences. This non-parametric test analyzes the rankings of the methods across multiple problem instances and evaluates whether their performances differ beyond random variation. If the null hypothesis, which states that all methods have similar performance, is rejected at the chosen α level, we proceed with the Nemenyi post hoc test followed by the second stage [73] p-value correction method. The Nemenyi test performs pairwise comparisons to identify which specific methods exhibit significant differences in performance coupled with the p-value correction method for a more accurate multi-stage statistical inference acceptance criteria.

$$RPD = (X/X_{min} - 1) \times 100 \tag{4}$$

5.1. Parameter settings

The parameters of the BRKGA-QL are tuned during the search process using the Q-Learning method. However, we also tested a version of the BRKGA without Q-Learning. The parameters of this BRKGA version, SA and ILS are tuned using an offline strategy. We applied an experimental design approach considering potential values for each parameter and identifying the "best" value for each parameter through many experiments on a subset of the problem instances. For each parameter, we consider three possible values, as listed in Table 6, resulting in $3^{\#parameter}$ possible configurations for each method, where $\#parameter$ represents the total number of parameters of the method. The computational experiments were conducted on six instances from each set (case_01, case_06, case_11, case_16, case_20, and case_24).

We ran each parameter configuration and instance five times, and the configuration that found the lowest average RPD was selected as the best parameter set for solving the IORSP with the specific method. The parameters of the ILS were set as $\beta_{min} = 0.10$ and $\beta_{max} = 0.20$. The parameters of the SA were $T_0 = 10^6$, $\alpha = 0.99$, $SA_{max} = 200$ $\beta_{min} = 0.05$, and $\beta_{max} = 0.20$. The parameters of the BRKGA without Q-Learning were $|P| = 1597$, $p_e = 0.20$, $\mu = 0.03$, and $\rho_e = 0.70$.

5.2. Computational results for literature instances

The case study modelling of Burdett and Kozan [10] does not completely match ours. The main difference is that, in their case, the wardroom is the long-time recovery room; in our case, the patient returns to the initial bed for a long recovery. In both cases, one type of room is reserved during each patient's whole stay. This allowed their case to be compatible with Formulation (A.1)–(A.5). For these instances, the distinction between the scenarios with and without business hours lies in the fact that, in the case with it, the ORs are only operational during regular business hours (from 8:00 AM to 5:00 PM from Monday until Friday), and the other rooms are always available, these are implemented with availability slots. Other distinctions include the absence of surgeon or equipment scheduling, compatibility between rooms and surgery types, and customization in initial schedules. These variances were more straightforward to accommodate when we examined a simplified scenario based on our case study. In the following

Table 5
Literature and case study instances details.

Literature				Case study			
Name	#Surgeries	#Rooms	Time limit (s)	Name	#Surgeries	#Rooms	Time limit (s)
case_01.dat	50	5,2,5,10	120	p070	58	31,3,2,4	60
case_02.dat	100	5,2,5,10	200	p078	66	23,3,3,4	70
case_03.dat	150	5,2,5,10	240	p087	75	29,3,3,4	80
case_04.dat	200	5,2,5,10	300	p093	81	30,3,3,4	90
case_05.dat	50	5,3,5,10	120	p098	86	40,3,2,4	100
case_06.dat	100	5,3,5,10	200	p100	84	39,4,3,5	100
case_07.dat	150	5,3,5,10	240	p109	93	35,4,3,5	100
case_08.dat	200	5,3,5,10	300	p120	104	46,4,4,5	100
case_09.dat	50	5,4,5,20	120	p138	118	43,5,4,5	120
case_10.dat	100	5,4,5,20	200	p153	133	71,5,4,7	120
case_11.dat	150	5,4,5,20	240	p165	141	78,6,4,7	130
case_12.dat	200	5,4,5,20	300	p178	154	88,6,4,6	130
case_13.dat	50	5,5,5,20	120	p183	159	89,6,4,6	140
case_14.dat	100	5,5,5,20	200	p189	161	71,7,5,7	140
case_15.dat	150	5,5,5,20	240	p192	164	72,7,4,8	140
case_16.dat	200	5,5,5,20	300	p193	165	78,7,5,7	140
case_17.dat	50	5,10,5,20	120	p197	173	99,6,5,7	160
case_18.dat	100	5,10,5,20	200	p201	177	73,6,4,8	160
case_19.dat	150	5,10,5,20	240	p216	188	95,7,5,9	160
case_20.dat	200	5,10,5,20	300	p233	197	88,9,4,10	160
case_21.dat	50	5,10,5,30	120				
case_22.dat	100	5,10,5,30	200				
case_23.dat	150	5,10,5,30	240				
case_24.dat	200	5,10,5,30	300				

Table 6
Parameter settings configuration.

Method	Parameter	Values
BRKGA	$\|P\|$	{610, 987, 1597}
	p_e	{0.10, 0.15, 0.20}
	μ	{0.01, 0.02, 0.03}
	ρ_e	{0.60, 0.65, 0.70}
SA	SA_{max}	{200, 500, 1000}
	α	{0.95, 0.97, 0.99}
	β_{min}	{0.005, 0.01, 0.05}
	β_{max}	{0.05, 0.10, 0.20}
	T_0	$\{10^4, 10^5, 10^6\}$
ILS	β_{min}	{0.005, 0.05, 0.10}
	β_{max}	{0.10, 0.20, 0.40}

tables, the displayed lower bounds are the best ones computed by any of our relaxed formulations; the formulations and their results are detailed on Appendices A–C.

Table 7 shows our heuristic results with the literature instances without business hours. The old lower bound and Hybrid Simulated Annealing (HSA) results are reported from [10], algorithms were coded on C++ and ran on a personal computer with a 2.6 GHz processor and 16 GB of RAM under Windows 7. The first tree columns describe the instances, followed by the literature lower bound (Old), our best computed lower bound from Table C.10 (New), followed by the HSA and RKO heuristic results, the makespan values for best solution found (best) in days, its best and average relative percentile deviation, respectively BRPD and ARPD, the standard deviations (STD), and the average time to the best solution (ATTB) in seconds. Analysing the results, our relaxed formulations computed lower bound values better than the ones in the literature. Besides, the RKO methods found better results for all literature instances without business hours. The SA found 15 new best-known solutions (BKS), the ILS found 11 new BKS, and the BRKGA-QL found two new BKS. These methods are robust with very small RPDs. Statistical tests indicated that among the RKO heuristics, ILS and SA outperformed BRKGA-QL (ILS vs. BRKGA-QL: Nemenyi test p-values = 0.006 for BRPD and p-values = 0.0007 for ARPD; SA vs. BRKGA-QL: Nemenyi test p-values = 0.015 for BRPD and p-values = 0.005 for ARPD), as shown in Fig. 6(a) and 6(b). No statistically significant difference was found between SA and ILS (p-value = 1.000 for BRPD and ARPD). The ATTB shows us that the BRKGA-QL was the fastest to converge to the best solution found. Looking at the obtained new lower bounds and upper bounds, instance "case_21" has an optimal result proved, and such a solution was found by all three RKO heuristics.

Table 8 shows our heuristic results with the literature instances with business hours. Similar to the previous table, the HSA results are reported from [10] with its solution times in seconds, makespan values for the best solution found (best) in days, its best and average relative percentile deviation, respectively BRPD and ARPD, the standard deviations (STD), and the average time to best (ATTB) in seconds. Upon analysis of the results, we observed that the RKO methods consistently enhanced the results for all instances from the literature. Specifically, SA found six new best-known solutions (BKS), ILS found three new BKS, and BRKGA-QL found 18 new BKS. Statistical tests confirm that the BRKGA-QL heuristics exhibited superior performance compared to SA and ILS, as shown in Fig. 6(c) and 6(d). Specifically, considering the BRPD BRKGA-QL outperformed SA and ILS, BRKGA-QL vs. SA Nemenyi test p-value = 0.024, and BRKGA-QL vs. ILS p-value = 0.037. However, no statistically significant difference was observed between the methods considering the ARPD (BRKGA-QL vs. SA p-value = 0.075, BRKGA-QL vs. ILS p-value = 0.241, and SA vs. ILS p-value = 1.000). SA and ILS also show no statistically significant difference for BRPD (p-value = 1.000). Notably, the disparity between the ARPDs was more pronounced than in Table 7.

Fig. 7 presents the performance profile, as proposed by Dolan and Moré [74], comparing the computational time of the evaluated heuristics: BRKGA-QL, SA, ILS, and HSA, under a 1% optimality gap threshold relative to the best-known solutions (BKS). The results demonstrate the superior performance of BRKGA-QL, which solves approximately 93% of the instances within the shortest time ratio and exhibits a steep curve that quickly plateaus, indicating both efficiency and robustness. In contrast, SA and ILS show similar trends, achieving up to 80% of instances solved but with a more gradual curve, suggesting higher variability in computational performance. Notably, HSA did not solve any instance within the 1% gap threshold, indicating a significant limitation in its ability to achieve high-quality solutions under strict optimality criteria. These results highlight the effectiveness of BRKGA-QL in providing high-quality solutions efficiently across a diverse set of problem instances.

The Time To Target (TTT) plot, as described in [75], is used to analyse the RKO convergence rate considering BRKGA-QL, SA, and ILS

Table 7
Literature instances heuristic results without business hours.

Instance	Lower bounds		HSA			SA					ILS					BRKGA-QL				
	Old	New	best	BRPD (%)	time(s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)
case_01	14.82	16.41	16.80	1.54	173	**16.54**	0.00	0.05	0.00	63	**16.54**	0.00	0.06	0.00	54	16.55	0.04	0.14	0.01	66
case_02	32.24	35.32	36.04	1.73	1822	**35.42**	0.00	0.03	0.01	147	35.43	0.01	0.02	0.00	137	35.43	0.01	0.09	0.01	45
case_03	44.29	49.06	50.10	1.86	6282	49.18	0.01	0.03	0.01	180	**49.17**	0.00	0.02	0.01	166	49.18	0.02	0.08	0.02	49
case_04	60.81	67.05	68.13	1.43	16110	67.17	0.01	0.04	0.01	201	**67.16**	0.00	0.02	0.01	201	67.17	0.01	0.05	0.01	82
case_05	15.37	16.88	17.41	2.70	151	**16.95**	0.00	0.04	0.00	77	16.95	0.00	0.05	0.00	62	16.96	0.05	0.12	0.01	48
case_06	29.89	32.94	34.49	4.47	1606	**33.01**	0.00	0.04	0.00	152	33.01	0.01	0.04	0.00	136	33.02	0.04	0.09	0.01	40
case_07	44.18	48.81	49.45	1.17	6080	48.88	0.01	0.03	0.01	155	**48.87**	0.00	0.02	0.01	155	48.88	0.02	0.05	0.01	46
case_08	61.27	67.50	68.73	1.72	11999	67.57	0.002	0.02	0.01	207	**67.57**	0.00	0.03	0.06	213	67.57	0.00	0.03	0.01	83
case_09	6.79	7.60	8.17	5.82	122	**7.71**	0.00	0.27	0.01	78	7.72	0.13	0.34	0.01	77	7.72	0.13	0.41	0.01	72
case_10	14.99	16.53	17.54	5.13	1397	**16.68**	0.00	0.09	0.01	171	16.68	0.02	0.10	0.01	130	16.68	0.02	0.22	0.02	51
case_11	22.09	24.31	25.95	6.11	5231	24.46	0.03	0.10	0.01	171	**24.45**	0.00	0.07	0.01	167	24.45	0.01	0.12	0.02	79
case_12	30.57	33.60	35.13	4.12	12834	33.73	0.01	0.10	0.02	211	**33.73**	0.00	0.22	0.18	232	33.74	0.05	0.12	0.02	126
case_13	7.34	8.11	9.08	10.47	119	**8.22**	0.00	0.20	0.01	74	8.23	0.12	0.20	0.00	76	8.23	0.15	0.30	0.01	71
case_14	15.41	16.92	18.72	9.76	1318	**17.04**	0.00	0.09	0.01	182	17.04	0.004	0.13	0.01	119	17.05	0.09	0.25	0.02	80
case_15	23.31	25.50	27.11	5.82	4788	25.63	0.05	0.11	0.01	171	**25.61**	0.00	0.16	0.11	157	25.62	0.02	0.14	0.02	71
case_16	32.23	35.27	37.02	4.58	12395	**35.38**	0.00	0.10	0.01	230	35.38	0.01	0.09	0.08	256	35.38	0.01	0.12	0.02	125
case_17	8.23	9.03	9.79	7.66	97	**9.09**	0.00	0.27	0.01	73	**9.09**	0.00	0.31	0.01	69	9.10	0.14	0.40	0.01	75
case_18	15.46	16.99	18.22	6.74	1202	17.07	0.004	0.07	0.01	179	**17.07**	0.00	0.07	0.00	138	17.07	0.01	0.15	0.01	90
case_19	23.91	26.23	27.67	5.17	4450	**26.31**	0.00	0.09	0.01	150	26.31	0.01	0.15	0.14	178	26.31	0.02	0.10	0.01	86
case_20	30.48	33.55	35.29	4.95	11135	**33.62**	0.00	0.08	0.01	216	33.63	0.01	0.06	0.01	245	33.63	0.02	0.09	0.01	135
case_21	5.04	**6.09**	6.54	7.36	119	**6.09**	0.00	0.00	0.00	52	**6.09**	0.00	0.00	0.00	30	**6.09**	0.00	0.00	0.00	30
case_22	10.02	11.05	12.31	9.80	1220	**11.20**	0.00	0.23	0.01	175	11.22	0.14	0.27	0.01	138	11.20	0.02	0.31	0.02	119
case_23	14.18	15.70	17.31	9.27	4356	**15.84**	0.00	0.24	0.01	197	15.85	0.05	0.19	0.01	182	15.84	0.01	0.22	0.02	121
case_24	19.27	21.26	23.02	7.51	10876	21.42	0.11	0.21	0.02	234	21.40	0.03	0.16	0.01	249	**21.40**	0.00	0.19	0.03	160
Averages				5.29			0.01	0.11	0.01			0.02	0.12	0.03			0.04	0.16	0.01	

Table 8
Literature instances heuristic results with business hours.

Instance	HSA			SA					ILS					BRKGA-QL				
	best	BRPD (%)	time (s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)	best	BRPD (%)	ARPD (%)	STD	ATTB (s)
case_01	19.63	12.22	900	**17.49**	0.00	1.26	0.25	82	18.24	4.26	4.59	0.02	48	18.20	4.05	4.57	0.03	60
case_02	42.19	10.19	9432	38.57	0.53	2.52	0.20	167	**38.36**	0.00	2.30	0.39	148	38.49	0.33	2.57	0.44	101
case_03	59.33	10.87	32796	57.29	5.31	6.76	0.48	186	57.35	5.42	7.27	1.22	173	**54.40**	0.00	5.22	1.19	172
case_04	79.46	6.65	110160	78.50	0.05	2.23	0.85	234	79.24	0.99	2.42	1.67	202	**78.46**	0.00	1.53	0.68	207
case_05	20.01	13.48	828	17.63	0.00	0.27	0.06	82	17.66	0.17	3.00	0.22	73	17.68	0.28	2.55	0.23	77
case_06	38.59	11.60	8100	36.43	0.44	1.36	0.17	134	36.48	0.59	1.25	0.07	103	**36.27**	0.00	1.14	0.20	87
case_07	56.85	8.68	32652	53.24	1.76	4.32	1.24	212	53.22	1.72	3.29	0.59	196	**52.32**	0.00	2.75	1.06	151
case_08	68.41	1.25	13572	67.57	0.01	0.02	0.01	203	**67.57**	0.00	0.01	0.00	219	67.57	0.01	0.03	0.01	118
case_09	10.06	16.18	576	**8.66**	0.00	0.33	0.06	68	**8.66**	0.00	1.35	0.07	55	**8.66**	0.00	1.11	0.07	68
case_10	20.52	15.68	6120	18.33	1.58	2.46	0.13	163	18.28	1.32	2.32	0.11	148	**18.05**	0.00	2.42	0.27	106
case_11	32.08	21.55	24480	27.39	3.83	10.63	0.64	172	27.25	3.30	6.66	0.87	192	**26.38**	0.00	3.29	0.96	161
case_12	42.34	13.34	76464	38.42	0.30	4.29	0.67	294	38.34	0.10	2.99	0.51	283	**38.30**	0.00	2.11	0.50	241
case_13	11.65	28.91	504	**9.04**	0.00	2.42	0.10	74	9.23	2.17	4.01	0.05	66	9.26	2.41	3.43	0.06	84
case_14	21.41	16.70	5760	18.38	0.21	1.23	0.27	159	18.36	0.06	0.65	0.16	154	**18.35**	0.00	0.96	0.13	108
case_15	32.74	19.06	24012	28.99	5.31	7.03	0.17	192	28.42	3.23	6.68	0.18	185	**27.53**	0.00	3.82	0.66	215
case_16	44.43	15.57	69624	40.13	1.81	5.60	1.05	273	39.44	0.06	3.00	0.44	286	**39.42**	0.00	2.19	0.38	233
case_17	11.73	20.28	432	**9.85**	0.00	3.30	0.11	88	10.25	4.05	5.05	0.07	60	10.21	3.69	4.24	0.05	74
case_18	20.47	11.87	4716	18.41	0.09	1.03	0.22	155	18.40	0.08	0.45	0.05	139	**18.39**	0.00	0.39	0.06	108
case_19	33.41	18.79	20700	29.36	3.25	4.53	0.27	202	29.36	3.25	3.83	0.24	193	**28.44**	0.00	2.94	0.29	200
case_20	41.87	14.93	59904	37.49	0.17	2.58	0.37	260	37.48	0.15	1.98	0.36	265	**37.43**	0.00	1.59	0.44	222
case_21	7.88	22.99	360	**6.41**	0.00	0.67	0.02	72	6.45	0.75	1.26	0.04	68	**6.41**	0.00	1.21	0.05	63
case_22	15.15	22.80	4500	12.56	1.43	4.31	0.27	160	12.48	0.85	1.85	0.10	169	**12.38**	0.00	2.44	0.24	124
case_23	22.12	26.55	17928	18.31	4.38	5.44	0.19	214	18.31	4.42	4.86	0.09	190	**17.54**	0.00	4.67	0.22	167
case_24	27.36	14.65	56124	24.55	0.52	4.48	0.43	262	24.55	0.49	3.70	0.32	281	**24.43**	0.00	2.81	0.39	243
Averages		15.62			1.29	3.29	0.34			1.56	3.12	0.33			0.45	2.50	0.36	

heuristics. The instance set fjspnostr/case_21 was considered for this test. All three solution procedures find the same solution, which is also the best-known solution for this instance. The experiment consists of running each method 100 times for the instance. Each run is independent and stops when a solution with a cost at least as good as a given target value is found. These experiments consider an integer value at most 0.50% greater than the BKS solution as the target. The analysis of Fig. 8 indicates that within the first 20 s, BRKGA-QL has a 97% probability of reaching the target value, compared to 93% for ILS and 44% for SA. By 40 s, BRKGA-QL consistently achieves the target value, while ILS and SA reach probabilities of 98% and 89%, respectively. The likelihood of reaching the target increases to 100% at approximately 50 s for ILS and 58 s for SA. These results demonstrate that while all three methods exhibit strong convergence, BRKGA-QL achieves the fastest convergence.

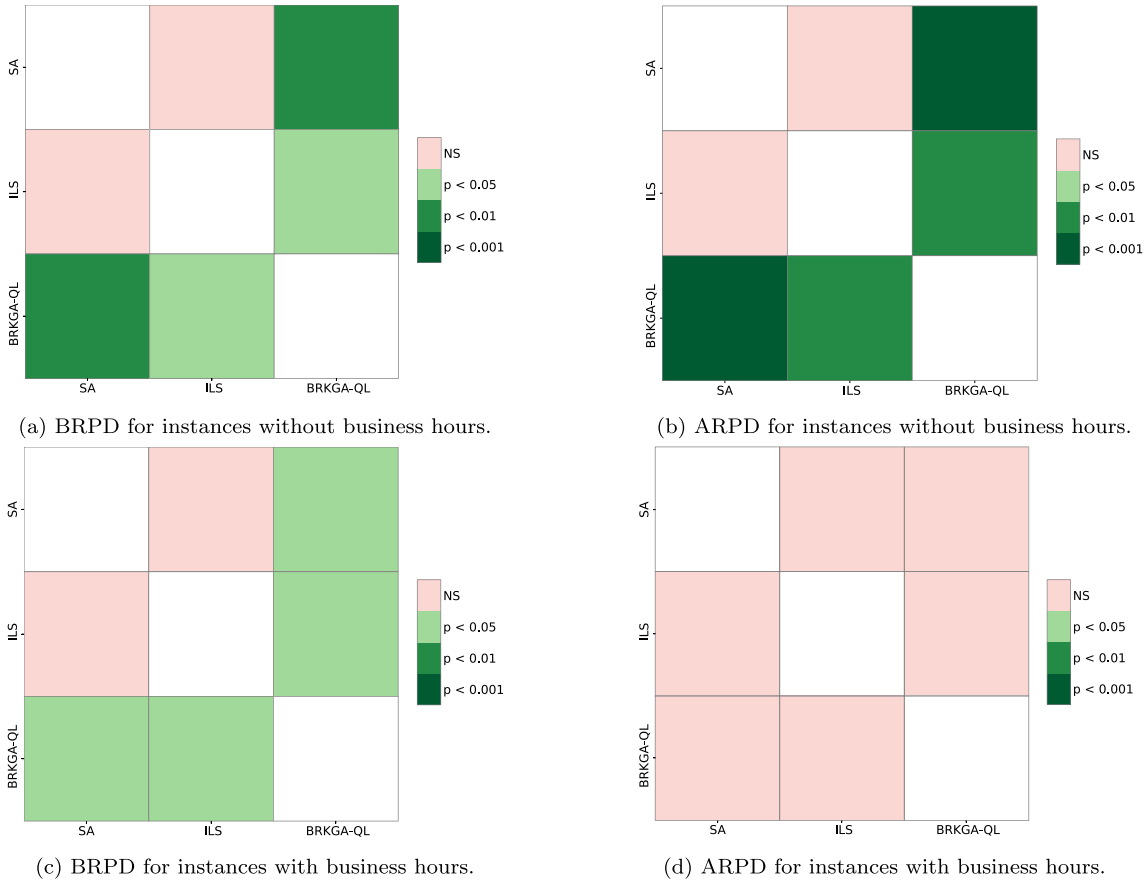
Based on computational tests, no single method outperformed the others in all cases. Therefore, we selected the BRKGA-QL method for the case study, as it provided better results for instances considering business hours. Additionally, this method does not require offline parameter tuning, making it more convenient for day-to-day hospital management. The case study instances are more complex and constrained by factors such as surgeon schedules, operating room types, equipment availability, and other additional attributes.

5.3. Performance comparison of different RKO components

To better understand the contribution of each RKO component, we conducted additional computational experiments analysing the impact of different shaking moves and local search heuristics. In this section, we evaluate the performance of the shaking method when restricted to a single move at a time and the effectiveness of individual local search heuristics compared to the RVND method. These experiments provide insights into the role of each component in the optimization process, helping to justify our design choices and offering guidance for future research in this domain.

5.3.1. Shaking component

Statistical analysis revealed distinct behaviours among the BRKGA-QL, ILS, and SA heuristics when different shaking operators were applied separately. For BRKGA-QL, no statistically significant difference was observed between the shaking methods. This behaviour is expected, as the shaking function in BRKGA-QL is used to perturb



(a) BRPD for instances without business hours.

(b) ARPD for instances without business hours.

(c) BRPD for instances with business hours.

(d) ARPD for instances with business hours.

Fig. 6. Nemenyi test considering the best and average relative percentile deviation (RPD) for the RKO-based heuristics.

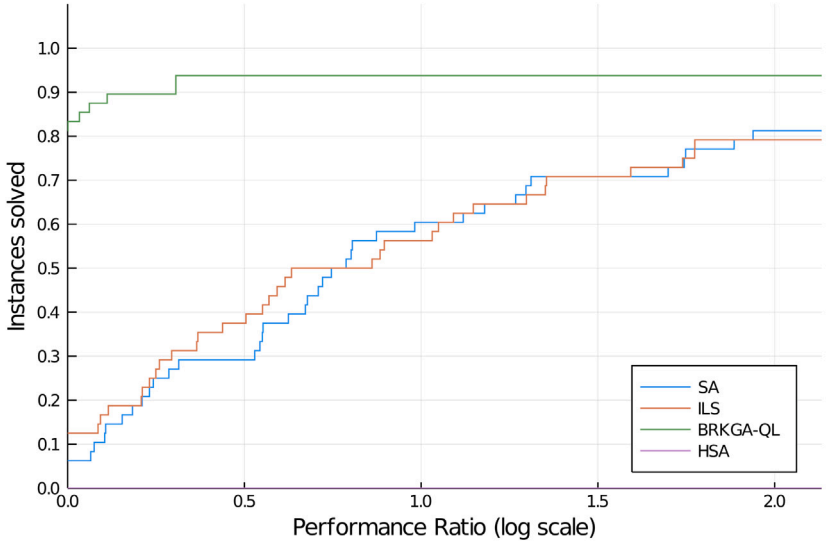


Fig. 7. Performance Profile considering the RKO-based heuristics.

solutions from communities rather than directly guiding the search process. In the case of ILS, only the shaking operator based on the mirror move led to significantly worse results, while the other operators exhibited no statistical difference. Finally, no significant difference was found for SA between using all shaking operators and applying only the random or swap moves. However, SA performed worse when restricted to the mirror and swap neighbour operators, indicating that a more diverse set of perturbations is beneficial for this method. Fig. 9 presents the heatmap with p-values of the Nemenyi Test.

5.3.2. Local search component

Fig. 10 displays the box plots of the RPD performance for each metaheuristic: BRKGA-QL, ILS, and SA, respectively, across the test cases. In each metaheuristic, the version using the RVND (which combines the four local search heuristics) is compared against four simplified variants, each using a single local search heuristic instead of the RVND. In all comparisons, the total CPU time is kept constant. The results indicate that the RVND version outperformed all simplified variants. Among the single-heuristic versions, Swap yielded the best results for

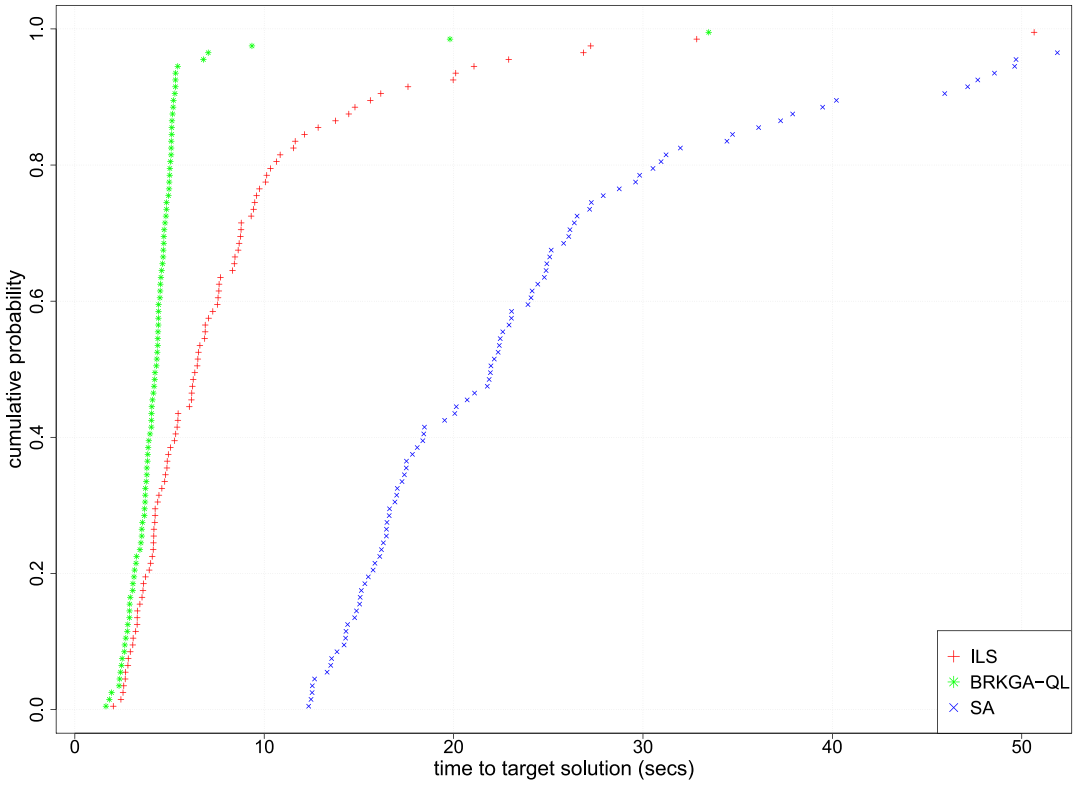
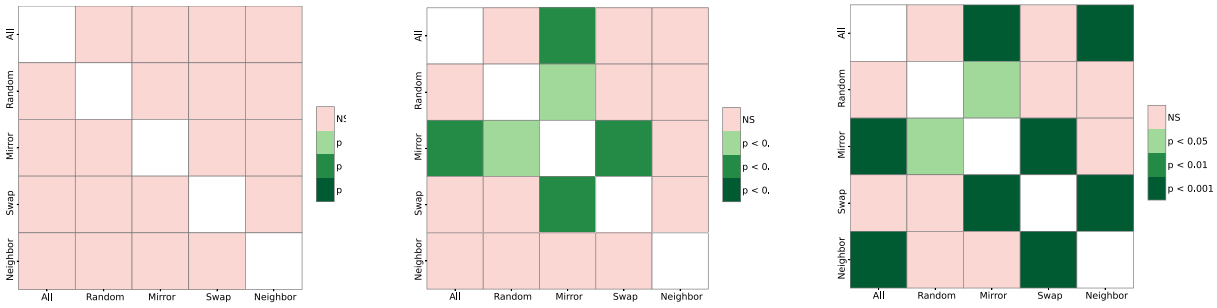
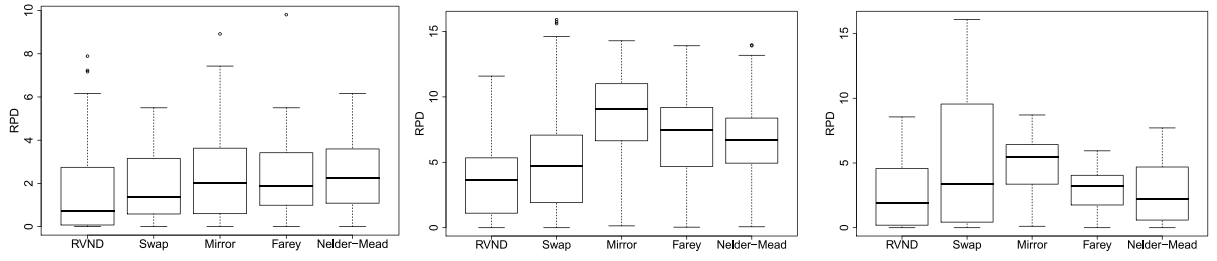


Fig. 8. Time to target plot (Instance fjspnostr/case_21).



(a) BRKGA-QL. (b) ILS. (c) SA.

Fig. 9. Nemenyi test considering the relative percentile deviation (RPD) for the RKO-based heuristics with different Shaking moves (Random, Mirror, Swap, and Swap Neighbour).



(a) BRKGA-QL. (b) ILS. (c) SA.

Fig. 10. Boxplots considering the relative percentile deviation (RPD) for the BRKGA-QL, ILS, and SA with RVND and different local search heuristics (Swap LS, Mirror LS, Farey LS, and Nelder–Mead LS).

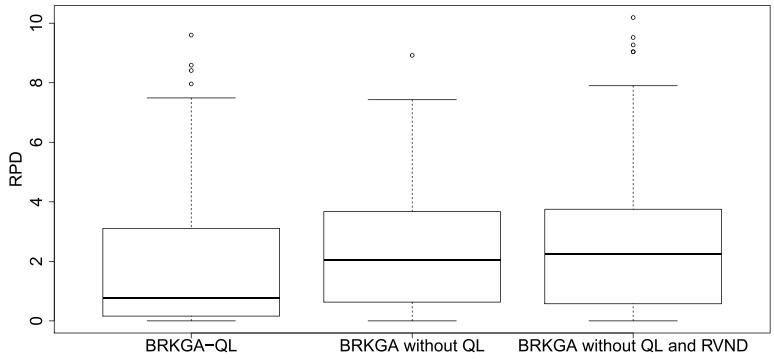


Fig. 11. Boxplots considering the relative percentile deviation (RPD) for the BRKGA-QL, BRKGA without Q-Learning, and BRKGA without Q-Learning and RVND.

BRKGA-QL and ILS, while Nelder–Mead performed best for SA. These findings reinforce that no single heuristic consistently dominates and that combining multiple heuristics within the RVND improves the robustness and overall performance of the metaheuristics.

5.3.3. Q-learning component

Fig. 11 displays the box plots with the different performances of the BRKGA with and without the Q-Learning and without Q-Learning and the RVND local search. As shown in other works [14], it is observable that the Q-Learning component helps it to produce better solutions on average with a smaller spread of results compared to the other versions, reinforced by the Nemenyi test, which shows statistical relevance when comparing superiority of the BRKGA-QL against the BRKGA without QL ($p \ll 0.05$) and against the BRKGA without QL and RVND ($p \ll 0.05$).

5.4. Computational results for the case study instances

Our case study instances were primarily generated using data collected from Santa Casa, a non-profit hospital located in several Brazilian cities. Most of the modelling resulted from several meetings with management staff and interviews with surgeons and nurses that detailed the decision-making process when scheduling surgeries and its expected occurrences. For the production of the test instances, management provided some statistical data with surgery types and operating times. Further data about probability distribution functions for occurrence were sourced from [76,77]. The script initially receives or randomly picks some surgeries, multiplies per a constant, and picks a number inside a ±15% range for the number of surgeons, number of each room type, and available types of equipment. For a 14-day planning horizon, operating rooms and surgeons are only available from 7:00 AM to 10:00 PM. The other room types are always available. Each day, a surgeon is available with a 70% chance. Based on our case study collected data, each operating room is always available from Monday to Friday and from 7:00 AM to 1:00 PM on Saturdays with a 35% chance. The equipment requirement is determined according to the surgery type. The generated set of instances is publicly available at a Github repository.[1]

Table 9 shows our heuristic results with the case study instances. The first three columns describe the instances, followed by the best computed lower bound from Table C.11, followed by the BRKGA-QL results split between the cases with and without availability slots (AS). For these instances, the tests without availability slots imply that all subjects (rooms, surgeons, patients, or equipment) are always available. Each instance has the best and average makespan values in days, as well as the average CPU time in seconds. The case with no availability slots has a percentile optimality gap comparing the best solution to the best lower bound.

[1] https://github.com/brunosalezze/iorsp-instances

Analysing the gap results, eight instances had gaps smaller than 1%, even among the larger ones. This shows that BRKGA-QL found almost optimal results even in instances with many more constraints (surgeon schedules, OR compatibility, and equipment availability) in relation to the instances of Burdett and Kozan [10]. BRKGA-QL demonstrated the ability to identify high-quality solutions within seconds of computation time, even for large-scale instances containing approximately 200 surgical procedures. The algorithm was executed 15 times for each instance, exhibiting robust performance by consistently producing average solutions close to the best solutions obtained. These attributes make BRKGA-QL suitable for practical implementation in hospital management systems, where computational efficiency and solution quality are essential.

5.5. Surgery fixing and rescheduling

Due to unforeseen reasons, a previously scheduled surgical procedure can be postponed or cancelled. This leads to surgery rescheduling, as highlighted by Burdett and Kozan [10]. It is a challenging but necessary aspect of healthcare management. It requires balancing and accommodating patient needs, ensuring patient safety, and efficiently managing resources. While some previously scheduled surgeries can be ultimately rescheduled, often, most of the surgeries closest to execution need to be maintained during the next scheduling. Our approach was developed with initial availability slots for every resource to allow a more flexible input. Some of the reserved initial schedules can also be pre-scheduled surgeries.

As an example of how input can be used for rescheduling efforts, we selected the best-known solution to instance $p70$, shown in Fig. 12, the non-available moments are marked with the colour purple $(171, 99, 250)$, on a 0–255 (red, green, blue) scale. We highlight the schedule of a patient as an example. The patient of colour orange $(255, 161, 90)$ has its events numbered. The patient arrives at the hospital and first goes to Room 12022 – 0 (1), goes to surgery in Room 46048 – 1 (2), leaves to observation in Room 89947 – 3 (3) and moves to recovery back in Room 12022 – 0 (4).

Out of the initial 58 surgeries to be scheduled, we randomly removed 11 to simulate cancelled surgeries, followed by the fixation of programmed schedules of 10 other ones, fixing the rooms, surgeons, and types of equipment reserved for these time intervals. We selected mainly from the first to be performed for the initial scheduling. Finally, we created 7 new surgeries. The BRKGA-QL was run with the new instance under the same conditions as when $p70$ was initially solved with ten runs. The room scheduling for the best solution found can be observed in Figure Fig. 13. The fixed time intervals are the same colour as the non-business hour intervals for all rooms, marked with red $(239, 85, 59)$. The algorithm could fulfil the ORs' time slots as efficiently as possible without pre-scheduled surgeries.

Table 9
Case study instances heuristic results.

Instance	Surgeries	Rooms	Best LB	BRKGA-QL								
				No AS					With AS			
				Best	Average	ARPD (%)	Time (s)	Gap (%)	Best	Average	ARPD(%)	Time (s)
p070	58	31, 3, 2, 4	4.440	4.610	4.632	0.48	60.1	3.85	6.615	6.960	5.22	60.1
p078	66	23, 3, 3, 4	5.536	6.057	6.073	0.26	70.1	9.41	8.959	9.036	0.86	70.4
p087	75	29, 3, 3, 4	5.247	5.679	5.698	0.33	80.1	8.23	9.663	9.678	0.16	80.1
p093	81	30, 3, 3, 4	5.485	5.848	5.870	0.38	90.1	6.62	9.678	9.706	0.29	94.4
p098	86	40, 3, 2, 4	5.719	5.728	5.731	0.05	100.1	0.15	9.648	9.675	0.28	104.6
p100	84	39, 4, 3, 5	4.604	4.981	4.998	0.34	100.1	8.19	8.708	8.740	0.37	104.9
p109	93	35, 4, 3, 5	5.364	5.926	5.942	0.27	100.1	10.47	9.534	9.570	0.38	103.7
p120	104	46, 4, 4, 5	5.466	5.497	5.515	0.33	100.2	0.57	9.823	9.876	0.54	105.0
p138	118	43, 5, 4, 5	5.224	5.776	5.801	0.43	120.2	10.55	9.252	9.315	0.68	126.0
p153	133	71, 5, 4, 7	5.486	5.503	5.515	0.22	120.2	0.3	9.994	10.115	1.21	124.9
p165	141	78, 6, 4, 7	5.019	5.042	5.055	0.26	120.3	0.46	9.233	9.257	0.26	131.3
p178	154	88, 6, 4, 6	5.335	5.353	5.369	0.30	120.3	0.34	9.901	10.026	1.26	128.8
p183	159	89, 6, 4, 6	5.556	5.588	5.596	0.14	140.4	0.56	10.344	10.528	1.78	146.2
p189	161	71, 7, 5, 7	4.902	5.238	5.278	0.76	140.4	6.86	8.847	8.934	0.98	140.3
p192	164	72, 7, 4, 8	4.964	5.257	5.280	0.44	140.4	5.9	9.260	9.354	1.02	141.4
p193	165	78, 7, 5, 7	4.952	5.017	5.034	0.34	140.2	1.3	9.098	9.232	1.47	140.3
p197	173	99, 6, 5, 7	5.767	5.782	5.794	0.21	160.4	0.25	10.263	10.321	0.57	160.9
p201	177	73, 6, 4, 8	6.053	6.081	6.095	0.23	160.4	0.46	10.945	11.085	1.28	160.5
p216	188	95, 7, 5, 9	5.505	5.529	5.547	0.33	160.5	0.44	10.247	10.294	0.46	160.8
p233	197	88, 9, 4, 10	4.778	5.222	5.258	0.69	160.5	9.27	8.980	9.090	1.22	160.0
Averages						0.34		4.20			1.010	

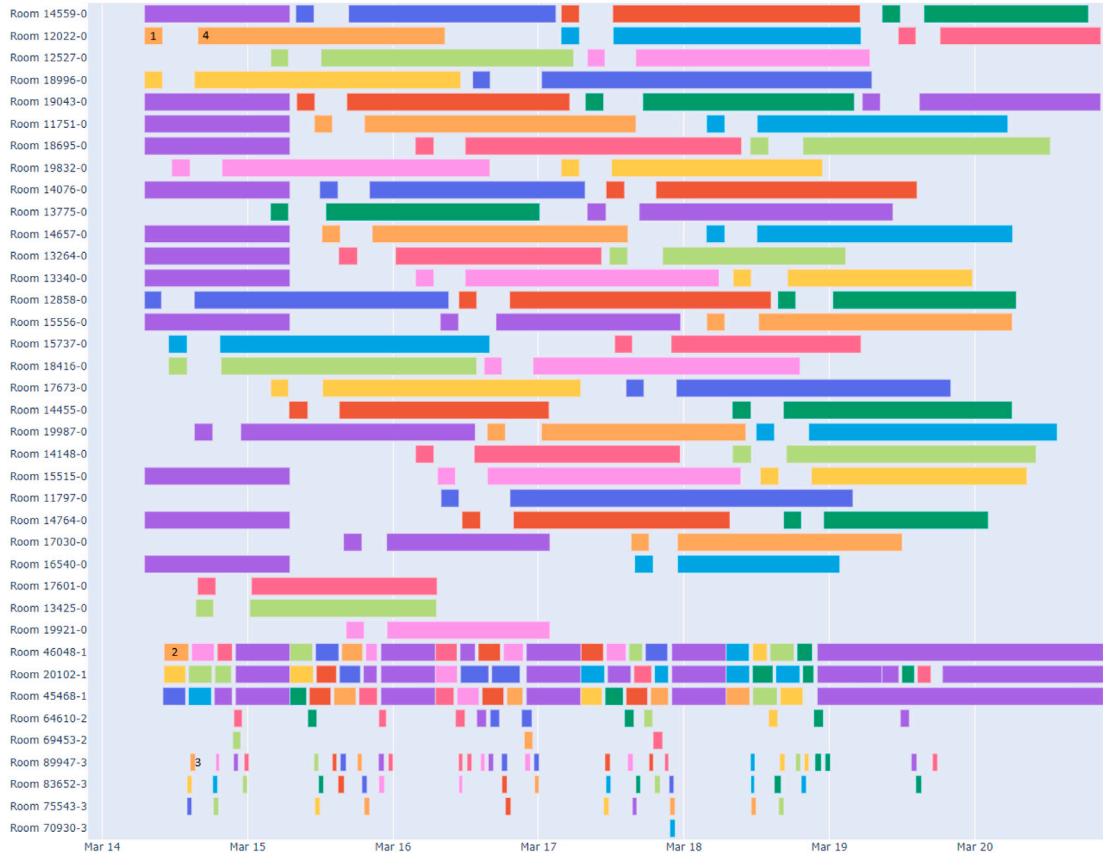


Fig. 12. Rooms schedule of the best-known solution to *p*70.

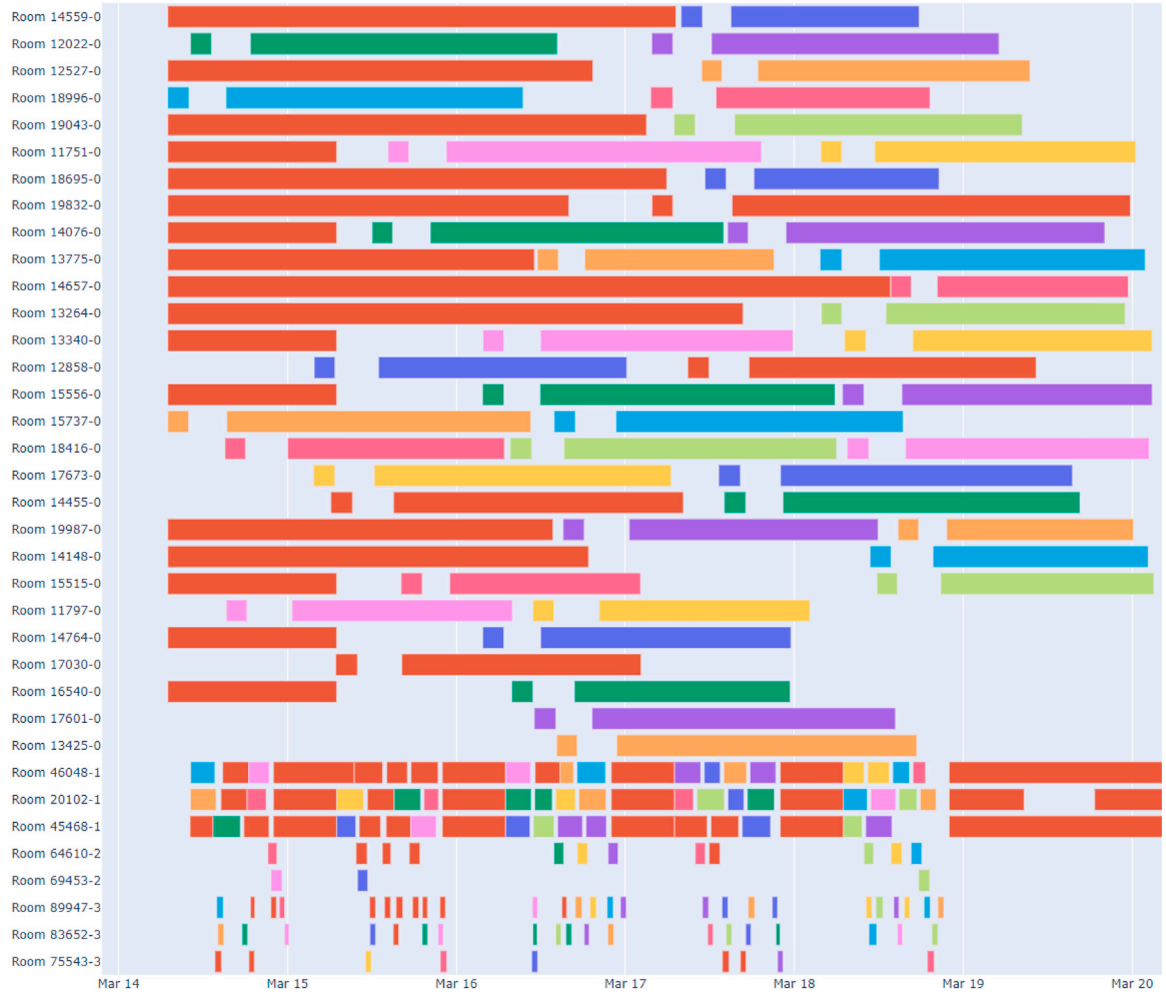


Fig. 13. Rooms schedule of a rescheduled solution to $p70$.

6. Conclusions and future works

This research underscores the critical role of optimized surgery scheduling in enhancing hospital efficiency, improving patient outcomes, and strengthening healthcare system resiliency. By introducing a Biased Random-Key Genetic Algorithm with Q-Learning (BRKGA-QL), Simulated Annealing, and an Iterated Local Search derived from a Random-Key Optimizer (RKO) framework, we have developed an innovative and adaptable approach to solving the complex challenges of surgery scheduling. Our method effectively manages the intricate constraints of multi-room scheduling, equipment availability, and personnel coordination, ensuring operational flexibility and robust rescheduling capabilities in dynamic healthcare environments.

A key contribution of this study is the development of simple yet effective lower-bound formulations, which provide critical benchmarks for assessing heuristic performance. Through rigorous computational testing on both existing literature instances and newly generated real-world datasets, our approach demonstrates significant improvements in both upper and lower-bound quality, surpassing existing methods in terms of accuracy and efficiency. Notably, we prove one optimal result for a literature instance, showcasing the strength of our approach. Among the tested heuristics, BRKGA-QL emerges as the most effective, particularly when considering business-hour constraints, making it a valuable tool for real-world hospital applications.

This study fills several critical gaps in the existing literature. Firstly, our encoding/decoding layer within the RKO framework allows seamless adaptation to different hospital settings, making it applicable across multiple healthcare facilities with varying operational constraints. Second, many scheduling models do not incorporate rescheduling mechanisms, a crucial component for handling unexpected changes such as surgeon unavailability or emergency surgeries. Our approach explicitly integrates flexible scheduling and re-scheduling capabilities, ensuring that hospitals can maintain operational continuity in the face of disruptions.

Furthermore, our work contributes to healthcare resiliency and sustainability as our methodology enables hospitals to handle more surgeries within the same infrastructure, thereby reducing unnecessary costs and operational bottlenecks. This contributes to sustainable hospital management, ensuring that healthcare facilities can adapt to workforce shortages, supply chain disruptions, and increasing patient demands without excessive resource strain.

For future works, we plan to explore a complete MIP model to capture real-world constraints in a better-structured manner. Additionally, we aim to integrate decomposition techniques to reduce optimality gaps further efficiently. Decomposition methods are well-suited for hybrid exact algorithms, improving solution quality and bound tightening. Given that real-world surgery scheduling is highly uncertain – subject to last-minute changes due to emergencies, delays, or surgeon availability – extending our approach to a stochastic or robust optimization framework would enhance its practical applicability. Methodologically, we also intend to develop new versions of Farey Local Search, incorporating this concept into a Variable Neighbourhood Descent framework. Other metaheuristics, such as GA and ABC, can also be adapted using similar components.

CRediT authorship contribution statement

Bruno Salezze Vieira: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Eduardo Machado Silva:** Writing – review & editing, Visualization, Validation, Formal analysis. **Antônio Augusto Chaves:** Writing – review & editing, Validation, Supervision, Software, Methodology, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Bruno Salezze Vieira reports financial support was provided by State of Sao Paulo Research Foundation. Antonio Augusto Chaves reports a relationship with State of Sao Paulo Research Foundation that includes: funding grants. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the São Paulo Research Foundation (FAPESP) under grant #2021/09482-4. Antonio A. Chaves was supported by FAPESP under grants #2018/15417-8, #2022/05803-3 and #2024/08848-3, and the National Council for Scientific and Technological Development (CNPq) under grants #423694/2018-9 and #303736/2018-6. Eduardo M. Silva was supported by the São Paulo Research Foundation (FAPESP) under grant #2023/04588-4.

Appendix A. Model bounded by beds

Formulation (A.1)–(A.5) assumes that beds are the scheduling bottleneck. Our only decision variable x_{rk} is a binary one that indicates whether surgery k is allocated to bed r ($x_{rk} = 1$) or not ($x_{rk} = 0$).

The objective function (A.1) minimizes the best case makespan, Constraints (A.2) calculate the minimal allocated time for each room, where $T_k^T := \sum_{t \in T_k} \gamma_t^d + \gamma_t^m$ is the minimal total time the patient from surgery s takes from arriving at the hospital until discharge (sum of duration γ_t^d and moving γ_t^m times), Constraints (A.3) ensure each surgery is allocated to some room. Constraint (A.4) allocates an arbitrary surgery to a room for symmetry breaking, we display the surgery 0 being allocated to bed 0, and Constraints (A.5) state all x_{rk} variables as binary.

$$Minimize\ z_1 \tag{A.1}$$

Subject to:

$$\sum_{k \in K} T_k^T x_{rk} \leq z_1 \qquad \forall\ r \in R^{bed} \tag{A.2}$$

$$\sum_{r \in R^{bed}} x_{rk} = 1 \qquad \forall\ k \in K \tag{A.3}$$

$$x_{00} = 1 \tag{A.4}$$

$$x_{rk} \in \{0,1\} \qquad \forall\ r \in R^{bed} \quad \forall\ k \in K \tag{A.5}$$

Appendix B. Model bounded by operation rooms

The other proposed solution, Formulation (B.1)–(B.7), also offers a relaxed approach to solving the IORSP. It assumes that ORs are the scheduling bottleneck. Our objective function, (B.1), minimizes the hypothetical makespan. Constraints (B.2) estimate the best possible makespan for each OR, where $T_k^F := \gamma^d_{First(k)} + \gamma^m_{First(k)}$ is the duration and moving times before the OR for surgery k, T_k^S is the surgery added to the moving time after it and T_k^L is the sum of all duration and transfer times after the OR task. Analogous to these constants, we have our binary decision variables, f_{rk} indicating if surgery k is the first one allocated to OR r, x_{rk} indicates if surgery k is allocated to OR r and l_{rk} indicates if surgery k is the last one allocated to OR r.

Constraints (B.3) ensure that all surgeries are allocated to a room, while Constraints (B.4) and (B.5) ensure that each OR has a first and last scheduled surgery. Constraints (B.6) ensure that surgery k can only be the first or last if it is allocated to the respective OR, while Constraints (B.7) define all x_{rk}, f_{rk}, and l_{rk} variables as binary.

$$Minimize\ z_2 \tag{B.1}$$

Subject to:

$$\sum_{k \in K_r} (T_k^F f_{rk} + T_k^S x_{rk} + T_k^L l_{rk}) \leq z_2 \qquad \forall\ r \in R^{or} \tag{B.2}$$

$$\sum_{r \in R_k^{or}} x_{rk} = 1 \qquad \forall\ k \in K \tag{B.3}$$

$$\sum_{k \in K_r} f_{rk} = 1 \qquad \forall\ r \in R^{or} \tag{B.4}$$

$$\sum_{k \in K_r} l_{rk} = 1 \qquad \forall\ r \in R^{or} \tag{B.5}$$

$$x_{rk} \geq f_{rk} + l_{rk} \qquad \forall\ r \in R^{or} \quad \forall\ k \in K_r \tag{B.6}$$

$$x_{rk}, f_{rk}, l_{rk} \in \{0,1\} \qquad \forall\ r \in R^{or} \quad \forall\ k \in K_r \tag{B.7}$$

Appendix C. Lower bound computations

Table C.10 shows the results of our proposed models with the literature instances. For each instance is shown its name, number of surgeries, and number of rooms sorted by room type (for instance "case_10", using our nomenclature, there are 5 beds, 4 ORs, 5 ICUs, and 20 wards) and the computed lower bound and CPU time spent for each model.

Analysing the results, the model bounded by ORs (B.1)–(B.7) is much easier to solve than that bounded by Beds (A.1)–(A.5). In contrast, only seven out of the 24 instances have optimal lower bounds with 3600 s, while all computed lower bounds by the OR model are optimal. Our statistical testing showed the Model Bed to generate superior bounds with p-value $\ll 0.05$. Thus, the instances are more bounded by Bed availability by a large margin.

We can visualize this attribute in Fig. C.14, a heuristic solution of "case_01". Each unique colour represents a patient, and we can observe that all rooms, except for beds, have sparse allocations. A bed is reserved from the moment the patient enters a Ward. This explains the empty spaces between the bed usages; however, they are fully utilized.

Table C.11 shows the results of our proposed models with the case study instances. For each instance, the instances' names, number of surgeries, number of rooms sorted by room type, the computed lower bound in days, and CPU time spent for each model type are presented. Analysing the results, we can observe that 16 instances were more bounded by OR availability and four by Bed availability. The statistical testing shows that the OR formulation computed better bounds with p-value = 0.005. This shows some balance during the instances' procedural generation. As in the literature instances, the model bounded by ORs is much easier to solve than the model bounded by Beds, whereas only nine out of the 20 instances have optimal lower bounds within 3600 s, and all computed lower bounds by the OR model are optimal.

To visualize how a solution with all the availability slots looks like, Fig. C.15 shows the Gantt view of the best solution of instance p098 without availability slots. We can observe that the operating rooms are allocated very close to the limit of their capacity, and the last occupied beds are released almost simultaneously, not as well distributed as in Fig. C.14.

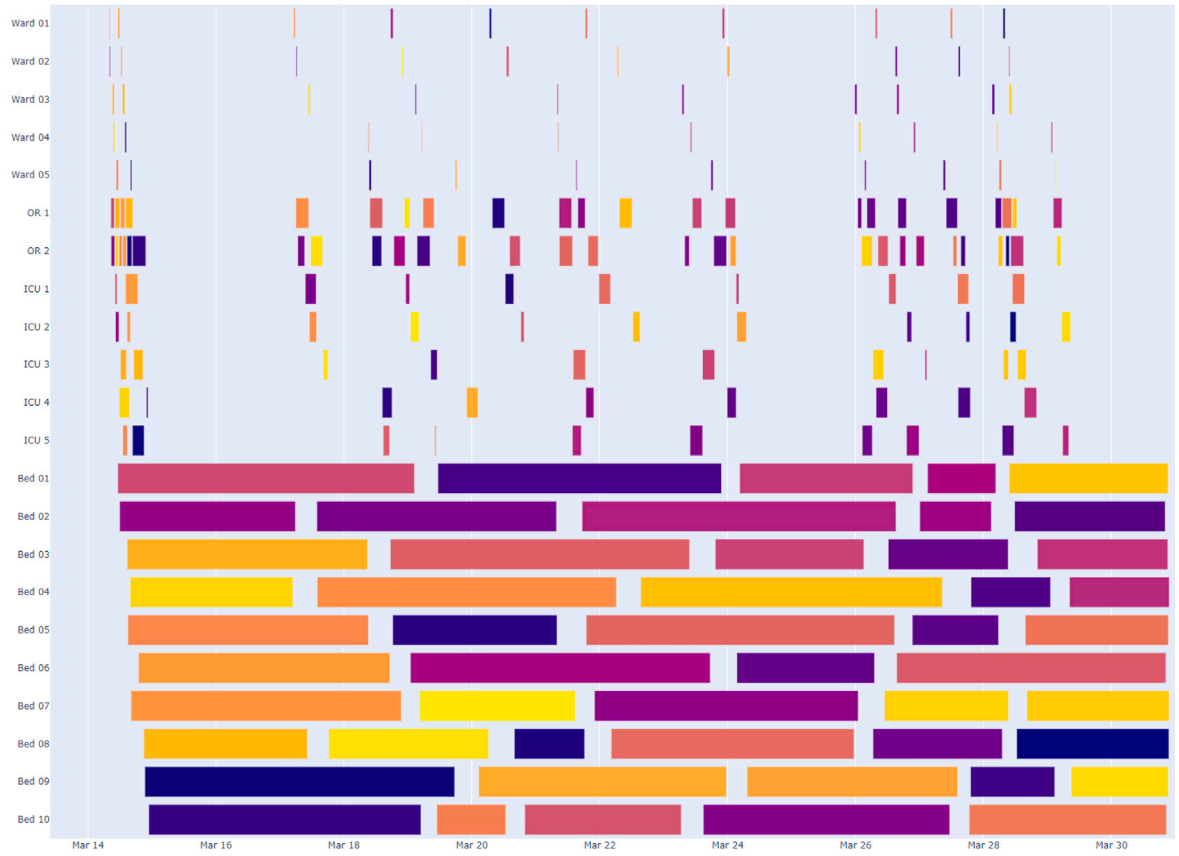


Fig. C.14. Example of a solution found by BRKGA-QL for a literature instance bounded by bed availability.

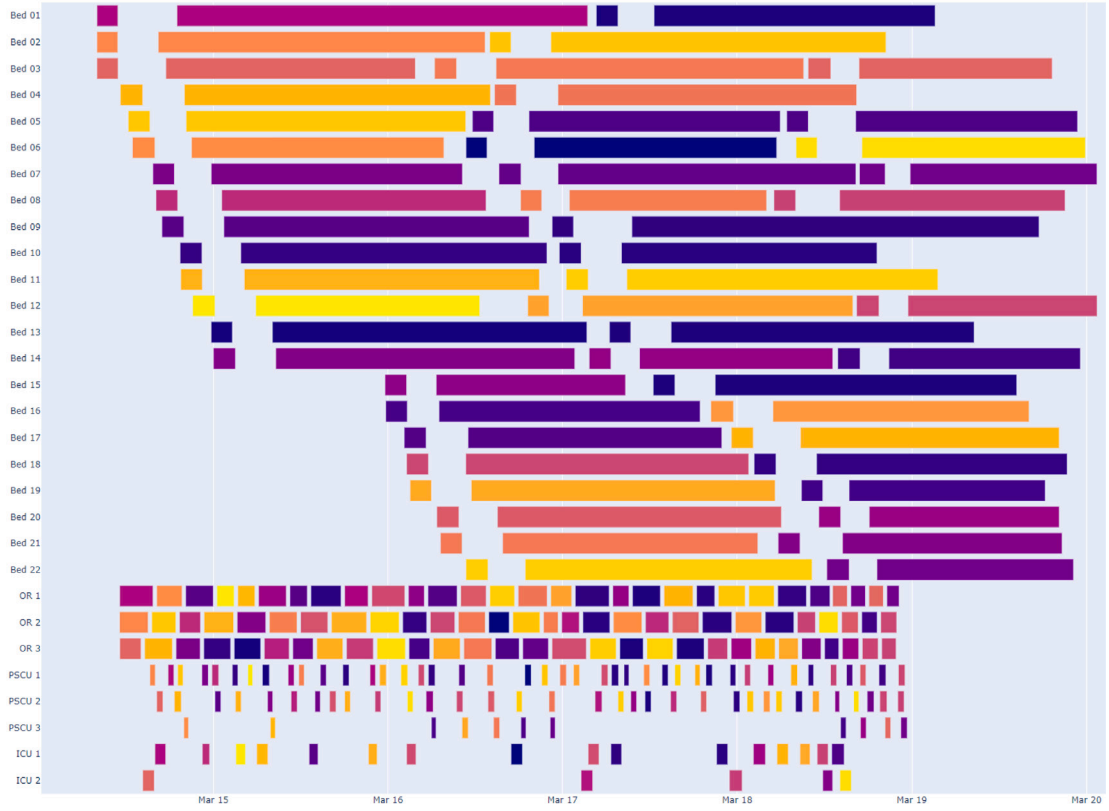


Fig. C.15. Example of a solution found by BRKGA-QL for a case study instance bounded by OR availability.

Table C.10
Literature instances formulation results.

Model	Surgeries	Rooms	Bed		OR	
Instance			Lower bound	Time(s)	Lower bound	Time(s)
case_01	50	5, 2, 5, 10	**16.41**	3600.01	4.57	0.01
case_02	100	5, 2, 5, 10	**35.32**	340.21	7.89	0.01
case_03	150	5, 2, 5, 10	**49.06**	243.07	11.69	0.01
case_04	200	5, 2, 5, 10	**67.05**	355.09	14.93	0.01
case_05	50	5, 3, 5, 10	**16.88**	3600.01	3.72	0.02
case_06	100	5, 3, 5, 10	**32.94**	145.64	5.97	0.02
case_07	150	5, 3, 5, 10	**48.81**	95.60	7.89	0.03
case_08	200	5, 3, 5, 10	**67.50**	281.53	10.09	0.02
case_09	50	5, 4, 5, 20	**7.60**	3600.01	2.85	0.10
case_10	100	5, 4, 5, 20	**16.53**	3600.02	4.64	0.02
case_11	150	5, 4, 5, 20	**24.31**	3600.01	5.92	0.06
case_12	200	5, 4, 5, 20	**33.60**	3600.02	7.73	0.04
case_13	50	5, 5, 5, 20	**8.11**	3600.01	2.67	0.19
case_14	100	5, 5, 5, 20	**16.92**	3600.01	3.91	0.07
case_15	150	5, 5, 5, 20	**25.50**	3600.01	5.10	0.10
case_16	200	5, 5, 5, 20	**35.27**	3600.01	6.66	0.04
case_17	50	5, 10, 5, 20	**9.03**	3600.01	2.44	65.02
case_18	100	5, 10, 5, 20	**16.99**	3600.01	2.70	50.53
case_19	150	5, 10, 5, 20	**26.23**	3600.01	3.29	0.64
case_20	200	5, 10, 5, 20	**33.55**	3600.01	3.96	19.58
case_21	50	5, 10, 5, 30	**6.09**	0.51	2.36	76.05
case_22	100	5, 10, 5, 30	**11.05**	3600.16	2.62	15.84
case_23	150	5, 10, 5, 30	**15.70**	3600.02	3.27	160.01
case_24	200	5, 10, 5, 30	**21.26**	3600.01	3.90	95.01
Averages RPD (%)			0.000		77.321	

Table C.11
Case study instances formulation results.

Model	Surgeries	Rooms	Bed		OR	
Instance			Lower bound	Time (s)	Lower bound	Time (s)
p070	58	31, 3, 2, 4	3.899	0.32	**4.440**	0.01
p078	66	23, 3, 3, 4	**5.536**	3600.02	4.722	0.07
p087	75	29, 3, 3, 4	**5.247**	3600.02	5.146	0.06
p093	81	30, 3, 3, 4	5.253	3600.01	**5.485**	0.07
p098	86	40, 3, 2, 4	4.444	26.32	**5.719**	0.02
p100	84	39, 4, 3, 5	4.486	3600.03	**4.604**	0.14
p109	93	35, 4, 3, 5	**5.364**	3600.01	5.015	0.36
p120	104	46, 4, 4, 5	4.699	3600.03	**5.466**	0.46
p138	118	43, 5, 4, 5	**5.224**	3600.01	4.945	0.04
p153	133	71, 5, 4, 7	3.823	2.32	**5.486**	0.09
p165	141	78, 6, 4, 7	3.895	4.76	**5.019**	5.69
p178	154	88, 6, 4, 6	3.848	2.89	**5.335**	1.09
p183	159	89, 6, 4, 6	3.875	282.51	**5.556**	1.76
p189	161	71, 7, 5, 7	4.849	2912.48	**4.902**	8.45
p192	164	72, 7, 4, 8	4.701	3600.01	**4.964**	3.97
p193	165	78, 7, 5, 7	4.113	3600.02	**4.952**	13.92
p197	173	99, 6, 5, 7	3.829	30.11	**5.767**	3.16
p201	177	73, 6, 4, 8	4.875	3600.03	**6.053**	5.82
p216	188	95, 7, 5, 9	3.842	24.05	**5.505**	3.19
p233	197	88, 9, 4, 10	4.484	3600.08	**4.778**	36.02
Averages RDP(%)			13.944		1.424	

Appendix D. Pseudocode of the RKO components

Algorithm 6: Shaking method

Input: Random-key vector A, β_{min}, β_{max}
Output: Changed random-key vector A

Generate shaking rate β randomly within the interval $[\beta_{min}, \beta_{max}]$;
for $k \leftarrow 1$ ***to*** $\beta \times n$ **do**
  Randomly select one shaking move m from $\{1, 2, 3, 4\}$;
  switch m **do**
    case 1 **do**
      Apply Random move in A;
    end
    case 2 **do**
      Apply Invert move in A;
    end
    case 3 **do**
      Apply Swap move in A;
    end
    case 4 **do**
      Apply Swap Neighbour move in A;
    end
  end
end
return A;

Algorithm 7: Blending method

Input: Random-key vector A^a, Random-key vector A^b, $factor$, ρ, μ
Output: New random-key vector A^c

for $i \leftarrow 1$ ***to*** n **do**
  if $\texttt{UnifRand}(0,1) < \mu$ **then**
    $A_i^c \leftarrow \texttt{UnifRand}(0,1)$;
  end
  else
    if $\texttt{UnifRand}(0,1) < \rho$ **then**
      $A_i^c \leftarrow A_i^a$
    end
    else
      if $factor = 1$ **then**
        $A_i^c \leftarrow A_i^b$
      end
      if $factor = -1$ **then**
        $A_i^c \leftarrow 1.0 - A_i^b$
      end
    end
  end
end
return A^c;

Algorithm 8: RVND

Input: A
Output: The best solution in the neighbourhoods.

Initialize the Neighbourhood List (NL);
while $NL \neq 0$ **do**
  Choose a neighbourhood $\mathcal{N}^i \in NL$ at random;
  Find the best neighbour A' of $A \in \mathcal{N}^i$;
  if $\delta_{D(A')} < \delta_{D(A)}$ **then**
    $A \leftarrow A'$;
    Restart NL;
  else
    Remove $\mathcal{N}^i$ from the NL ;
return A

Algorithm 9: Swap Local Search.

Input: Random-key vector A
Output: Best random-key vector A^{best} found in the neighbourhood

Define a vector RK with random order for the random-key indices;
Update the best solution found $A^{best} \leftarrow A$;
for $i \leftarrow 1$ ***to*** $n-1$ **do**
  for $j \leftarrow i+1$ ***to*** n **do**
    Swap random keys RK_i and RK_j of A;
    if $\delta_{D(A)} < \delta_{D(A^{best})}$ **then**
      $A^{best} \leftarrow A$;
    else
      $A \leftarrow A^{best}$;
return A^{best};

Algorithm 10: Mirror Local Search.

Input: Random-key vector A
Output: Best random-key vector A^{best} found in the neighbourhood

Define a vector RK with random order for the random-key indices;
Update the best solution found $A^{best} \leftarrow A$;
for $i \leftarrow 1$ ***to*** n **do**
  Change the value of the random key RK_i of A to its complement;
  if $\delta_{D(A)} < \delta_{D(A^{best})}$ **then**
    $A^{best} \leftarrow A$;
  else
    $A \leftarrow A^{best}$;
return A^{best};

Algorithm 11: Farey Local Search.

Input: Random-key vector A
Output: Best random-key vector A^{best} found in the neighbourhood

Define a vector RK with random order for the random-key indices;
Update the best solution found $A^{best} \leftarrow A$;
for $i \leftarrow 1$ ***to*** n **do**
  for $j \leftarrow 1$ ***to*** $|F|$ **do**
    Set the value of the random key RK_i of A with $\texttt{UnifRand}(F_j, F_{j+1})$;
    if $\delta_{D(A)} < \delta_{D(A^{best})}$ **then**
      $A^{best} \leftarrow A$;
    else
      $A \leftarrow A^{best}$;
return A^{best};

Algorithm 12: Nelder–Mead Local Search.

Input: A_1, A_2, A_3, n
Output: The best solution found in simplex X

begin
  Initialize simplex: $X \leftarrow \{A_1, A_2, A_3\}$;
  Sort simplex X by objective function value ;
  Compute the simplex centroid $A_0 \leftarrow \text{Blending}(A_1, A_2, 1)$;
  $iter \leftarrow 0$;
  $numIter \leftarrow n \cdot \lfloor 1.0/h \rfloor$;
  while $iter < numIter$ **do**
    $shrink \leftarrow 0$;
    $iter \leftarrow iter + 1$;
    Compute reflection solution $A_r \leftarrow \text{Blending}(A_0, A_3, -1)$;
    if $\delta_{\mathscr{D}(A_r)} < \delta_{\mathscr{D}(A_1)}$ **then**
      Compute expansion solution $A_e \leftarrow \text{Blending}(A_r, A_0, -1)$;
      if $\delta_{\mathscr{D}(A_e)} < \delta_{\mathscr{D}(A_r)}$ **then**
        $A_3 \leftarrow A_e$;
      else
        $A_3 \leftarrow A_r$;
    else
      if $\delta_{\mathscr{D}(A_r)} < \delta_{\mathscr{D}(A_2)}$ **then**
        $A_3 \leftarrow A_r$;
      else
        if $\delta_{\mathscr{D}(A_r)} < \delta_{\mathscr{D}(A_3)}$ **then**
          Compute contraction solution $A_c \leftarrow \text{Blending}(A_r, A_0, 1)$;
          if $\delta_{\mathscr{D}(A_c)} < \delta_{\mathscr{D}(A_r)}$ **then**
            $A_3 \leftarrow A_c$;
          else
            $shrink \leftarrow 1$;
        else
          Compute contraction solution $A_c \leftarrow \text{Blending}(A_0, A_3, 1)$;
          if $\delta_{\mathscr{D}(A_c)} < \delta_{\mathscr{D}(A_3)}$ **then**
            $A_3 \leftarrow A_c$;
          else
            $shrink \leftarrow 1$;
    if $shrink = 1$ **then**
      Replace all solutions except the best A_1 with $A_i \leftarrow \text{Blending}(A_1, A_i, 1)$, $i = 2, 3$;
    Sort simplex X by objective function value ;
    Compute the simplex centroid $A_0 \leftarrow \text{Blending}(A_1, A_2, 1)$;
  return A_1 ;

Data availability

The data is in a public repository disclosed in the paper.

References

[1] M. Al Amin, R. Baldacci, V. Kayvanfar, A comprehensive review on operating room scheduling and optimization, Oper. Res. 25 (1) (2024) 3, http://dx.doi.org/10.1007/s12351-024-00884-z.

[2] E. Aktaş, H.E. Atmaca, H.E. Akbulut, Operating room and surgical team members scheduling: A comprehensive review, J. Proj. Manag. 9 (2024) 149–162, http://dx.doi.org/10.5267/j.jpm.2023.12.001.

[3] M. Mazloumian, M.F. Baki, M. Ahmadi, A robust multiobjective integrated master surgery schedule and surgical case assignment model at a publicly funded hospital, Comput. Ind. Eng. 163 (2022) 107826, http://dx.doi.org/10.1016/j.cie.2021.107826.

[4] M. Lotfi, J. Behnamian, Collaborative scheduling of operating room in hospital network: Multi-objective learning variable neighborhood search, Appl. Soft Comput. 116 (2022) 108233, http://dx.doi.org/10.1016/j.asoc.2021.108233.

[5] B. Akbarzadeh, G. Moslehi, M. Reisi-Nafchi, B. Maenhout, A diving heuristic for planning and scheduling surgical cases in the operating room department with nurse re-rostering, J. Sched. 23 (2) (2020) 265–288, http://dx.doi.org/10.1007/s10951-020-00639-6.

[6] X. Ma, Y. Fu, K. Gao, H. Zhang, J. Mou, A knowledge-based multi-objective evolutionary algorithm for solving home health care routing and scheduling problems with multiple centers, Appl. Soft Comput. 144 (2023) 110491, http://dx.doi.org/10.1016/j.asoc.2023.110491.

[7] R. Azab, A. Eltawil, M. Gheith, A bi-objective stochastic model for operating room scheduling considering surgeons' preferences and collaborative surgeries, Decis. Anal. J. 14 (2025) 100544, http://dx.doi.org/10.1016/j.dajour.2024.100544.

[8] Ş. Gür, M. Pınarbaşı, H.M. Alakaş, T. Eren, Operating room scheduling with surgical team: a new approach with constraint programming and goal programming, Central Eur. J. Oper. Res. 31 (4) (2023) 1061–1085, http://dx.doi.org/10.1007/s10100-022-00835-z.

[9] W. Xiang, J. Yin, G. Lim, An ant colony optimization approach for solving an operating room surgery scheduling problem, Comput. Ind. Eng. 85 (2015) 335–345, http://dx.doi.org/10.1016/j.cie.2015.04.010.

[10] R.L. Burdett, E. Kozan, An integrated approach for scheduling health care activities in a hospital, European J. Oper. Res. 264 (2) (2018) 756–773, http://dx.doi.org/10.1016/j.ejor.2017.06.051.

[11] M. Vali, K. Salimifard, A.H. Gandomi, T.J. Chaussalet, Application of job shop scheduling approach in green patient flow optimization using a hybrid swarm intelligence, Comput. Ind. Eng. 172 (2022) 108603, http://dx.doi.org/10.1016/j.cie.2022.108603.

[12] P. Brucker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing 45 (4) (1990) 369–375, http://dx.doi.org/10.1007/BF02238804.

[13] M.J. Schuetz, J.K. Brubaker, H. Montagu, Y. van Dijk, J. Klepsch, P. Ross, A. Luckow, M.G. Resende, H.G. Katzgraber, Optimization of robot-trajectory planning with nature-inspired and hybrid quantum algorithms, Phys. Rev. Appl. 18 (2022) 054045, http://dx.doi.org/10.1103/PhysRevApplied.18.054045.

[14] A.A. Chaves, L.H.N. Lorena, An adaptive and near parameter-free BRKGA using Q-learning method, in: 2021 IEEE Congress on Evolutionary Computation, CEC, 2021, pp. 2331–2338, http://dx.doi.org/10.1109/CEC45853.2021.95047.

[15] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680, http://dx.doi.org/10.1126/science.220.4598.671.

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

[17] I.A. Chaudhry, A.A. Khan, A research survey: review of flexible job shop scheduling techniques, Int. Trans. Oper. Res. 23 (3) (2016) 551–591, http://dx.doi.org/10.1111/itor.12199.

[18] I. Ozkarahan, Allocation of surgical procedures to operating rooms, J. Med. Syst. 19 (4) (1995) 333–352, http://dx.doi.org/10.1007/BF02257264.

[19] F. Dexter, A. Macario, R.D. Traub, Which Algorithm for Scheduling Add-on Elective Cases Maximizes Operating Room Utilization: Use of Bin Packing Algorithms and Fuzzy Constraints in Operating Room Management, Anesthesiology 91 (5) (1999) http://dx.doi.org/10.1097/00000542-199911000-00043, 1491–1491.

[20] Z. Abdelrasol, N. Harraz, A. Eltawil, Operating room scheduling problems: A survey and a proposed solution framework, in: H.K. Kim, S.-I. Ao, M.A. Amouzegar (Eds.), Transactions on Engineering Technologies, Springer Netherlands, Dordrecht, 2014, pp. 717–731, http://dx.doi.org/10.1007/978-94-017-9115-1_52.

[21] V. Roshanaei, K.E. Booth, D.M. Aleman, D.R. Urbach, J.C. Beck, Branch-and-check methods for multi-level operating room planning and scheduling, Int. J. Prod. Econ. 220 (2020) 107433, http://dx.doi.org/10.1016/j.ijpe.2019.07.006.

[22] M. Yazdi, M. Zandieh, H. Haleh, A mathematical model for scheduling elective surgeries for minimizing the waiting times in emergency surgeries, Int. J. Eng. 33 (3) (2020) 448–458, http://dx.doi.org/10.5829/ije.2020.33.03c.09.

[23] V. Roshanaei, C. Luong, D.M. Aleman, D.R. Urbach, Reformulation, linearization, and decomposition techniques for balanced distributed operating room scheduling, Omega 93 (2020) 102043, http://dx.doi.org/10.1016/j.omega.2019.03.001.

[24] A. Augustin, P. Jouvet, N. Lahrichi, A. Lodi, L.-M. Rousseau, A data-driven approach to include availability of ICU beds in the planning of the operating room, Omega 109 (2022) 102608, http://dx.doi.org/10.1016/j.omega.2022.102608.

[25] H. Fei, C. Chu, N. Meskens, Solving a tactical operating room planning problem by a column-generation-based heuristic procedure with four criteria, Ann. Oper. Res. 166 (1) (2009) 91–108, http://dx.doi.org/10.1007/s10479-008-0413-3.

[26] H. Fei, N. Meskens, C. Chu, A planning and scheduling problem for an operating theatre using an open scheduling strategy, Comput. Ind. Eng. 58 (2) (2010) 221–230, http://dx.doi.org/10.1016/j.cie.2009.02.012, Scheduling in Healthcare and Industrial Systems.

[27] Y. Liu, C. Chu, K. Wang, A new heuristic algorithm for the operating room scheduling problem, Comput. Ind. Eng. 61 (3) (2011) 865–871, http://dx.doi.org/10.1016/j.cie.2011.05.020.

[28] J.M. Molina-Pariente, V. Fernandez-Viagas, J.M. Framinan, Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations, Comput. Ind. Eng. 82 (2015) 8–20, http://dx.doi.org/10.1016/j.cie.2015.01.006.

[29] R. Aringhieri, P. Landa, P. Soriano, E. Tànfani, A. Testi, A two level metaheuristic for the operating room scheduling and assignment problem, Comput. Oper. Res. 54 (2015) 21–34, http://dx.doi.org/10.1016/j.cor.2014.08.014.

[30] P. Landa, R. Aringhieri, P. Soriano, E. Tànfani, A. Testi, A hybrid optimization algorithm for surgeries scheduling, Oper. Res. Heal. Care 8 (2016) 103–114, http://dx.doi.org/10.1016/j.orhc.2016.01.001.

[31] N. Dellaert, J. Jeunet, A variable neighborhood search algorithm for the surgery tactical planning problem, Comput. Oper. Res. 84 (2017) 216–225, http://dx.doi.org/10.1016/j.cor.2016.05.013.

[32] G. Durán, P.A. Rey, P. Wolff, Solving the operating room scheduling problem with prioritized lists of patients, Ann. Oper. Res. 258 (2) (2017) 395–414, http://dx.doi.org/10.1007/s10479-016-2172-x.

[33] S. Zhu, W. Fan, T. Liu, S. Yang, P.M. Pardalos, Dynamic three-stage operating room scheduling considering patient waiting time and surgical overtime costs, J. Comb. Optim. 39 (1) (2020) 185–215, http://dx.doi.org/10.1007/s10878-019-00463-5.

[34] A. Thomas Schneider, J. Theresia van Essen, M. Carlier, E.W. Hans, Scheduling surgery groups considering multiple downstream resources, European J. Oper. Res. 282 (2) (2020) 741–752, http://dx.doi.org/10.1016/j.ejor.2019.09.029.

[35] Y.K. Lin, M.Y. Li, Solving operating room scheduling problem using artificial bee colony algorithm, Healthcare 9 (2) (2021) http://dx.doi.org/10.3390/healthcare9020152.

[36] J. Park, B.I. Kim, M. Eom, B.K. Choi, Operating room scheduling considering surgeons' preferences and cooperative operations, Comput. Ind. Eng. 157 (2021) 107306, http://dx.doi.org/10.1016/j.cie.2021.107306.

[37] C.L. Siqueira, E.F. Arruda, L. Bahiense, G.L. Bahr, G.R. Motta, Long-term integrated surgery room optimization and recovery ward planning, with a case study in the Brazilian National Institute of Traumatology and Orthopedics (INTO), European J. Oper. Res. 264 (3) (2018) 870–883, http://dx.doi.org/10.1016/j.ejor.2016.09.021.

[38] T. Khaniyev, E. Kayis, R. Güllü, Next-day operating room scheduling with uncertain surgery durations: exact analysis and heuristics, European J. Oper. Res. 286 (1) (2020) 49–62, http://dx.doi.org/10.1016/j.ejor.2020.03.002.

[39] B. Addis, G. Carello, E. Tanfani, Evaluating the impact of the level of robustness in operating room scheduling problems, Heal. (Basel) 12 (20) (2024) http://dx.doi.org/10.3390/healthcare12202023.

[40] I. Rahimi, A.H. Gandomi, A comprehensive review and analysis of operating room and surgery scheduling, Arch. Comput. Methods Eng. 28 (3) (2021) 1667–1688, http://dx.doi.org/10.1007/s11831-020-09432-2.

[41] I. Marques, M.E. Captivo, Bicriteria elective surgery scheduling using an evolutionary algorithm, Oper. Res. Heal. Care 7 (2015) 14–26, http://dx.doi.org/10.1016/j.orhc.2015.07.004, ORAHS 2014 - The 40th international conference of the EURO working group on Operational Research Applied to Health Services.

[42] M. Hamid, M.M. Nasiri, F. Werner, F. Sheikhahmadi, M. Zhalechian, Operating room scheduling by considering the decision-making styles of surgical team members: A comprehensive approach, Comput. Oper. Res. 108 (2019) 166–181, http://dx.doi.org/10.1016/j.cor.2019.04.010.

[43] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA J. Comput. 6 (2) (1994) 154–160, http://dx.doi.org/10.1007/s10729-008-9080-9.

[44] J.F. Gonçalves, J.R. De Almeida, A hybrid genetic algorithm for assembly line balancing, J. Heuristics 8 (2002) 629–642, http://dx.doi.org/10.1023/A:1020377910258.

[45] J.F. Gonçalves, M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization, J. Heuristics 17 (5) (2011) 487–525, http://dx.doi.org/10.1007/s10732-010-9143-1.

[46] T.L. Lin, S.J. Horng, T.W. Kao, Y.H. Chen, R.S. Run, R.J. Chen, J.L. Lai, I.H. Kuo, An efficient job-shop scheduling algorithm based on particle swarm optimization, Expert Syst. Appl. 37 (3) (2010) 2629–2636, http://dx.doi.org/10.1016/j.eswa.2009.08.015.

[47] L.A. Bewoor, V. Chandra Prakash, S.U. Sapkal, Evolutionary hybrid particle swarm optimization algorithm for solving NP-hard no-wait flow shop scheduling problems, Algorithms 10 (4) (2017) 121, http://dx.doi.org/10.3390/a10040121.

[48] L.A. Bewoor, V.C. Prakash, S.U. Sapkal, Production scheduling optimization in foundry using hybrid particle swarm optimization algorithm, Procedia Manuf. 22 (2018) 57–64, http://dx.doi.org/10.1016/j.promfg.2018.03.010.

[49] C. Garcia-Santiago, J. Del Ser, C. Upton, F. Quilligan, S. Gil-Lopez, S. Salcedo-Sanz, A random-key encoded harmony search approach for energy-efficient production scheduling with shared resources, Eng. Optim. 47 (11) (2015) 1481–1496, http://dx.doi.org/10.1080/0305215X.2014.971778.

[50] L.S. Pessoa, C.E. Andrade, Heuristics for a flowshop scheduling problem with stepwise job objective function, European J. Oper. Res. 266 (3) (2018) 950–962, http://dx.doi.org/10.1016/j.ejor.2017.10.045.

[51] C.E. Andrade, S.D. Byers, V. Gopalakrishnan, E. Halepovic, D.J. Poole, L.K. Tran, C.T. Volinsky, Scheduling software updates for connected cars with limited availability, Appl. Soft Comput. 82 (2019) 105575, http://dx.doi.org/10.1016/j.asoc.2019.105575.

[52] C.E. Andrade, R.F. Toso, J.F. Gonçalves, M.G. Resende, The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications, European J. Oper. Res. 289 (1) (2021) 17–30, http://dx.doi.org/10.1016/j.ejor.2019.11.037.

[53] A.D. Mangussi, H. Pola, H.G. Macedo, L.A. Julião, M.P.T. Proença, P.R.L. Gianfelice, B.V. Salezze, A.A. Chaves, Meta-heurísticas via chaves aleatórias aplicadas ao problema de localização de hubs em árvore, in: Anais do Simpósio Brasileiro de Pesquisa Operacional, Galoá, São José dos Campos, 2023, p. 25, http://dx.doi.org/10.59254/sbpo-2023-174934.

[54] A.A. Chaves, M.G.C. Resende, R.M.A. Silva, A random-key GRASP for combinatorial optimization, J. Nonlinear Var. Anal. 8 (6) (2024) http://dx.doi.org/10.23952/jnva.8.2024.6.03.

[55] T.F. Noronha, C.C. Ribeiro, Biased random-key genetic algorithms: A tutorial with applications, in: ACM International Conference Proceeding Series, 2024, pp. 110–115, http://dx.doi.org/10.1145/3665065.3665083.

[56] M.A. Londe, L.S. Pessoa, C.E. Andrade, M.G. Resende, Biased random-key genetic algorithms: A review, European J. Oper. Res. 321 (1) (2025) 1–22, http://dx.doi.org/10.1016/j.ejor.2024.03.030.

[57] C.E. Andrade, T. Silva, L.S. Pessoa, Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm, Expert Syst. Appl. 128 (2019) 67–80, http://dx.doi.org/10.1016/j.eswa.2019.03.007.

[58] L.D. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, http://dx.doi.org/10.1016/s0004-3702(98)00016-2.

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

[60] A. Subramanian, L.M. Drummond, C. Bentes, L.S. Ochi, R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery, Comput. Oper. Res. 37 (11) (2010) 1899–1911, http://dx.doi.org/10.1016/j.cor.2009.10.011.

[61] I. Niven, H.S. Zuckerman, H.L. Montgomery, An Introduction to the Theory of Numbers, John Wiley & Sons, 1991, http://dx.doi.org/10.1126/science.90.2329.158.b.

[62] J.A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313, http://dx.doi.org/10.1093/comjnl/7.4.308.

[63] W. Spears, K. De Jong, On the virtues of parametrized uniform crossover, in: ICGA, 1991, pp. 230–236, http://dx.doi.org/10.21236/ADA293985.

[64] N. Dang, L.P. Cáceres, P. De Causmaecker, T. Stützle, Configuring irace using surrogate configuration benchmarks, in: GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 243–250, http://dx.doi.org/10.1145/3071178.3071238.

[65] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3) (1992) 279–292, http://dx.doi.org/10.1007/BF00992698.

[66] R.S. Sutton, A. Barto, Reinforcement Learning, J. Cogn. Neurosci. 11 (1) (1999) 126–134, http://dx.doi.org/10.1162/089892999563184.

[67] G. Karafotias, M. Hoogendoorn, A.E. Eiben, Evaluating reward definitions for parameter control, in: A.M. Mora, G. Squillero (Eds.), Applications of Evolutionary Computation, Springer International Publishing, Cham, 2015, pp. 667–680, http://dx.doi.org/10.1007/978-3-319-16549-3_54.

[68] V. Černý, Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, J. Optim. Theory Appl. 45 (1) (1985) 41–51, http://dx.doi.org/10.1007/BF00940812.

[69] A. Dekkers, E. Aarts, Global optimization and simulated annealing, Math. Program. 50 (1) (1991) 367–393, http://dx.doi.org/10.1007/BF01594945.

[70] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092, http://dx.doi.org/10.1063/1.1699114.

[71] J. Baxter, Local optima avoidance in depot location, J. Oper. Res. Soc. 32 (1981) 815–819, http://dx.doi.org/10.2307/2581397.

[72] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2023.

[73] Y. Benjamini, Y. Hochberg, Controlling the false discovery rate: A practical and powerful approach to multiple testing, J. R. Stat. Soc. Ser. B Stat. Methodol. 57 (1) (1995) 289–300, http://dx.doi.org/10.1111/j.2517-6161.1995.tb02031.x, URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1995.tb02031.x, arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1995.tb02031.x.

[74] E.D. Dolan, J.J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2) (2002) 201–213, http://dx.doi.org/10.1007/s101070100263.

[75] R.M. Aiex, M.G. Resende, C.C. Ribeiro, TTT plots: a perl program to create time-to-target plots, Optim. Lett. 1 (4) (2007) 355–366, http://dx.doi.org/10.1007/s11590-006-0031-4.

[76] A.d.S. Costa, Assessment of operative times of multiple surgical specialties in a public university hospital, Einstein (Sao Paulo) 15 (2017) 200–205, http://dx.doi.org/10.1590/S1679-45082017GS3902.

[77] C.L. Siqueira, E.F. Arruda, L. Bahiense, G.L. Bahr, G.R. Motta, Long-term integrated surgery room optimization and recovery ward planning, with a case study in the Brazilian National Institute of Traumatology and Orthopedics (INTO), European J. Oper. Res. 264 (3) (2018) 870–883, http://dx.doi.org/10.1016/j.ejor.2016.09.021.