A biased random-key genetic algorithm for the home health care problem

Alberto F. Kummer[a,*], Olinto C.B. de Araújo[b], Luciana S. Buriol[c], Mauricio G.C. Resende[d]

[a]*Federal University of Rio Grande do Sul. Porto Alegre, Rio Grande do Sul, Brazil*
[b]*Federal University of Santa Maria. Santa Maria, Rio Grande do Sul, Brazil*
[c]*Amazon.com, Inc., Bellevue, WA, USA and Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil*
[d]*Amazon.com, Inc., Bellevue, WA, USA and University of Washington, Seattle, WA, USA*
E-mail: afkneto@inf.ufrgs.br; olinto@ctism.ufsm.br; buriol@inf.ufrgs.br; mgcr@berkeley.edu

Abstract

Home health care problems consist in scheduling visits to home patients by health professionals while following a series of requirements. This paper studies the Home Health Care Routing and Scheduling Problem, which comprises a multi-attribute vehicle routing problem with soft time windows. Additional route inter-dependency constraints apply for patients requesting multiple visits, either by simultaneous visits or visits with precedence. We apply a mathematical programming solver to obtain lower bounds for the problem. We also propose a biased random-key genetic algorithm, and we study the effects of additional state-of-art components recently proposed in the literature for this genetic algorithm. We perform computational experiment using a publicly available benchmark dataset. Regarding the previous local search-based methods, we find results up to 26.1% better than those of the literature. We find improvements from around 0.4% to 6.36% compared to previous results from a similar genetic algorithm.

Keywords: home health care problem; vehicle routing problem with time-windows; route inter-dependencies; route synchronization; solution space; exploration and exploitation

1. Introduction

The increase in life expectancy directly impacts the demand for health services, thus motivating public authorities to design and implement such services. Home Health Care (HHC) was proposed as an affordable alternative to provide health assistance to the population, leveraging hospital bed availability and the comfort and sanitation of home-assisted patients (Eveborn et al., 2006; Frifita et al., 2017). Home health care prevents moving the elderly from their homes, positively impacting their mental health and

*Corresponding author.

quality of life (Gutiérrez and Vidal, 2014). In most cases, home health care is a less intrusive option than nursing homes (Eveborn et al., 2006). People currently receiving medical treatments can also benefit from this type of service, which reduces hospitalization stress (Landers et al., 2016). It also allows environmental conditions, which is important in scenarios such as the COVID-19 pandemic. Thus, a home health provider can implement a protocol to follow-up low-risk patients at home, which helps to reduce the spread of contaminant pathogens. This way, only medium-to-high risk patients would need to receive total healthcare assistance at hospitals.

According to the literature, the HHC was introduced to increase the access to healthcare services by people living in rural areas (Fernandez et al., 1974). Nowadays, it is the social component that motivates the implementation of public and private initiatives of HHC. For example, there are initiatives focused on enhancing the population's quality of life by providing services like psychology, occupational therapy, and physiotherapy to encourage the physical exercise and maintain the mental well-being of the elderly (Maya Duque et al., 2015). From a management perspective, home care is the most cost-effective among other alternatives (Gutiérrez and Vidal, 2014).

For those reasons, the HHC has attracted more attention in the last two decades, and we see initiatives to implement services in all developed countries worldwide. The counterpart is that HHC practitioners are faced with the complex logistics of routing caregivers around the coverage area while taking care of fulfilling regulations, maintaining work balance, and maximizing the preferences of both nurses and patients. The lack of computational tools to optimize HHC operations often leads to poor solutions, incurring problems like unnecessarily costly routing and poor scheduling of caregiver working times (Grenouilleau et al., 2019).

This paper discusses efficient solution methods for the daily Home Health Care Routing and Scheduling problem (HHCRSP), introduced by Mankowska et al. (2014). The contributions of this work can be summarized as follows.

- Improved lower bounds by simple tweaks of a standard mathematical optimization software;
- A new heuristic-powered decoding strategy for the HHCRSP;
- Analysis of state-of-art intensification components of biased random-key genetic algorithms;
- Study how more convoluted decoding strategies compare against a simpler decoding algorithm.
- New lower and upper bounds for a literature dataset

The remainder of the paper is organized as follows. Section 2 reviews related HHC literature, focusing on the daily HHC problem. Section 3 defines the problem addressed in this research. Section 4 introduces our proposed solution method using a biased random key genetic algorithm. This section also discusses the relevant implementation details of our metaheuristic algorithm. In Section 5, we present the outcomes of the proposed solution method, and we perform a comparison with previous results from the literature. Lastly, we summarize the findings of this study and point to some potential future work in Section 6.

2. Previous work

The HHC problem (HHCP) is not a new research topic in the OR literature, and the first paper dates back to the 1970s with the seminal work of Fernandez et al. (1974). Since then, we have observed an increasing interest in these problems, and their state-of-art is very complex yet robust by considering

several practical constraints. Typically, the core feature set of HHCP consists of a routing problem, a set of skilled/qualified caregivers, and a set of patients spread over a geographical region. In the VRP literature, this is often referred to as a multi-attribute Vehicle Routing Problem (Grenouilleau et al., 2019). Most authors consider a single-depot problem (Cissé et al., 2017; Grieco et al., 2020), and the Euclidean space is frequently used to model travel times between locations. Recent surveys indicate the prevalence of single transportation modes (Fikar and Hirsch, 2017; Grieco et al., 2020). Patients usually have hard time windows (Bredström and Rönnqvist, 2008; Rasmussen et al., 2012), but some authors allow some flexibility via soft time windows (Mankowska et al., 2014; Decerle et al., 2018). Any combination of these features configures a complex vehicle routing problem. Furthermore, this is not an exhaustive list of characteristics from the HHC literature, and there is no consensus of what configures a standard problem (Fikar and Hirsch, 2017; Grenouilleau et al., 2019; Grieco et al., 2020).

To better define the scope of our research, we focus our study on the variant of the daily home health care problem introduced by Mankowska et al. (2014). This problem considers multiple service types, caregiver qualification levels, soft time windows, and route inter-dependency constraints for patients requesting multiple visits. Bredström and Rönnqvist (2008) approached a similar problem with hard time windows, and they also model caregiver preferences through additional soft constraints. Both publications applied mathematical programming models, solving only small instances to optimality. In both cases, the most promising algorithms were heuristics. Mankowska et al. (2014) solved instances with up to 300 patients and 40 caregivers, and Bredström and Rönnqvist (2008) solved instances with up to 80 patients and 16 caregivers.

Lasfargeas et al. (2019) proposed stochastic local-search operators to solve HHCRSP of Mankowska et al. (2014). Similar to the literature, the authors used a matrix-based structure to represent solutions and facilitate the implementation of the local search operators, especially the ones for optimizing multiple visits patients. The authors tested their algorithm over the benchmark dataset proposed by Mankowska et al. (2014). According to their results, the stochastic approach benefits the local search technique, leading to improved results and shorter computational times compared to the deterministic algorithms proposed by Mankowska et al. (2014). Despite that, the authors only tested instances up to 75 patients, supposedly due to feasibility issues on larger test cases.

Rasmussen et al. (2012) approached a very similar problem to that of Mankowska et al. (2014) but considered hard time windows for the patient visits. They modeled the patient assignment as soft constraints to circumvent feasibility issues, allowing a solution to have unscheduled visits. The authors also generalized the route inter-dependency requirements for multiple visit patients, and they propose a master formulation using such generalized constraints (Desaulniers et al., 2006). They applied a branch-and-price algorithm, but the generalized constraints were implicitly enforced during the branching phase, so there is no need to develop a specialized algorithm for the pricing problem. Proposed algorithms were tested over the dataset of Bredström and Rönnqvist (2008). The authors also devised new random instances, partially based on a real HHC problem arriving in Denmark. They solved instances of 15 caregivers and 150 patients exactly with a branch-and-price algorithm, exploiting the problem structure with tailored branching rules from routing problems with transshipment (Drexl, 2012).

A few other authors further tested the dataset of Mankowska et al. (2014). Kummer et al. (2019) provide new solutions via a fix-and-optimize *matheuristic* that, iteratively, efficiently re-optimizes small portions of the problem. They applied the constructive heuristic of Mankowska et al. (2014) to generate initial solutions, and the *matheuristic* runs until achieving a certain number of non-improving iterations.

The authors were capable of improving the best-known solutions to instances of up to 100 patients in a competitive computational time compared to Mankowska et al. (2014) and Lasfargeas et al. (2019). Their fix-and-optimize demonstrated to be inapplicable in instances with more than 100 patients, mainly due to high memory consumption by the solver.

Kummer et al. (2020) were the first to propose a biased random-key genetic algorithm (BRKGA) (Gonçalves and Resende, 2011) for solving an HHC problem. Their decoder employed a greedy best-insertion heuristic that selects the best caregiver(s) to fulfill patient demands, and the patients were processed according to the permutation derived from the chromosome. Their metaheuristic algorithm was tested on the instances from Mankowska et al. (2014). According to the authors, the exploit-and-explore capabilities of their BRKGA performed consistently better than the local search operators from the literature, mainly in medium to large-sized instances. Due to the greediness of the decoder used, the metaheuristic algorithm could not achieve an optimal solution for some small instances. Despite that, the proposed algorithm outperformed almost all previous literature results for instances with 25 patients and five caregivers up to the largest test cases of the dataset, comprising 300 patients and 40 caregivers.

Considering that there is still no agreement in the literature about a standard home health care problem, we summarize in Table 1 the closely related literature to the single-day HHC problem we study. For a more comprehensive overview of the literature, please c.f. Cissé et al. (2017), Fikar and Hirsch (2017), and Grieco et al. (2020).

Table 1: Summary of publications regarding the HHCP.

Work	M	PH	RM	T	WT	PR	PTW	CTW	WB	WT	OT	BR	SK	CP	SY	SM	Largest inst. solved		
																	#C	#P	#J
Fernandez et al. (1974)	–	S	TD	H	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Bredström and Rönnqvist (2008)†	x	S	TT	H	–	S	H	H	S	–	–	–	–	–	S, P	MIP+RH	16	80	80
Rasmussen et al. (2012)†	x	S	TC	S	–	S	H	H	–	–	–	–	H	–	S, P	B&P	16	150	150
Mankowska et al. (2014)*	x	S	TT	H	–	–	S	–	–	–	–	–	H	–	S, P	MIP, AVNS	40	300	390
Lasfargeas et al. (2019)*	x	M	TT	H	–	–	S	H	–	–	–	–	H	H	S, P	VNS	15	75	98
Decerle et al. (2018)†	x	S	TD	H	–	–	S	H	–	–	–	–	H	–	S	MA	6	109	109
Kummer et al. (2019)*	x	S	TT	H	–	–	S	–	–	–	–	–	H	–	S, P	F&O	20	100	130
Kummer et al. (2020)*	x	S	TT	H	–	–	S	–	–	–	–	–	H	–	S, P	GA	40	300	390

M: Proposes a model to the problem; **PH:** Length of the planning horizon (S: short/single day, M: medium/up to one week, L: long/several months); **RM:** Routing metric used to evaluate the solution cost of the routing (TD: travel distance, TC: travel cost, TT: travel time); **T:** Tasks must be fulfilled (H: hard) or may not (S: soft, with penalization in the objective function); **WT:** Constraints to model waiting times; **PR:** Preference of assignments between caregivers and patients; **PTW:** Patient time-windows; **CTW:** Caregiver time-windows; **WB:** Working balance constraints; **WT:** Constraints to model work time regulations; **OT:** Constraints to model overtime; **BR:** Breaks/rest time constraints; **SK:** Presence of skills/qualification levels for the caregivers; **SY:** Route inter-dependency constraints (S: simultaneous attendance, P: precedence constraints); **SM:** Best-performing solution method (LP+RH: linear programming plus rounding heuristic, RMH: repeated matching heuristic; PSO: particle-swarm optimization, MIP+RH: MIP with rounding heuristic, VNS: variable neighborhood search, B&P: branch-and-price solver, AVNS: adaptive VNS, MA: memetic algorithm, F&O: fix-and-optimize matheuristic, GA: genetic algorithm); **#C:** Number of caregivers; **#P:** Number of patients; **#J:** Number of jobs. Works that use the benchmark dataset proposed in Bredström and Rönnqvist (2008)†, and Mankowska et al. (2014)*, respectively .

3. The home health care problem

Mankowska et al. (2014) proposed a single-day problem called the home health care routing and scheduling problem (HHCRSP). We use the terms HHCRSP and HHCP interchangeably in this section and the following. Let $\mathcal{V}$ be the set of the caregivers/vehicles, and $\mathcal{C}$ the set of patients/nodes. This routing component aims to assign caregivers to the patients, seeking to minimize each caregiver's travel time. Let $\mathcal{C}^0 = \{0\} \cup \mathcal{C}$, where 0 represents the depot. Parameter d_{ij} defines the travel time between the tuples $(i, j) \in \mathcal{C}^0 \times \mathcal{C}^0$. All caregivers travel at the same time–i.e. the problem considers a single transportation modality. The patients also have time-windows. More precisely, each patient $i \in \mathcal{C}$ has a hard time-window starting e_i, meaning that a visit can only happen after the start of the time-window. Conversely, patient i also has a soft time-window ending l_i, and such services should be served ideally before the end of the patient time-window. If such a requirement is not fulfilled, then attendance is considered tardy, and a penalty is incurred in the objective function. In the next section, we introduced service types, a characteristic of this HHCP that sets routing components as multi-attribute VRPTW (Grenouilleau et al., 2019).

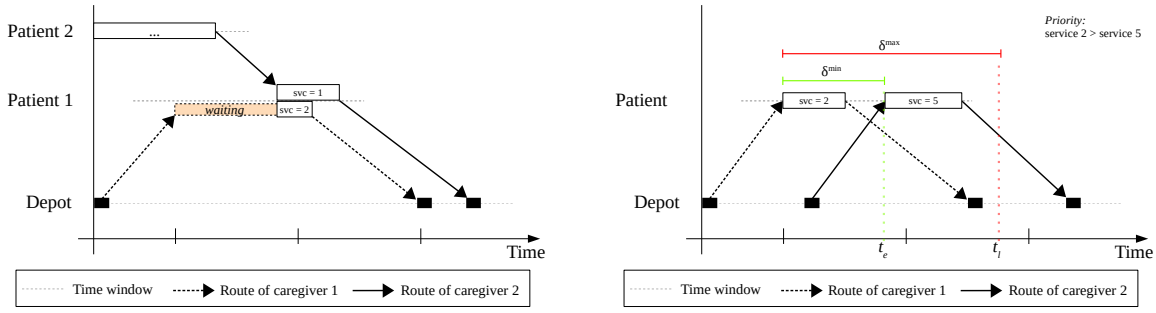
3.1. Service types, qualification, and job requirements

In the HHCRSP, a patient may request up to two distinct *service types*, which skilled caregivers must fulfill. This feature is inherent to the HHCP, requiring multiple visits to some patients. Let $\mathcal{S}$ be the set of service types considered in the problem. Caregivers are skilled in the sense that each caregiver has a subset of service types that they can perform. For each $v \in \mathcal{V}$ and service type $s \in \mathcal{S}$, the parameter $a_{vs} = 1$ indicates if caregiver v can perform service type s, and is otherwise 0. Patients have a similar parameter setting. The parameter $r_{is} = 1$ indicates if patient $i \in \mathcal{C}$ requires attendance for service type $s \in \mathcal{S}$, and is otherwise 0. Additionally, the processing time required to complete such attendance is defined according to the parameter $p_{is} > 0$.

In general, the problems are composed mostly of *single service* patients, requiring a visit by a single skilled caregiver. Patients requesting two distinct service types are known as *double service* patients. We introduce the set $\mathcal{C}^{\mathrm{d}} \subseteq \mathcal{C}$ to refer to the subset of double service patients. In double services, two visits are required by distinct (and skilled) caregivers to fulfill each patient's demands, and additional constraints are imposed when performing such visits.

3.2. Synchronization constraints for double service patients

As aforementioned, some patients require more than one service type. For those, two new constraints, inherent to the scope of HHCP, are imposed. These constraints establish route inter-dependency among the caregivers, either in *simultaneous attendances* or by *precedence constraints*. A first requirement is that both caregivers be simultaneously present with the patient. Thus, the processing of both services starts simultaneously. Additionally, the caregivers can only depart from the patient after the completion of the longest of the two services, as shown in Figure 1a.



(a) Double service with simultaneous attendance

(b) Double service with precendece constraint

Fig. 1: A time-space visualization of a double service patient. In 1a, caregiver #1 arrives at patient #1 before caregiver #2, thus the attendance of the services are delayed until both caregivers are present with patient #1. In 1b, caregiver #2 must start service type #5 within the *dynamic time-window* $[t_e, t_l]$.

In precedence constraints, we have a priority order for which the service types requested must be serviced. Without loss of generality, HHCRSP considers a *global priority* among the service types in $\mathcal{S}$. For example, with a set $\mathcal{S} = \{1, 2, 3\}$, service 1 has priority over services 2 and 3. Similarly, service 2 also has priority over 3. In other words, service i has priority over service j if $i, j \in \mathcal{S}$.

In addition to this priority, patients with precedence constraints require a minimum and maximum separation time between their two service requests. Parameter $\delta^{\min}$ indicates the minimum difference between the start of both service types, which may cause the caregiver responsible for the lower priority service to wait until the start of their operation. Similarly, parameter $\delta^{\max}$ indicates the maximum difference between the start time of the services. In this case, such a requirement can cause the caregiver responsible for the high priority service to delay their operation to maintain the solution feasibility. In the example of Figure 1b, service type 5 can be started at any time after t_e, and before t_l.

3.3. Mixed integer programming model

In addition to discussing the features of the HHCRSP, we also introduce the model of Mankowska et al. (2014) to define the problem formally. This MIP considers three sets of variables. The decision variables $x_{ijvs} \in \{0, 1\}$ indicates if caregiver $v \in \mathcal{V}$ departs from $i \in \mathcal{C}^0$ to $j \in \mathcal{C}^0$ to perform the service type $s \in S$ in the destination node. Considering caregiver capabilities and the patient requirements, we can rewrite the bounds of these variables as $x_{ijvs} \in \{0, a_{vs}r_{is}\}$, allowing the removal of several decision variables from the problem. Variable $t_{ivs} \geqslant 0$ indicates the service start time of patient $i \in \mathcal{C}$ by caregiver $v \in \mathcal{V}$ for service type $s \in \mathcal{S}$. Variable $z_{is} \geqslant 0$ is the tardiness when the service type $s \in \mathcal{S}$ starts after the end of the patient's time window.

The HHCP has three minimization criteria, computed in variables $D, T, T^{\max} \geqslant 0$, respectively, regarding the total travel distances, the total tardiness, and the largest tardiness over all the patients. Those criteria are weighted according the multiplicative factors λ_1, λ_2, and λ_3 in a single objective function. The MIP model for the HHCRSP is given in (1–15).

$$\text{Minimize} \quad \lambda_1 D + \lambda_2 T + \lambda_3 T^{\max} \tag{1}$$

Subject to:

$$D = \sum_{v\in\mathcal{V}} \sum_{i\in\mathcal{C}^0} \sum_{j\in\mathcal{C}^0} \sum_{s\in\mathcal{S}} d_{ij} x_{ijvs} \tag{2}$$

$$T = \sum_{i\in\mathcal{C}} \sum_{s\in\mathcal{S}} z_{is} \tag{3}$$

$$T^{\max} \geqslant z_{is} \qquad \forall i \in \mathcal{C}, s \in \mathcal{S} \tag{4}$$

$$\sum_{i\in\mathcal{C}^0} \sum_{s\in\mathcal{S}} x_{0ivs} = \sum_{i\in\mathcal{C}^0} \sum_{s\in\mathcal{S}} x_{i0vs} \qquad \forall v \in \mathcal{V} \tag{5}$$

$$\sum_{j\in\mathcal{C}^0} \sum_{s\in\mathcal{S}} x_{jivs} = \sum_{j\in\mathcal{C}^0} \sum_{s\in\mathcal{S}} x_{ijvs} \qquad \forall i \in \mathcal{C}, v \in \mathcal{V} \tag{6}$$

$$\sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{C}^0} a_{vs} x_{jivs} = r_{is} \qquad \forall i \in \mathcal{C}, s \in \mathcal{S} \tag{7}$$

$$t_{ivs_1} + p_{is_1} + \mathrm{d}_{ij} \leqslant t_{jvs_2} + M\left(1 - x_{ijvs_2}\right) \qquad \begin{array}{l}\forall i \in \mathcal{C}^0, j \in \mathcal{C}, \\ v \in \mathcal{V}, s_1, s_2 \in \mathcal{S}\end{array} \tag{8}$$

$$t_{ivs} \geqslant e_i \qquad \forall i \in \mathcal{C}, v \in \mathcal{V}, s \in \mathcal{S} \tag{9}$$

$$t_{ivs} \leqslant l_i + z_{is} \qquad \begin{array}{l}\forall i \in \mathcal{C}, v \in \mathcal{V}, \\ s \in \mathcal{S}\end{array} \tag{10}$$

$$\begin{aligned} t_{iv_2s_2} - t_{iv_1s_1} &\geqslant \delta_i^{\min} \\ &- M\left(2 - \sum_{j\in\mathcal{C}^0} x_{jiv_1s_1} - \sum_{j\in\mathcal{C}^0} x_{jiv_2s_2}\right) \end{aligned} \qquad \begin{array}{l}\forall i \in \mathcal{C}^{\mathrm{d}} \\ v_1, v_2 \in \mathcal{V}, \\ s_1, s_2 \in \mathcal{S} : s_1 < s_2\end{array} \tag{11}$$

$$\begin{aligned} t_{iv_2s_2} - t_{iv_1s_1} &\leqslant \delta_i^{\max} \\ &+ M\left(2 - \sum_{j\in\mathcal{C}^0} x_{jiv_1s_1} - \sum_{j\in\mathcal{C}^0} x_{jiv_2s_2}\right) \end{aligned} \qquad \begin{array}{l}\forall i \in \mathcal{C}^{\mathrm{d}}, \\ v_1, v_2 \in \mathcal{V}, \\ s_1, s_2 \in \mathcal{S} : s_1 < s_2\end{array} \tag{12}$$

$$x_{ijvs} \in \{0, a_{vs} r_{js}\} \qquad \forall i, j \in \mathcal{C}^0, v \in \mathcal{V}, s \in \mathcal{S} \tag{13}$$

$$t_{ivs},\ z_{is} \geqslant 0 \qquad \forall i \in \mathcal{C}^0, v \in \mathcal{V}, s \in \mathcal{S} \tag{14}$$

$$D,\ T,\ T^{\max} \geqslant 0 \tag{15}$$

Expression (1) defines the objective function, using the solution quality indicators calculated in constraints (2–4). Constraints (5) indicate that each vehicle should depart and return to the depot once, and constraints (6) model the flow balance to the other vertices. Constraints (7) guarantee that the service requests of the patients are performed by skilled caregivers. Constraints (8) are the sub-tour elimination constraints considering the distances between nodes and service processing times. Parameter M is a large number. Constraints (9, 10) model the time windows and compute the services start times on patients. Constraints (11, 12) model both simultaneous double services and separation times on double services with precedence. Finally, constraints (13–15) define the domain of the decision variables.

4. Proposed solution method

The weak linear relaxation of model (1–15) severely hurts the performance of branch-and-cut methods from the state-of-art MIP solvers (Mankowska et al., 2014). Although new valid inequalities can potentially help to strengthen the linear relaxation of the MIP, the number of variables required to model the problem is still an issue, rendering the employment of solvers impractical for instances with more than 100 patients with a standard computer of 8 GB of memory (Kummer et al., 2019). Furthermore, methods based on the modification of solutions are often ineffective in solving the HHCRSP. Regarding the optimization landscape, the presence of time-windows makes it hard to escape from local optima (Gendreau et al., 2010). Previous work in the literature show how complex it is to keep the solution structure for trajectory-based methods in the presence of route inter-dependency constraints (Ait Haddadene et al., 2016). Drexl (2012) suggests that it may be worthwhile to try to solve problems with route inter-dependencies using an indirect approach over an *auxiliary search space* to overcome feasibility issues.

Instead of crafting nifty local search operators to navigate through the neighborhoods of a promising solution, we propose exploring the problem solution space indirectly through a multi-population multi-parent biased random-key genetic algorithm, with an additional component of implicit path-relinking for intensification of the search (Andrade et al., 2021).

4.1. The biased random-key genetic algorithm

The biased random-key genetic algorithm (BRKGA) has been applied to a wide variety of combinatorial optimization problems, including the container loading problem, the parallel machines scheduling problem, and project scheduling problems (Gonçalves et al., 2011; Gonçalves and Resende, 2012; Chaves et al., 2016). To the best of our knowledge, just one paper in the literature proposes solving the HHCP with a genetic algorithm (Kummer et al., 2020).

In a BRKGA, the initial population is comprised of vectors of real-valued random keys from $[0, 1)$. Let p be the population size of the genetic algorithm. In each generation, individuals of the current population are sorted according to their fitness values. The best p_e individuals of the population form the elite set, and the other $p - p_e$ individuals compose the non-elite set. The BRKGA is elitist and copies all individuals of the elite set of the current population to the next population. This way, the BRKGA monotonically keeps the best solutions found over the search (Gonçalves and Resende, 2011).

A BRKGA uses two strategies to create the offspring in the next population. The first strategy aims to keep the new population's diversity by inserting p_m new random mutants in the next population. The second strategy consists of mating one member from the elite set with another member from the non-elite set by employing a parametrized uniform crossover operator to generate offspring. Each allele of the offspring is either inherited from the elite parent, with a probability of ρ_e , or from the non-elite parent, with a probability of $1 - \rho_e$. The mating generates $p - p_e - p_m$ new individuals, which are inserted in the next population. Finally, the next population becomes the current population, and the process repeats until achieving a maximum number of generations or some other stopping criterion. Figure 2 depicts, at a high level, these operations, highlighting the sequence of operations.

The multi-population BRKGA consists of evolving multiple islands simultaneously. The islands consist of several populations evolving independently, allowing distinct islands to (possibly) achieve distinct local minima. *Immigration* allows exploiting the problem's solution space by moving some individuals from one island to the other. With k individual populations, m immigrants from the island i are moved to the island $\min\{i + 1, 1\}$, and this process repeats for $i = 1, 2, \ldots, k$. Typically, the immigration mechanism is triggered by some criteria during the evolutionary process, e.g., periodically after a fixed number of generations (Toso and Resende, 2015).

A BRKGA has some strengths that help to manage the complicating constraints of the HHCP. It uses a parameterized uniform crossover, dismissing the necessity of repair operators by delegating feasibility issues to the problem's objective function, requiring a well-designed decoder to prevent the evolutionary process from getting stuck due to infeasibilities (Gonçalves and Resende, 2011). The choice of chromosome representations with random keys and the constant generation of mutants enables exploring the solution space. In contrast, the crossover operator and the decoder exploit the structural properties of the solutions (Eiben et al., 2003). Such characteristics also relieve the necessity to employ local search, which is frequently a very time-consuming component of the heuristics and has a negligible impact on the solutions (Drexl, 2012).

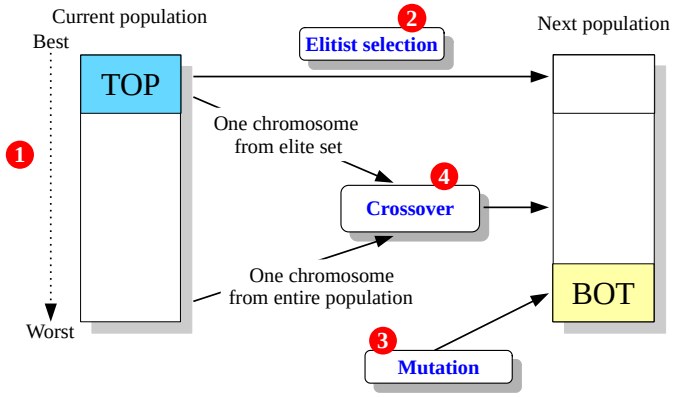


Fig. 2: Overview of the BRKGA internals. The operation (1) sorts the current population. In (2) the elite individuals are copied to the next population. In (3), some new mutant individuals are generated to keep the diversity of the next population. In (4), the remaining space of the next population is filled up with offspring.

4.2. Multi-parent BRKGA

As aforementioned, the *standard* BRKGA performs the mating process over two parents from the population. Further researchers identified simple modifications that improve the convergence of the mating process. Computational experiments presented in Lucena et al. (2014) showed that the multi-parent mating produces consistently better results than the standard and the *gender-defining* variant of BRKGA (Lucena et al., 2014).

In multi-parent BRKGA, the metaheuristic selects a total of π_t parents for each mating, with π_e individuals from the elite set, and the other $\pi_t - \pi_e$ from the non-elite set. Contrary to specifying the bias for selecting alleles from the parents directly, the multi-parent BRKGA uses a two-step approach. First, the parents are put into a list, which is then sorted according to their *fitness*. The second step consists of applying a bias function to compute each parent's weight according to their *rank* r in the sorted list. Andrade et al. (2021) suggest using one of the non-decreasing bias functions Φ to compute such weights (Bresina, 1996). In addition to these functions, we derived the quadratic $\Phi(r) = r^{-2}$ and cubic $\Phi(r) = r^{-3}$ functions.

4.3. Implicit path-relinking in random-keys space

Path-relinking is an intensification procedure that explores the intermediate solutions between a base and a guide solution by incrementally introducing changes in the base solution according to the guide. It finishes when both guide and base solutions are equal (Glover, 1997), or after exploring a maximum number of intermediate solutions (Resende et al., 2010). According to the studies of Resende and Ribeiro (2016), the *direction* of this exploration matters. In the *forward path-relinking*, the changes come from the guide solution. Conversely, the algorithm is called *backward path-relinking* when the changes come from the base solution. Resende and Ribeiro (2016) also propose the *mixed path-relinking*, which combines both forward and backward strategies. According to their experiments, this mixed intensification approach often finds a better solution than the others.

A typical implementation of the path-relinking heuristic is usually tied closely to the problem (Andrade et al., 2021). The heuristic is structured around the solution components (or variables), and how these components can be interchanged between the base and guide solution to generate new intermediate solutions. It is very hard to generalize a single implementation of path-relinking to a wide variety of problems. Nevertheless, Andrade et al. (2021) propose a path-relinking implementation that takes advantage of the generic solution encoding of the BRKGA to implement what they call an implicit path-relinking procedure (IPR).

As the reader may expect, there is a significant overhead of applying a path-relinking heuristic inside another metaheuristic such as the BRKGA. For this reason, Andrade et al. (2021) also propose a minimum distance metric between the base and the guide solution to allow running the IPR. This strategy relies on the fact that measuring such distances is much faster than running the heuristic itself, so it works as an effective speedup strategy to skip IPR runs between solutions that are too similar.

Andrade et al. (2021) suggested two variations of the IPR that operates differently according to how the solutions are encoded. Specifically to routing and sequencing problems, the decoder uses the random keys to establish a permutation of the solution elements. For a vehicle routing problem, e.g., each client

is associated with a chromosome position; thus, the visit order is obtained by sorting the chromosome (Ruiz et al., 2019). For this type of solution encoding, Andrade et al. (2021) proposed a *permutation IPR* that uses the guide chromosome to *induce permutations* in the base chromosome, as shown in Figure 3. This variation of the implicit IPR starts by sorting both *guide* and *base individuals*. At this point the algorithm tests for the minimal distance criterion, using the Kendall-tau distance over the permutation of array indices of $sort(b)$ and $sort(g)$. After that, the IPR generates *intermediate individuals* by swapping keys in either *base* or *guide* individual, following a hybrid forward-backward path relinking strategy. In the example of the figure, the guide individual sets that the first key must be associated with index 5 (according to sort(g)), but the base solution associates the index 1 to the first key (according to sort(b)). The IPR then swaps the key associated with the index 5 in b (following the permutation of g), with the key currently occupying position 1 of b. This procedure generates the individual b', whose first element (in *sort*(b')) respects the first element in the permutation of g. After introducing this change, the decoder evaluates the intermediate individual, hopefully finding an improved solution.

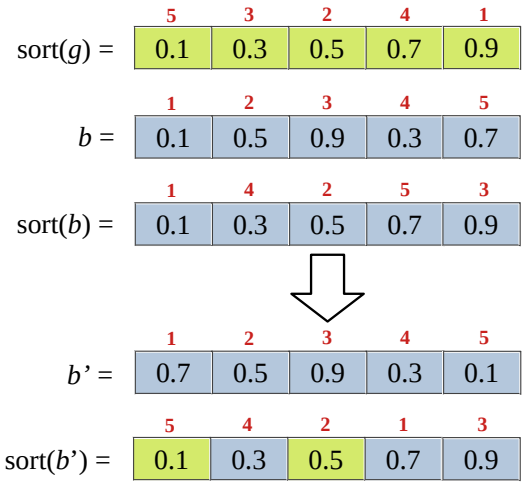


Fig. 3: Example of *permutation IPR*. The guide individual g is used to induce changes in the base individual b.

In presence of multiple islands, the implicit path-relinking is applied between pairs of elite individuals from distinct islands, either by selecting the best individuals or selecting random individuals from their elite sets, following the same circular strategy of the immigration mechanism. For example, with three islands, the IPR runs between some elite individuals from islands 1 and 2, then from islands 2 and 3, and finally 3 and 1. With the "best selection" strategy, each IPR run selects the elite individuals according to their fitness. With the random selection, the IPR simply selects two random individuals from the elite sets. When only a single island is considered, the heuristic operates between elite individuals from the same island (Andrade et al., 2021).

4.4. Heuristic-powered decoder for the HHCRSP

In our solution approach, the genetic algorithm evolves the sequence in which the patients are inserted into the solution, and each chromosome has length $|\mathcal{C}| + 2$. The example of Figure 4 demonstrates the decoding process on an instance with three patients. The algorithm starts pairing the list of patients with the chromosome, then it sorts the task list according to the increasing order of the allele values, reaching the task sequence depicted in *task insertion sequence*. This process uses the first $|\mathcal{C}|$ alleles of the chromosome. Then, a greedy cheapest-insertion heuristic assigns the sorted sequence of tasks to the caregivers, and the last two alleles are used to toggle optional components of the decoding algorithm. When allele $|\mathcal{C}| + 1 < 0.5$, the decoder only considers the increase in total travel time by inserting the patient into the caregiver's route. When this allele has a value $\geqslant 0.5$, then the decoder also includes the distance for returning the caregiver to the depot after servicing the patient. In the context of this manuscript, we refer to this behavior as the *convex hull* strategy. The last allele $|\mathcal{C}| + 2$ toggles a simple heuristic that tries to balance the workload of caregivers during the greedy construction phase.

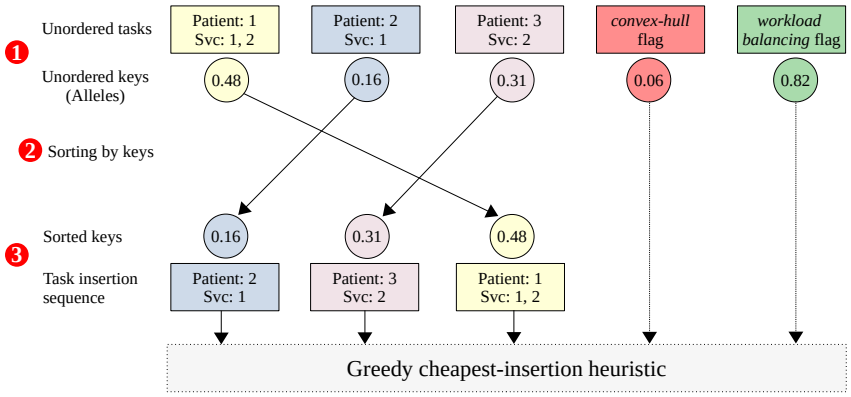


Fig. 4: Decoding example for a instance with three patients.The greedy phase has the *convex hull* strategy set off and the caregiver workload balancing heuristic set on.

Algorithm 1 details the implementation of the proposed decoder. In line 1, the decoder starts an empty solution, with all the vehicles placed at the depot. Line 2 performs the same steps depicted in the example of Figure 4. Lines 3 and 4 check if the optional components should be enabled during the rest of the decoding. Line 5 initializes the total workload of each caregiver as 0. The decoder then takes each task from the sorted task sequence (line 6) and tests all the possible caregiver assignments to such task (lines 7–19), considering all the service types requested by the patient. This is done by passing the service types requested by $tasks\,[i]$ to the function `AllVehicleCombinations`, which returns a list with all pairs of vehicles with qualification to perform the service types s_1 and s_2, respectively. The heuristic then evaluates the cost of assigning the task to the current vehicles v_1 and v_2 through `FindInsertionCost` (line 10). Lines 11–13 update the cost $\bar{c}$ according to the workload balancing heuristic. Lines 14-18 check if the current pair of vehicles leads to a cheaper insertion cost (strict improvement). If the cost current choice of vehicles $\bar{c}$ is very close to c^*, then the algorithm still updates the choice of best candidate

vehicles v_1^* and v_2^* if the value of the allele associated with the task is $\geqslant 0.5$. This allows the genetic algorithm to reach elements of the solution space otherwise inaccessible. After deciding the best pair of candidates, the algorithm updates the solution by inserting the patient into the routes of the selected caregivers (line 20). Line 21 updates the cost of s using a similar algorithm to `FindInsertionCost`. Lines 22–25 update the workload of the selected vehicles. Finally, the decoder returns the final solution cost at line 27. Considering $n = |\mathcal{V}|$ and $m = |\mathcal{C}|$, the decoder algorithm has the worst-case complexity of $O(mn^2)$.

Algorithm 1: Greedy cheapest-insertion constructive heuristic.

Input: A chromosome ch.
Output: Solution cost.

1 $s \leftarrow$ `EmptySolution()`
2 $tasks \leftarrow$ `Sort`$(\mathcal{C}, ch)$ `/* Gets the permutation of patients. */`
3 $convHull \leftarrow$ `GetKey`$(ch, |\mathcal{C}| + 1) \geqslant 0.5$ `/* Uses the second to last key. */`
4 $wloadHeur \leftarrow$ `GetKey`$(ch, |\mathcal{C}| + 2) \geqslant 0.5$ `/* Uses the last key. */`
5 $wload \leftarrow$ `Zeros`$(|\mathcal{V}|)$ `/* List of zeros. */`
6 **for** $i \leftarrow 1$ **to** $|tasks|$ **do**
7 $\quad c^* \leftarrow \infty$
8 $\quad (s_1, s_2) \leftarrow$ `GetSkills`$(tasks[i])$
9 $\quad$ **for** $(v_1, v_2) \in$ `AllVehicleCombinations`(s_1, s_2) **do** `/* All pairs of qualified vehicles. */`
10 $\quad\quad \bar{c} \leftarrow$ `FindInsertionCost`$(s, v_1, v_2, tasks[i], convHull)$
11 $\quad\quad$ **if** $wloadHeur =$ `true` **then** `/* Workload balancing heuristic. */`
12 $\quad\quad\quad \bar{c} \leftarrow \bar{c} + wload[v_1] + (s_2 \neq$ `null` $\,?\, wload[v_2] : 0)$
13 $\quad\quad$ **end**
14 $\quad\quad$ **if** $\bar{c} < c^*$ **or** $(\bar{c} = c^*$ **and** `GetKey`$(tasks[i]) \geqslant 0.5)$ **then** `/* Strict/non-strict improvement. */`
15 $\quad\quad\quad v_1^* \leftarrow v_1$
16 $\quad\quad\quad v_2^* \leftarrow v_2$
17 $\quad\quad\quad c^* \leftarrow \bar{c}$
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 $\quad$ `UpdateRoutes`$(s, v_1^*, v_2^*, tasks[i])$ `/* Updates solution structure. */`
21 $\quad$ `UpdateCost`$(s, v_1^*, v_2^*, tasks[i])$ `/* Updates incumbent. */`
22 $\quad wload[v_1^*] \leftarrow wload[v_1^*] +$ `ServiceDuration`$(task[i], s_1)$ `/* Updates total workload of` v_1^*. `*/`
23 $\quad$ **if** $s_2 \neq$ `null` **then**
24 $\quad\quad wload[v_2^*] \leftarrow wload[v_2^*] +$ `ServiceDuration`$(task[i], s_2)$ `/* Updates total workload of` v_2^*. `*/`
25 $\quad$ **end**
26 **end**
27 **return** `GetCost`(s)

In addition to this description of the decoding algorithm, a few points still need to be explained. First is that function `GetSkills` returns all the service types requested by the patient and, in case of a single service request, s_2 will have the special value `null`. `AllVehicleCombinations` is also aware of this behavior, so in such cases, all tuples from the returned list of vehicles will have v_2 equals to `null`. The value of s_2 is also used to decide which vehicles are considered when calculating the workload balancing (line 12) and when updating the workload of the vehicles selected to service to a patient (lines 23–25). Note that function `FindInsertionCost` can also handle the cases in which v_2 is `null`.

The cost calculation in the insertion heuristic is not straightforward due to the double services and soft time windows. Algorithm 2 details how this can be implemented, considering all the soft constraints of the problem. The function starts by querying the current location of vehicles in solution s (line 2). Line 3 calculates the arrival time of v_1 at the patient i, taking into account the travel time between the current vehicle location and i. In case of a single service patient, the algorithm proceeds with lines 5–10, calculating visit tardiness. In case of the *convex hull* strategy to be set on, the algorithm takes a few more steps to subtract the travel times from $vpos_1$ to depot, to then add the travel times from node i to depot (lines 8–10). In the case of a double service patient, the algorithm calculates the arrival time for v_2, as well as the increase in travel times. Lines 14–17 apply similar calculations as lines 8–10. In the case of simultaneous double services, the algorithm calculates the service start times according to line 19. The next two lines calculate the visit tardiness. For a double services patient with precedence, the algorithm calculates the visit times according to the minimum and maximum separation times for patient i (lines 23–27). If the maximum separation time is violated, then a delay is inserted into the start time of the first vehicle (lines 25–27). Violations of the soft time windows are computed in lines 6, 20, and 28, and the maximum tardiness in lines 7, 21, and 29. Finally, the function takes the updated solution quality indicators and returns its cost (lines 32–35). Assuming that all operations inside `FindInsertionCost` run at constant time, the function has a complexity of $O(1)$.

Despite its similarity to the decoding algorithm proposed by Kummer et al. (2020), our new decoder has some important additions that allow the genetic algorithm to reach a larger amount of the solution space of the HHCRSP. These additions–notably the *convex hull* strategy–allow our BRKGA to achieve solutions otherwise inaccessible by previous work in the literature. Similarly, the *workload balancing heuristic* can also help to improve solution quality when a problem instance has similar vehicles with respect to qualification levels, which indirectly guide to solutions with improved travel times.

5. Computational results

This section presents computational results for our proposed metaheuristic. We describe the benchmark dataset used in our computational experiments and the benchmark machine. We also compare our results with all the previous publications that use this same dataset, and we highlight which solution methods work best for each instance tested.

5.1. Computational environment and benchmark dataset

We use in our experiments an Intel Xeon E5-2697 v2 computer running at 2.70 GHz, with 64 GB of memory. Our code is implemented in C++, and we compile our binary using the GNU G++ compiler, version 7.5.0 running Ubuntu 18.04 with Linux kernel 4.15.0. We use the open-source `brkga_mp_ipr_cpp` framework of Andrade et al. (2021). This framework not only allows developing standard BRKGA algorithms, including the variant with an island model, but it also supports the IPR and the MP mating described in the former sections. This way, most of our implementation effort is to develop a problem-specific decoder compatible with the Andrade et al. framework, altogether with some auxiliary code to handle the problem data. The `brkga_mp_ipr_cpp` framework supports the new C++17 standard. Similar to its

Algorithm 2: Implementation of `FindInsertionCost`.

Input: s, v_1, v_2, $task$, $convHull$.
Output: Updated solution cost of appending $task$ to the routes of v_1 and v_2.

$i \leftarrow \texttt{GetNode}(task)$
$(vpos_1, vpos_2) \leftarrow \texttt{VehiclesLocation}(s, v_1, v_2)$ /* Current location of v_1 and v_2. */
$arrival_1 \leftarrow \texttt{max}\{e_i, \texttt{LeaveTime}(s, v_1) + \texttt{TravelTimes}(vpos_1, i)\}$
$dist \leftarrow \texttt{TravelTimes}(vpos_1, i)$
if $i \in \mathcal{C}^{\text{s}}$ **then** /* Processes a single service patient. */
  $tard \leftarrow \texttt{max}\{0, arrival_1 - l_i\}$ /* Vehicle starts processing immediatelly at $arrival_1$. */
  $tmax \leftarrow tard$
  if $convHull = \texttt{true}$ **then**
    $dist \leftarrow dist - \texttt{TravelTimes}(vpos_1, 0) + \texttt{TravelTimes}(i, 0)$
  end
else
  $arrival_2 \leftarrow \texttt{max}\{e_i, \texttt{LeaveTime}(s, v_2) + \texttt{TravelTimes}(vpos_2, i)\}$
  $dist \leftarrow dist + \texttt{TravelTimes}(vpos_2, i)$
  if $convHull = \texttt{true}$ **then**
    $dist \leftarrow dist - \texttt{TravelTimes}(vpos_1, 0) + \texttt{TravelTimes}(i, 0)$
    $dist \leftarrow dist - \texttt{TravelTimes}(vpos_2, 0) + \texttt{TravelTimes}(i, 0)$
  end
  if $\delta_i^{\min} = \delta_i^{\max} = 0$ **then** /* Simultaneous double service. */
    $svcStart \leftarrow \texttt{max}(arrival_1, arrival_2)$
    $tard \leftarrow 2 \cdot \texttt{max}\{0, svcStart - l_i\}$
    $tmax \leftarrow \texttt{max}\{0, svcStart - l_i\}$
  else /* Double service with precedence. */
    $svcStart_1 \leftarrow arrival_1$
    $svcStart_2 \leftarrow \texttt{max}\{arrival_1 + \delta_i^{\min}, arrival_2\}$
    if $svcStart_2 - svcStart_1 > \delta_i^{\max}$ **then** /* v_2 arrived too late at the patient. */
      $svcStart_1 \leftarrow svcStart_2 - \delta_i^{\max}$ /* Adds some waiting to v_1. */
    end
    $tard \leftarrow \texttt{max}\{0, svcStart_1 - l_i\} + \texttt{max}\{0, svcStart_2 - l_i\}$
    $tmax \leftarrow \texttt{max}\{\texttt{max}\{0, svcStart_1 - l_i\}, \texttt{max}\{0, svcStart_2 - l_i\}\}$
  end
end
$dist \leftarrow dist + \texttt{GetTravelTimes}(s)$
$tard \leftarrow tard + \texttt{GetTotalTardiness}(s)$
$tmax \leftarrow \texttt{max}\{tmax, \texttt{GetTMax}(s)\}$
return $\lambda_1 \cdot dist + \lambda_2 \cdot tard + \lambda_3 \cdot tmax$

predecessor, the `brkgaAPI` of Toso and Resende (2015), it also supports multi-core processing through OpenMP directives. Thus, we take advantage of our benchmark machine's parallel capabilities to decode individuals in parallel using four *real* cores–we explicitly disabled the *hyperthreading* feature during our experiments.

We test out metaheuristic algorithm using the benchmark dataset proposed by Mankowska et al. (2014). This dataset is comprised of seven instance subsets, each one composed of ten synthetic instances. Table 2 highlights the general properties of each subset. The columns identify the instance family, the total number of patients, and the number of single service and double service patients, respectively. The last

column presents the number of caregivers. The set of service types $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$ is the same for all the 70 instances, and the selection of the requested service types per patient is at random. Each instance comprises 30% of double service patients, 15% requiring simultaneous double services, and 15% requiring double services with precedence constraints. The other 70% are single service patients. Note that single service patients require just one service type, and double service patients require exactly two. The caregiver skills are also set at random. Half of the caregivers can perform up to three service types from the set $\{1, 2, 3\}$. The same happens to the other half of $\mathcal{V}$, considering the service types $\{4, 5, 6\}$. The authors guarantee at least one caregiver capable of performing each service type. All the points (the depot and patient homes) were generated within a 100×100 grid, and the travel times are symmetric. Each patient has a two-hour time-windows drawn from a time horizon of ten hours.

Table 2: Characteristics of the benchmarking dataset proposed by Mankowska et al. (2014). For each instance subset, we indicate the total number of patients $|\mathcal{C}|$, and the number of single $|\mathcal{C}^{\mathrm{s}}|$ and double service patients $|\mathcal{C}^{\mathrm{d}}|$, respectively. Column $|\mathcal{V}|$ indicates the number of caregivers considered.

Instance subset	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{C}^{\mathrm{s}}\rvert$	$\lvert\mathcal{C}^{\mathrm{d}}\rvert$	$\lvert\mathcal{V}\rvert$
A	10	7	3	3
B	25	17	8	5
C	50	35	15	10
D	75	52	23	15
E	100	70	30	20
F	200	140	60	30
G	300	200	100	40
# of instances		70		

Lastly, we use the weights $\lambda_1 = \lambda_2 = \lambda_3 = {}^1/_3$ in the objective function. These are the same weights used in all previous publications that use the Mankowska et al. (2014) dataset.

5.2. Improved lower bounds with CPLEX 20.1.0.0

During computational experiments with CPLEX 12.3, Mankowska et al. (2014) attempted to solve their instance dataset by applying the MIP model (1–15). According to the authors, only instances with up to ten patients could be solved to optimality, and feasible solutions could only be found for other test cases with 25 patients. For these experiments with the MIP, the authors also report the best lower bound found within 10 hours of solver runtime. Thanks to the generation of additional cuts, its value improves during the branch-and-cut phase due. Thus, these best lower bounds are stronger than the simple lower bounds from the MIP linear relaxation. For the test cases with 200 and 300 patients, the authors simplify the experiments by relaxing the route inter-dependency constraints (11) and (12). This way, the best lower bound provided for these instances are (provably) weaker than the best lower bounds for the HHCRSP.

Prior to assessing the solution quality for our proposed metaheuristic algorithm, we first experiment with the Mankowska et al. model to obtain stronger new best lower bounds (LB^{+}), using the most updated version available of the CPLEX solver. In our experiment, we consider all the constraints for the MIP (1–15). We also use constraints (13) to skip generating decision variables that are trivially fixed to zero due to

either the lack of caregiver skills or missing patient request for some service types. Such preprocessing also allows us to skip generating empty flow constraints (6). With these minor improvements, we significantly reduce the MIP memory consumption by a third for instances with 300 patients. We solve the MIP using CPLEX 20.1.0.0 and we applied three configurations to the solver, aiming to test the standard optimizer performance (Defaults), and to test the effectiveness of our strategies for reducing the growth in memory consumption as the search progresses (configurations #1 and #2). We achieve the following parameter setting presented in Table 3 by inspecting the optimizer documentation (IBM, 2020). Reducing the number of threads available to CPLEX reduces the growth in memory consumption as the branch-and-cut algorithm proceeds. The solver also has a "memory emphasis" flag that disables some memory-intensive pre-processing routines, at expanse of worsening the solver performance.

Table 3: Parameter setting for CPLEX while obtaining strong best lower bounds.

Parameter	Defaults	Configuration #1	Configuration #2
# threads	(automatic)	1	1
Memory emphasis	off	on	on
MIP warmstart	off	off	on

In addition to the tweaked parameters, preliminary experiments indicate that CPLEX can greatly benefit from a *warm start* from a feasible solution to enhance further both memory usage and to produce better values of LB^+. This way, we use the genetic algorithm of Kummer et al. (2020) to produce such initial solutions. We use most of the parameter settings proposed in their paper, but we limit the number of evolved generations to 600 to reduce the time spent on the warm start. In these experiments, we use the same computer we describe in Section 5.1. We use CPLEX command line interface to run our tests, and we set a time limit of two hours for each run of the solver. To better represent a *standard personal computer*, we also limited CPLEX to use up to 8 GB of memory.

Table 4 presents our new average best lower bounds for each instance subset. The first column presents the instance subset's name, followed by the computational time and the best lower bounds reported by Mankowska et al. (2014). Note that we adjusted the times reported by Mankowska et al., as explained in Subsection 5.4. The next columns present our average results for each instance subset, for the standard CPLEX configuration, and configurations #1 and #2. For each configuration, we report the average solver runtime in seconds, the average number of branch-and-cut nodes explored, and the average LB^+. At the bottom of the table, we present the number of instances in which each experiment produced the best lower bound (# Best), the number of proved optimal solutions (# Opt), the number of instances that achieved a time limit of 2 hours (# TL), and the number of runs that were stopped due to out-of-memory (# OOM).

On average, the newer version of CPLEX can find better LB^+ values than the previously reported in the literature, although the most remarkable difference is that we use only 20% of the processing time of Mankowska et al. (2014) to produce these new results. We also managed to prove the optimal solutions to 7 out of 10 instances of subset B, which justifies the average running time of 3169.94 seconds for this group of instances (using CPLEX defaults). These results are obtained at the expense of exploring gigantic branch-and-cut trees with more than 10^6 nodes on average. Despite that, we observed no trees consuming more than 500 MB of memory for subset B. Naturally, the number of nodes explored drops as the problem size increases, as a consequence of the large time spent solving LP subproblems. The smaller

Table 4: New best lower-bound found with CPLEX 20.1.0.0. Values in bold indicate the best lower-bound available.

Inst. subset	Literature		Defaults			Configuration #1			Configuration #2		
	T (sec.)	LB^+	T (sec.)	# Nodes	LB^+	T (sec.)	# Nodes	LB^+	T (sec.)	# Nodes	LB^+
A	5.47	**225.18**	0.26	2158.70	**225.18**	0.61	1038.20	**225.18**	0.27	656.70	**225.18**
B	37,908	343.11	3169.94	2,445,294.20	387.14	7200.00	1,867,494.20	345.90	3250.88	705,862.00	**391.29**
C	37,908	342.62	4640.88	357,600.10	384.24	7200.00	217,070.70	382.00	7200.01	222,574.10	**393.75**
D	37,908	377.36	5595.87	123,144.00	399.73	7200.01	32,363.10	408.71	7200.02	38,002.30	**412.71**
E	37,908	404.53	4952.80	58,249.20	402.10	7200.02	9,987.50	**428.47**	7200.06	12,679.80	425.55
F	37,908	435.26	3210.13	2696.30	492.88	7200.37	1,682.90	530.97	7200.40	690.90	**536.43**
G	37,908	462.12	–	–	–	7200.84	358.00	**620.76**	7200.62	336.40	604.59
# Best	9		31			29			46		
# Opt	10		17			10			17		
# TL	60		17			60			53		
# OOM	0		36			0			0		

improvement of LB^+ on subsets D and E is related to some instances that are particularly difficult to solve, e.g., E3, for which the LB^+ improvement is only 3.90% between the literature and configuration #1, and 3.27% between the literature and configuration #2. We believe this issue may be related to phase transition between easy and hard instances of the HHCRSP, which is often discussed in the literature of other optimization problems.

5.3. Automatic algorithm configuration with irace

As the reader may expect, the presence of several components makes our proposed genetic algorithm heavily parameterized. In such scenarios, a state-of-art approach to configure an algorithm is to apply an automatic algorithm configuration (AAC) tool to automate the finding of good set of parameters. The AAC approach has several inherent advantages over the manual configuration procedure. It reduces the human effort, allowing the practical evaluation of thousands of configurations automatically. It also helps prevent experimental errors or preconceptions in consequence of any human bias with respect to algorithm behavior (Eggensperger et al., 2019). Tools like *irace* (López-Ibáñez et al., 2016) require a very minimal effort to set up an AAC experiment. First, we need to specify the *range of values* for each parameter we want *irace* to configure. We also need to provide a set of *training instances* and a *budget* of maximum runs of the algorithm to tune. *Irace* then evaluates many configurations automatically by iteratively sampling new ones, evaluating them over the training dataset, and then refining the sampling model towards promising configurations generated so far.

To configure our metaheuristic algorithm, we employed *irace* version 3.4.1 (R 3.4.4), and we used the same machine we described in Section 5.1. Regarding metaheuristic components, we set three parameters for controlling the "traditional" BRKGA elements; three parameters for the multi-parent mating component; three parameters controlling the island model; and four parameters controlling the implicit path relinking. We consider a total of 13 parameters listed in Table 5. The first column presents the parameter description, and the second column presents the domain of each parameter.

The third column depicts the range of values we allow *irace* to evaluate. The last column presents the parameter setting set by *irace* as the best configuration found after a budget of 300 runs of our BRKGA-based algorithm. Additionally, we use the G instance subset as training instances during the AAC experiment.

Table 5: Parameters and range of values configured by *irace*.

Parameter description	Type	Range of values	Best config.
Population size	`int`	$[500, 1500]$	1462
Elite (%)	`real`	$[0.05, 0.4]$	0.30678
Mutant (%)	`real`	$[0.01, 0.3]$	0.07575
# of parents	`int`	$[2, 6]$	5
# of elite parents	`int`	$[1, 5]$	4
Bias function	`categorical`	{`constant`, `cubic`, `exponential`, `linear`, `loginverse`, `quadatic`}	`constant`
Independent populations	`int`	$[2, 6]$	2
Immigrants	`int`	$[3, 150]$	73
Exchange elite frequency	`int`	$[30, 180]$	167
IPR pairs	`int`	$[1, 100]$	95
IPR selection criteria	`categorical`	{`best`, `random`}	`random`
IPR path (%) to explore	`real`	$[0.01, 1.0]$	0.33754
IPR frequency	`int`	$[5, 100]$	40

Some of the parameters required by our algorithm were excluded from the AAC experiment, and we set them to fixed values. For example, `brkga_mp_ipr_cpp` supports both *direct* and *permutation* implicit path-relinking, but we fixed our implementation to always the permutation IPR due to the properties mentioned by Andrade et al. (2021). We also set the minimum distance threshold for IPR runs to zero, leaving to *irace* to decide the frequency which IPR should be applied, as well as how many pairs of individuals to try.

From our computational experiments, we observe the island model helps to cope with the typical issue diversity loss among the evolutionary process. Using the parameter setting from Table 5, 16% of the elite sets immigrate from one island to the others periodically after 167 generations, keeping some diversity as the search progresses. Additionally, the implicit path-relinking also improves the diversity by generating hybrid solution from individual of distinct islands. Typically these intermediate individuals improve the diversity of both elite sets over the time. Considering the standard deviation metric applied to individuals' fitness, we observe an increase in the diversity of the elite set each time a new solution is found, mainly by the mating process, by immigration, and by the IPR, in this order.

The stopping criterion of our genetic algorithm is calculated dynamically, and it is proportional to the number of patients of each test cases. This way, our algorithm stops the search after achieving $\frac{|\mathcal{C}|}{2}$ generations without improvement.

5.4. Comparison with local-search-based method from literature

In this section, we compare our results with those of local-search-based methods from the literature. As we mention previously, several authors test their methods against the same dataset of Mankowska et al. (2014), enabling us to compare distinct approaches with our proposal. Despite that, we did not replicated all the experiments from the literature. Instead, we compare against the results from their original publications. We then correct the published processing times to our benchmark machine using the *PassMark* scores as comparison criteria (PassMark, 2021). Table 6 present these factors for Mankowska et al. (2014) and Lasfargeas et al. (2019). The first column presents the reference, and the second column presents the description of the machine used in their experiments. The third column presents the machine score according to *PassMark*. The last column presents a speed factor we use to correct their published processing times. As the table shows, our machine is actually *slower* than the ones used in all previous works, so we might expect to run our metaheuristic algorithm *faster* in their machines than ours. Furthermore, the source code of both Kummer et al. (2019) and Kummer et al. (2020) are available online, so we could re-run their experiments in our benchmark machine.

Table 6: Comparison of computational environments. Neither Mankowska et al. (2014) nor Lasfargeas et al. (2019) specify which machine they use in their experiments. Considering the date of the publications and the documentation available on the Intel website, we guess that Mankowska et al. use an Intel i3-4130, and that Lasfargeas et al. use an Intel i7-6700K as their benchmark machines.

Publication	Machine description	PassMark score	Factor
Mankowska et al. (2014)	Intel Core at 3.40 GHz	1887	1.0530
Lasfargeas et al. (2019)	Intel i7 at 4.00 GHz	2523	1.4080
This work	Intel Xeon E5-2697 v2 at 2.70 GHz	1792	1.0000

Table 7 presents the previous results from the literature, together with the results for the BRKGA of this paper. We solve each instance from Mankowska et al. dataset 20 times, using the numbers between 1 to 20 as seeds. We also rerun the fix-and-optimize *matheuristic* of Kummer et al. (2019) 20 times, using the same seeds as to our algorithm. The first column presents the instance name. The second column presents the new best lower bounds we discussed in Section 5.2. The next two columns present the results from Mankowska et al. (2014), respectively, the solve times adjusted using the factors we discussed earlier, and the solution cost reported by the authors. Note that the algorithms proposed by Mankowska et al. (2014) are all deterministic, so the authors reported results for a single run on each instance. The next four columns present the results from Lasfargeas et al. (2019): their solve times–also corrected using the speed factors, the best solution value they produced, and the average and standard deviation of the solution cost. According to their report, each instance was solved 40 times. The next four columns present the data as to Lasfargeas et al. (2019), but these columns report the values for the fix-and-optimize *matheuristic* proposed by Kummer et al. (2019). Again, the last four columns present these same data, but for the BRKGA-based algorithm proposed in this work. At the bottom of the table, we present the average solve time for each approach (Avg time (sec.)), the number of instances which each method produced the best solution value (# Best), and the number of times which each method produced the best average solution value (# Best avg).

To ease our writing, we refer to Mankowska et al. (2014), Lasfargeas et al. (2019), and Kummer et al. (2019) by MBB2014, LGS2019, and KBA2019, respectively.

Table 7: Extensive computational results comparing the performance of local search-based method from the literature, and our proposed BRKGA.

Inst.	LB^+	MMB2014		LGS2019				KBA2019				BRKGA			
		Time (sec)	Cost	Time (sec)	Best	Avg	SD	Time (sec)	Best	Avg	SD	Time (sec)	Best	Avg	SD
A1	218.20	2.11	**218.20**	<1	**218.20**	224.30	14.60	0.15	**218.20**	219.19	4.44	0.20	**218.20**	**218.20**	0.00
A2	246.63	5.27	**246.60**	<1	**246.60**	258.10	31.60	0.15	**246.63**	**246.63**	0.00	0.17	**246.63**	**246.63**	0.00
A3	305.86	7.37	**305.90**	<1	**305.90**	358.80	38.50	0.39	**305.86**	**305.86**	0.00	0.16	**305.86**	**305.86**	0.00
A4	186.90	8.42	**186.90**	<1	**186.90**	196.40	11.10	1.10	**186.90**	189.25	7.24	0.16	**186.90**	**186.90**	0.00
A5	189.54	2.11	**189.50**	<1	**189.50**	216.50	36.60	0.19	**189.54**	194.00	13.73	0.18	**189.54**	**189.55**	0.01
A6	200.10	2.11	**200.10**	<1	**200.10**	**200.10**	0.00	0.15	**200.10**	**200.11**	0.03	0.15	**200.13**	**200.13**	0.01
A7	225.40	1.05	**225.40**	<1	**225.40**	232.40	24.20	0.11	**225.37**	**225.37**	0.00	0.18	**225.37**	**225.37**	0.00
A8	232.05	4.21	**232.00**	<1	**232.00**	281.40	47.60	0.16	**232.05**	**232.05**	0.00	0.16	**232.05**	**232.05**	0.00
A9	222.30	21.06	**222.30**	<1	**222.30**	225.40	4.00	2.10	**222.30**	**222.30**	0.00	0.20	**222.30**	223.75	3.57
A10	225.01	1.05	**225.00**	<1	**225.00**	**225.00**	0.00	0.09	**225.01**	226.55	6.91	0.13	**225.01**	**225.01**	0.00
Avg	225.20	5.48	**225.19**	<1	**225.19**	241.84	20.82	0.46	**225.19**	226.13	3.23	0.17	**225.20**	**225.35**	0.36
B1	428.10	<1	458.90	74.76	434.10	552.80	93.40	36.17	**428.10**	434.79	8.06	0.78	**428.10**	**428.53**	0.15
B2	476.05	37,908.00	476.20	39.00	**476.00**	561.30	61.40	10.85	**476.05**	483.03	17.71	0.87	**476.05**	**476.92**	2.34
B3	399.09	37,908.00	399.20	89.41	**399.10**	527.60	72.50	177.66	**399.09**	419.51	12.07	0.99	402.80	**409.29**	4.78
B4	411.30	37,908.00	576.00	94.05	414.00	509.70	74.50	17.48	**411.30**	439.23	17.23	1.13	422.06	**430.46**	8.81
B5	366.34	25.76	391.10	19.29	385.60	496.90	98.10	64.06	**366.34**	390.17	20.30	0.98	369.44	**375.15**	3.88
B6	405.58	99.09	534.70	61.53	447.80	611.80	129.90	220.15	**441.70**	516.99	84.59	1.22	470.59	**470.70**	0.16
B7	328.67	14.81	355.50	86.59	**328.70**	398.80	64.80	35.97	**328.67**	334.33	9.72	0.93	**328.67**	**328.67**	0.00
B8	357.68	37,908.00	357.80	111.65	359.70	488.70	116.20	30.71	**357.68**	373.95	16.85	0.70	**357.68**	**359.40**	0.66
B9	330.30	37,908.00	403.80	87.44	404.10	483.40	60.30	185.51	**402.67**	410.59	18.52	0.91	404.11	**404.29**	0.23
B10	420.99	31.34	500.40	12.25	**462.70**	616.80	147.70	184.17	**462.75**	479.48	17.31	0.85	469.58	**469.58**	0.00
Avg	392.41	21079.00	445.36	67.60	411.18	524.78	91.88	96.27	**407.43**	428.21	22.24	0.93	412.91	**415.30**	2.10
C1	459.25	180.13	1123.60	135.45	974.20	1350.40	365.30	1058.09	**957.05**	1001.48	60.67	3.09	969.11	**973.87**	3.52
C2	373.94	114.99	673.80	149.81	605.10	685.50	55.60	330.47	**582.99**	610.95	43.52	2.88	584.18	**587.00**	3.26
C3	390.48	98.58	642.40	154.60	562.90	698.20	82.70	453.59	558.75	644.42	91.42	2.86	**549.63**	**552.52**	3.96
C4	371.99	76.20	580.40	158.26	521.90	630.40	101.80	180.66	**507.38**	527.13	25.72	2.97	520.13	**524.15**	2.88
C5	464.97	86.69	754.60	161.78	683.10	822.60	119.30	316.68	**667.53**	687.64	20.16	3.35	668.65	**685.92**	13.92
C6	360.73	208.76	951.60	163.19	854.60	1010.60	146.40	989.14	**822.85**	900.74	82.23	2.92	841.48	**846.83**	2.40
C7	354.15	82.90	577.40	154.04	529.20	572.50	29.70	351.53	**521.89**	**540.31**	10.63	3.37	533.92	541.88	5.94
C8	375.52	60.75	540.60	156.01	**471.00**	522.80	29.80	115.40	476.66	489.45	10.30	3.46	475.96	**478.39**	2.84
C9	355.29	98.60	608.70	162.48	551.10	642.70	77.60	346.79	**535.87**	572.36	29.68	3.44	545.18	**558.54**	15.13
C10	431.18	75.80	679.30	139.39	608.90	653.00	35.60	341.12	**590.26**	617.35	26.91	2.73	611.03	**614.59**	3.74
Avg	393.75	108.34	713.24	153.50	636.20	758.87	104.38	448.35	**622.12**	659.18	40.12	3.10	629.93	**636.37**	5.76
D1	492.09	5.27	1321.80	201.34	1278.20	1498.80	199.00	2248.63	**1149.70**	**1202.98**	66.99	7.99	1193.21	1215.79	11.11
D2	384.68	4.21	892.70	237.53	746.90	914.30	97.90	1017.40	690.21	761.25	50.31	7.24	**679.58**	**695.99**	11.49
D3	380.05	4.21	819.40	218.80	678.60	817.80	80.90	1097.04	**643.23**	680.90	36.63	8.50	644.16	**650.22**	3.65
D4	418.94	4.21	877.40	209.09	809.70	1073.10	197.50	1329.47	798.99	828.36	23.85	7.22	**795.15**	**827.28**	12.32
D5	415.81	5.27	872.10	211.62	777.00	924.90	120.80	1231.71	**688.53**	741.15	35.31	7.72	693.83	**702.68**	3.91
D6	392.08	5.27	835.20	217.68	768.60	886.60	97.40	1011.85	**712.15**	764.51	31.68	7.91	731.71	**743.64**	5.97
D7	372.49	6.32	706.30	236.68	600.10	680.40	31.60	1121.08	595.22	619.66	20.47	7.83	**586.10**	**597.25**	5.68
D8	409.35	4.21	811.40	210.92	715.50	775.80	31.10	975.76	666.09	712.81	31.97	8.10	**658.49**	**669.83**	7.02
D9	385.89	6.32	842.70	219.65	741.00	818.20	46.50	661.42	**671.23**	726.78	28.43	9.15	689.83	**710.32**	10.95
D10	485.63	3.16	1306.60	243.72	1424.60	1867.70	258.60	1733.73	1239.79	1329.02	88.15	6.87	**1189.32**	**1280.92**	62.72
Avg	413.70	4.84	928.56	220.70	854.02	1025.76	116.13	1242.81	**785.51**	836.74	41.38	7.85	786.14	**809.39**	13.48

Table 7: (Continued) Extensive computational results comparing the performance of local search-based method from the literature, and our proposed BRKGA.

Inst.	LB$^+$	MMB2014		LGS2019				KBA2019				BRKGA			
		Time (sec)	Cost	Time (sec)	Best	Avg	SD	Time (sec)	Best	Avg	SD	Time (sec)	Best	Avg	SD
E1	430.36	17.90	1604.90	–	–	–	–	4201.41	1337.87	1466.10	121.41	17.19	**1327.72**	**1340.36**	8.44
E2	444.88	10.53	1101.90	–	–	–	–	3860.38	858.38	924.91	51.62	17.14	**829.79**	**865.05**	45.91
E3	454.27	14.74	986.40	–	–	–	–	3246.19	795.72	883.85	43.01	16.70	**789.56**	**806.53**	17.56
E4	412.08	20.01	871.00	–	–	–	–	3710.59	732.28	791.98	40.14	14.77	**723.87**	**728.96**	3.45
E5	416.62	20.01	1018.00	–	–	–	–	4643.35	791.18	857.18	34.80	17.03	**780.04**	**817.13**	34.10
E6	416.60	20.01	1003.00	–	–	–	–	4216.42	831.76	881.96	39.75	18.28	**779.82**	**793.68**	8.15
E7	389.57	21.06	921.10	–	–	–	–	3565.23	744.15	813.42	42.63	18.02	**705.79**	**715.46**	7.89
E8	433.89	20.01	884.60	–	–	–	–	3889.80	745.18	808.44	30.05	17.20	**733.90**	**750.59**	6.90
E9	446.49	18.95	1131.70	–	–	–	–	3575.06	926.38	1011.13	49.52	16.38	**893.35**	**916.56**	14.53
E10	455.07	11.58	1053.60	–	–	–	–	4197.26	874.51	931.44	35.06	15.99	**822.85**	**841.57**	9.28
Avg	429.98	17.48	1057.62	–	–	–	–	3910.57	863.74	937.04	48.80	16.87	**838.67**	**857.59**	15.62
F1	548.88	936.12	1721.40	–	–	–	–	6765.23	2557.06	2681.11	63.85	124.35	**1311.10**	**1351.20**	22.09
F2	543.32	957.18	1763.80	–	–	–	–	4389.44	2544.32	2628.75	30.43	121.74	**1298.31**	**1337.41**	20.01
F3	547.64	914.00	1549.60	–	–	–	–	5325.04	2433.87	2473.77	16.10	116.49	**1215.96**	**1272.23**	56.87
F4	531.84	1391.01	1420.40	–	–	–	–	2650.15	2116.80	2125.39	3.77	135.95	**1100.66**	**1134.66**	35.94
F5	538.14	1205.69	1701.90	–	–	–	–	2502.22	2669.68	2697.29	6.50	119.79	**1298.55**	**1331.09**	21.10
F6	518.47	880.31	1639.70	–	–	–	–	9663.74	2455.95	2504.51	30.34	109.64	**1292.52**	**1368.41**	83.02
F7	512.98	1362.58	1384.30	–	–	–	–	2337.04	2253.09	2253.09	0.00	120.48	**1084.57**	**1125.37**	17.55
F8	536.15	972.97	1544.60	–	–	–	–	5802.21	2321.96	2374.64	26.50	107.70	**1123.22**	**1140.42**	9.52
F9	543.16	1729.03	1572.90	–	–	–	–	4297.17	2510.01	2529.09	8.81	125.42	**1263.19**	**1344.62**	84.05
F10	546.84	1396.28	1581.00	–	–	–	–	2593.33	2686.98	2709.23	7.75	119.55	**1383.08**	**1419.76**	15.41
Avg	536.74	1174.52	1587.96	–	–	–	–	4632.56	2454.97	2497.69	19.40	120.11	**1237.12**	**1282.52**	36.56
G1	612.37	7581.60	2248.00	–	–	–	–	3569.71	5089.92	5089.92	0.00	439.40	**1744.14**	**1824.34**	96.54
G2	605.84	7581.60	2316.10	–	–	–	–	3926.59	10299.50	10299.50	0.00	519.49	**1709.70**	**1799.78**	78.71
G3	614.20	7525.79	1885.30	–	–	–	–	3477.90	3152.55	3152.55	0.00	461.61	**1464.69**	**1511.86**	50.36
G4	604.30	7581.60	2023.20	–	–	–	–	3858.37	3277.34	3277.34	0.00	529.01	**1508.94**	**1569.01**	76.22
G5	633.66	7581.60	2247.60	–	–	–	–	3502.11	3584.11	3584.11	0.00	466.62	**1652.88**	**1681.01**	17.35
G6	621.46	7581.60	2144.40	–	–	–	–	3939.74	3498.75	3498.75	0.00	570.54	**1681.64**	**1719.18**	52.60
G7	602.42	7301.50	1971.50	–	–	–	–	3573.50	3214.17	3214.17	0.00	522.37	**1536.00**	**1604.96**	35.02
G8	618.74	7581.60	1987.40	–	–	–	–	3459.40	3241.68	3241.68	0.00	531.73	**1498.38**	**1535.90**	53.54
G9	662.70	7395.22	2415.50	–	–	–	–	3482.24	3673.47	3673.47	0.00	446.57	**1850.07**	**1976.27**	111.56
G10	633.76	7374.16	2373.40	–	–	–	–	3436.26	3751.71	3751.71	0.00	482.81	**1785.37**	**1868.56**	102.21
Avg	620.94	7508.63	2161.24	–	–	–	–	3622.58	4278.32	4278.32	0.00	497.01	**1643.18**	**1709.09**	67.41
Avg time (sec.)		4027.59		147.27				1993.37				92.29			
# Best		11		19				33				62			
# Best avg		–		2				8				70			

Observe that not all authors provide solutions to all the instances for the Mankowska et al. (2014) benchmark dataset. The missing results are marked with a dash. Lasfargeas et al. (2019) argued that their solution method could not produce results for instances larger than 75 patients unless some of the problem's constraints are relaxed. We conjecture that this issue is related to the failure of their initial solution heuristic to produce feasible solutions. Kummer et al. (2019) justified that instances with 200 and 300 patients cannot be solved with the fix-and-optimize matheuristic due to the memory required to keep the MIP model. For instances larger than 200 patients, we can expect only the AVNS of Mankowska et al. (2014) and our BRKGA-MP-IPR to produce feasible solutions, with an advantage for our metaheuristic algorithm both terms of solution value and processing time.

Mankowska et al. (2014) provided optimal solutions for subset A, and all four approaches achieved these solutions. In subset B, Lasfargeas et al. (2019) individually produced 4 out of 10 best solutions, and Kummer et al. (2019) produced the best values for the entire instance subset. Our metaheuristic algorithm also provides the four best results for this subset. For the instances with 25 patients, our BRKGA is, on average, the fastest method. It is followed by the VNS from Lasfargeas et al. (2019) and by the *matheuristic* from Kummer et al. (2019). Our metaheuristic algorithm produces almost all the best results for subset D and greater. Additionally, the metaheuristic algorithm of Mankowska et al. (2014) was the fastest method for solving up to 75 patients, but the solutions produced are the worst among all the compared methods. For subsets F and G, the largest test cases, the BRKGA outperformed all the others in both solution quality and runtime.

5.5. Comparison with the standard BRKGA

As we mentioned in Section 4, Kummer et al. (2020) tested a very comparable but more limited version of our heuristic consisting of a single population variant with the standard two-parent mating with the explicit setting of bias weight towards the elite parent. Furthermore, their proposed decoder has no extra components, namely the *convex hull* strategy and the *workload balancing heuristic* we proposed in this paper. Their algorithm also only selects candidate vehicles that strictly improve the cost during the greedy construction phase of the decoder. Despite these differences, both have so many similarities that it makes sense to run a dedicated comparison between the two methods.

Table 8 presents results for each instance from Mankowska et al. (2014) dataset (column "Instance"). Under "KBA2020", we present the average solve time in seconds, the value of the best solution, and the average and standard deviation of solution values produced by the algorithm of Kummer et al. (2020). We run their genetic algorithm 20 times, using the numbers between 1 and 20 as seeds. We present results for two variations of our BRKGA. The variation "BRKGA (SD)" uses the intensification components proposed by Andrade et al. (2021) but uses the simpler decoding (SD) algorithm proposed by Kummer et al. (2020). The variation "BRKGA (FD)" uses the full decoding (FD) scheme described in Subsection 4.4. For both variations, we report the average number of generations evolved by the algorithms (Gens (avg)), the average solve times (Time (sec)), the best solution value among the 20 runs of each instance (Best), and the average (Avg) and standard deviation (SD) of solution values per instance, respectively. At the bottom, we present the average solve time of each method (Avg time (sec)), the number of instances in which each method produced the best solution value (# Best), and the number of instances in which each method produced the best average solution value (# Best avg).

Table 8: Comparison of Kummer et al. (2020) genetic algorithm and our proposed BRKGA.

Instance	KBA2020				BRKGA (SD)					BRKGA (FD)				
	Time (sec)	Best	Avg	SD	Gens (avg)	Time (sec)	Best	Avg	SD	Gens (avg)	Time (sec)	Best	Avg	SD
A1	0.77	226.98	226.98	0.00	11.05	0.20	226.98	226.98	0.00	11.25	0.20	**218.20**	**218.20**	0.00
A2	0.77	**246.63**	**246.63**	0.00	9.00	0.15	**246.63**	**246.63**	0.00	10.40	0.17	**246.63**	**246.63**	0.00
A3	0.77	**305.86**	**305.86**	0.00	9.05	0.18	**305.86**	**305.86**	0.00	9.40	0.16	**305.86**	**305.86**	0.00
A4	0.75	**186.90**	**186.90**	0.00	9.75	0.17	**186.90**	**186.90**	0.00	9.75	0.16	**186.90**	**186.90**	0.00
A5	0.82	191.97	191.97	0.00	8.45	0.14	191.97	191.98	0.01	10.10	0.18	**189.54**	**189.55**	0.01
A6	0.79	**200.13**	**200.13**	0.00	8.30	0.12	**200.13**	**200.15**	0.01	8.70	0.15	**200.13**	**200.13**	0.01
A7	0.73	**225.37**	**225.37**	0.00	10.90	0.19	**225.37**	**225.37**	0.00	10.65	0.18	**225.37**	**225.37**	0.00
A8	0.73	**232.05**	**232.05**	0.00	10.05	0.17	**232.05**	**232.05**	0.00	10.00	0.16	**232.05**	**232.05**	0.00
A9	0.83	234.21	234.21	0.00	10.60	0.17	234.21	234.37	0.70	12.20	0.20	**222.30**	**223.75**	3.57
A10	0.73	**225.01**	**225.01**	0.00	7.35	0.12	**225.01**	**225.01**	0.00	7.55	0.13	**225.01**	**225.01**	0.00
Avg	0.77	227.51	227.51	0.00	9.45	0.16	227.51	227.53	0.07	10.00	0.17	**225.20**	**225.35**	0.36
B1	2.18	**428.10**	**428.32**	0.25	42.65	0.74	**428.10**	428.97	1.30	42.80	0.78	**428.10**	428.53	0.15
B2	1.99	483.63	485.31	1.54	47.30	0.82	483.63	484.26	1.04	48.35	0.87	**476.05**	**476.92**	2.34
B3	2.23	**402.80**	**402.80**	0.00	51.55	0.87	**402.80**	408.21	4.41	55.05	0.99	**402.80**	409.29	4.78
B4	2.09	**420.29**	432.55	3.37	60.00	1.02	432.16	439.93	5.85	62.35	1.13	422.06	**430.46**	8.81
B5	1.94	372.16	**374.65**	3.09	51.30	0.89	372.16	378.57	2.45	54.95	0.98	**369.44**	375.15	3.88
B6	2.29	471.00	471.95	1.43	55.45	0.98	471.17	474.22	1.99	66.40	1.22	**470.59**	**470.70**	0.16
B7	2.36	**328.67**	**328.67**	0.00	49.00	0.86	**328.67**	**328.71**	0.19	51.85	0.93	**328.67**	**328.67**	0.00
B8	2.35	359.70	359.70	0.00	34.15	0.59	359.70	359.70	0.00	38.00	0.70	**357.68**	**359.40**	0.66
B9	2.49	**402.67**	404.25	1.06	50.15	0.88	**402.67**	**404.10**	0.54	48.65	0.91	404.11	404.29	0.23
B10	2.21	**469.58**	**469.58**	0.00	45.15	0.79	**469.58**	**469.58**	0.00	46.40	0.85	**469.58**	**469.58**	0.00
Avg	2.21	413.86	415.78	1.07	48.67	0.84	415.06	417.63	1.78	51.48	0.93	**412.91**	**415.30**	2.10
C1	7.08	**965.15**	977.56	14.39	129.80	2.96	966.96	**972.56**	6.03	131.00	3.09	969.11	973.87	3.52
C2	7.39	583.39	590.45	8.53	121.20	2.84	**582.22**	588.15	3.04	120.20	2.88	584.18	**587.00**	3.26
C3	7.14	**548.79**	559.53	6.44	117.30	2.72	553.05	559.33	5.60	121.15	2.86	549.63	**552.52**	3.96
C4	6.74	**519.91**	531.91	7.38	112.45	2.57	520.97	528.59	5.25	126.55	2.97	520.13	**524.15**	2.88
C5	6.01	678.49	698.14	19.54	137.80	3.02	678.49	698.34	14.10	148.90	3.35	**668.65**	**685.92**	13.92
C6	6.99	**840.69**	**845.30**	4.30	121.90	2.81	842.99	846.66	1.97	123.30	2.92	841.48	846.83	2.40
C7	8.28	534.01	540.42	5.33	133.30	3.17	**528.56**	**537.72**	3.14	136.55	3.37	533.92	541.88	5.94
C8	6.82	474.55	479.75	3.60	127.75	2.87	**473.82**	480.68	2.66	148.60	3.46	475.96	**478.39**	2.84
C9	8.40	**534.30**	551.72	9.25	143.60	3.44	536.99	**548.04**	6.42	139.25	3.44	545.18	558.54	15.13
C10	6.61	611.25	618.83	4.36	115.20	2.63	611.67	617.06	3.36	116.85	2.73	**611.03**	**614.59**	3.74
Avg	7.14	**629.05**	639.36	8.31	126.03	2.90	629.57	637.71	5.16	131.24	3.10	629.93	**636.37**	5.76
D1	17.67	**1186.20**	1208.19	13.98	187.25	6.03	1199.65	**1206.88**	4.90	235.60	7.99	1193.21	1215.79	11.11
D2	14.38	693.28	719.52	15.00	222.25	6.74	695.81	713.60	12.03	231.85	7.24	**679.58**	**695.99**	11.49
D3	18.48	**635.67**	650.29	7.63	211.60	7.14	638.82	**644.33**	3.07	238.20	8.50	644.16	650.22	3.65
D4	14.46	809.45	840.61	13.72	187.95	5.74	811.33	841.19	10.31	227.25	7.22	**795.15**	**827.28**	12.32
D5	17.81	**691.50**	703.23	12.10	211.90	6.90	692.54	**699.64**	5.36	228.20	7.72	693.83	702.68	3.91
D6	19.30	733.67	744.99	7.32	195.75	6.75	**728.94**	743.78	6.16	221.30	7.91	731.71	**743.64**	5.97
D7	20.51	590.64	603.65	6.42	209.70	7.44	588.73	606.52	14.61	211.50	7.83	**586.10**	**597.25**	5.68
D8	18.66	661.78	681.10	12.47	196.65	6.61	660.33	670.59	6.85	230.85	8.10	**658.49**	**669.83**	7.02
D9	17.71	704.63	722.09	10.35	238.30	7.78	706.44	719.93	7.67	271.60	9.15	**689.83**	**710.32**	10.95
D10	13.37	1208.71	1294.66	66.45	200.80	5.97	1236.76	1289.61	54.11	222.05	6.87	**1189.32**	**1280.92**	62.72
Avg	17.23	791.55	816.83	16.54	206.22	6.71	795.94	813.61	12.51	231.84	7.85	**786.14**	**809.39**	13.48
E1	33.89	1331.49	1352.33	10.75	277.00	12.91	**1317.90**	**1337.99**	13.61	347.55	17.19	1327.72	1340.36	8.44
E2	33.01	848.08	869.45	15.00	327.80	15.08	837.54	**857.31**	11.91	357.25	17.14	**829.79**	865.05	45.91
E3	32.17	788.03	815.31	15.16	307.55	13.85	**787.94**	**805.38**	10.37	351.10	16.70	789.56	806.53	17.56
E4	34.99	**711.19**	731.24	11.75	276.55	13.03	713.78	**725.05**	5.99	292.25	14.77	723.87	728.96	3.45
E5	32.18	781.50	806.81	13.73	319.35	14.51	782.54	**799.66**	11.26	354.15	17.03	**780.04**	817.13	34.10
E6	34.38	788.08	803.28	7.86	328.20	15.40	790.51	799.31	6.18	371.35	18.28	**779.82**	**793.68**	8.15
E7	37.90	711.11	731.83	9.38	285.95	14.40	712.56	728.04	7.11	343.10	18.02	**705.79**	**715.46**	7.89
E8	31.83	748.48	761.06	7.40	284.55	12.80	737.01	762.65	16.46	366.10	17.20	**733.90**	**750.59**	6.90
E9	30.96	921.78	950.89	13.90	324.55	14.47	901.83	939.67	17.90	352.20	16.38	**893.35**	**916.56**	14.53
E10	33.61	825.24	847.64	14.48	294.65	13.65	**822.24**	**840.13**	8.25	327.45	15.99	822.85	841.57	9.28
Avg	33.49	845.50	866.98	11.94	302.62	14.01	840.39	859.52	10.90	346.25	16.87	**838.67**	**857.59**	15.62

Table 8: (Continued) Comparison of Kummer et al. (2020) genetic algorithm and our proposed BRKGA.

Instance	KBA2020				BRKGA (SD)					BRKGA (FD)				
	Time (sec)	Best	Avg	SD	Gens (avg)	Time (sec)	Best	Avg	SD	Gens (avg)	Time (sec)	Best	Avg	SD
F1	133.86	1372.69	1421.76	17.34	686.70	91.38	1369.44	1395.91	16.45	886.80	124.35	**1311.10**	**1351.20**	22.09
F2	143.70	1336.33	1383.55	23.01	855.10	117.93	1301.28	1347.48	22.79	833.60	121.74	**1298.31**	**1337.41**	20.01
F3	137.42	1263.39	1288.66	14.08	668.75	90.42	1230.40	**1248.36**	10.22	814.70	116.49	**1215.96**	1272.23	56.87
F4	169.13	1124.24	1145.53	10.23	653.15	103.74	1111.23	**1127.86**	10.44	811.45	135.95	**1100.66**	1134.66	35.94
F5	153.59	1316.54	1361.64	20.75	720.25	105.20	**1298.00**	**1323.14**	16.49	770.95	119.79	1298.55	1331.09	21.10
F6	128.41	1322.89	1369.38	25.66	755.45	96.41	1307.26	**1346.67**	38.96	807.15	109.64	**1292.52**	1368.41	83.02
F7	164.91	1131.27	1163.97	17.53	645.15	99.49	1110.86	1133.37	14.68	736.90	120.48	**1084.57**	**1125.37**	17.55
F8	145.91	1132.77	1165.67	16.72	695.65	96.86	**1118.15**	**1139.21**	10.36	726.10	107.70	1123.22	1140.42	9.52
F9	155.50	1293.78	1347.78	22.91	657.55	97.43	1264.42	**1305.77**	21.42	802.15	125.42	**1263.19**	1344.62	84.05
F10	149.67	1418.53	1446.56	15.68	737.80	105.59	1392.71	1421.59	14.29	787.05	119.55	**1383.08**	**1419.76**	15.41
Avg	148.21	1271.24	1309.45	18.39	707.56	100.44	1250.38	**1278.94**	17.61	797.69	120.11	**1237.12**	1282.52	36.56
G1	348.03	1778.54	1851.25	32.13	1224.20	413.30	1758.18	**1787.46**	16.29	1211.20	439.40	**1744.14**	1824.34	96.54
G2	388.24	1824.74	1899.33	34.71	1404.65	508.39	**1708.75**	**1780.34**	28.55	1316.45	519.49	1709.70	1799.78	78.71
G3	340.51	1514.23	1546.44	13.92	1131.70	381.37	**1453.53**	**1485.42**	19.77	1294.70	461.61	1464.69	1511.86	50.36
G4	397.61	1564.42	1599.35	23.92	1113.75	417.95	**1491.44**	**1541.94**	24.76	1313.95	529.01	1508.94	1569.01	76.22
G5	328.49	1694.50	1750.03	28.07	1092.55	354.84	**1630.25**	**1671.12**	23.07	1358.10	466.62	1652.88	1681.01	17.35
G6	420.99	1714.38	1779.29	21.68	1161.00	454.69	**1671.76**	**1704.90**	15.71	1353.85	570.54	1681.64	1719.18	52.60
G7	354.21	1640.07	1677.66	22.99	1203.90	411.84	1568.52	1614.15	19.17	1430.45	522.37	**1536.00**	**1604.96**	35.02
G8	348.98	1547.63	1582.71	20.05	1244.15	418.63	1499.45	**1522.14**	15.26	1486.25	531.73	**1498.38**	1535.90	53.54
G9	357.05	1942.21	1974.16	19.69	1366.90	472.92	**1836.61**	**1895.25**	62.12	1190.35	446.57	1850.07	1976.27	111.56
G10	317.27	1872.08	1931.99	25.31	1283.50	403.33	1821.23	**1863.12**	55.45	1437.50	482.81	**1785.37**	1868.56	102.21
Avg	360.14	1709.28	1759.22	24.25	1222.63	423.72	1643.97	**1686.59**	28.02	1339.28	497.01	**1643.18**	1709.09	67.41
Avg time (sec)		81.31					78.40					92.29		
# Best		22					27					44		
# Best avg		13					37					38		

The results indicate a mixed performance among BRKGA (SD) and BRKGA (FD). Very few instances presented better solution via the genetic algorithm of Kummer et al. (2020) (B4, C1, C6, D1, D3, D5, and E4), although the largest difference was 1.36% between the best solution our algorithm produced to the test case D3 and KBA2020. Furthermore, we credit this difference to the stopping criteria of the solution methods. While KBA2020 used a fixed number of generations on their tests, our algorithm calculates its stopping criterion according to the instance sizes. For instance D1, KBA2020 evolved a solution by 1823 generations, while our BRKGA (FD) achieved, on average, 235.60 generations.

In general, our BRKGA produced the best solution up to 44 out of 70 instances from the Mankowska et al. (2014) dataset, compared to the best 22 results from Kummer et al. (2020). (# Best at the bottom of the table). Furthermore, the BRKGA (SD) was the fastest among the three approaches, which indicates that a clever stopping criterion can leverage short computational times, even in the presence of potentially expensive components such as the multi-population and the IPR heuristic. Note also that BRKGA (FD) is, on average, slightly slower than the other two algorithms. This result comes from the larger effort by BRKGA (FD) to converge the solution, mainly due to the increased number of possible solutions that it can achieve. This increased effort can be further verified in the average number of generations evolved by BRKGA (FD) for subset G (1339.28 versus 1222.63 generations by BRKGA (SD)).

5.6. Component analysis for the new genetic algorithm

Although we already verified the superiority of our new BRKGA in producing better solution values to the HHCRSP, it is still interesting to further analyze by how much each of the additional intensification components proposed by Andrade et al. (2021) effectively contributes in producing new solution values. With that in mind, we dissected our algorithm BRKGA (FD) in four variants as follows. Note that we used the best configuration of *irace* from Subsection 5.3 among all these four algorithms. Thus some parameters were just ignored when their relative components were explicit turned off.

- BRKGA-MP (FD). Uses the decoding algorithm from Subsection 4.4, but only the multi-parent mating component of Andrade et al. (2021);
- BRKGA-MP-MI (FD). Similar to BRKGA-MP (FD), but also uses the multiple islands to evolve independent populations;
- BRKGA-MP-IPR (FD). Similar to BRKGA-MP (FD), but also uses the *implicit path-relinking* of Andrade et al. (2021);
- BRKGA-MP-MI-IPR (FD). Uses all the intensification components proposed by Andrade et al. (2021), plus the *full decoding* scheme of Subsection 4.4. This is the same as "BRKGA (FD)" we mention in Subsections 5.4 and 5.5.

Table 9 lists the extensive computational results for all the four combinations of the intensification components tested. For each approach, the table indicates the average solve time in seconds (Time (sec)), the best solution produced per instance (Best), and the average solution value (Avg). We used the same testing protocol as our other experiments, so we solved each instance 20 times by each approach, using the numbers 1 to 20 as seeds. At the bottom of the table, we summarize some data regarding the whole experiment. We present the average solve times in second (Avg time (sec)), the number of instances in which each approach produced the best solution values (# Best), and the number of times which each algorithm produced the best average solution values (# Best avg).

Table 9: Results of combined intensification components from the literature.

Instance	BRKGA-MP (FD)			BRKGA-MP-MI (FD)			BRKGA-MP-IPR (FD)			BRKGA-MP-MI-IPR (FD)		
	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg
A1	0.11	**218.20**	218.33	0.20	**218.20**	**218.20**	0.10	**218.20**	218.33	0.20	**218.20**	**218.20**
A2	0.10	**246.63**	**246.63**	0.18	**246.63**	**246.63**	0.10	**246.63**	**246.63**	0.17	**246.63**	**246.63**
A3	0.10	**305.86**	**305.86**	0.15	**305.86**	**305.86**	0.10	**305.86**	**305.86**	0.16	**305.86**	**305.86**
A4	0.10	**186.90**	**186.90**	0.16	**186.90**	**186.90**	0.10	**186.90**	**186.90**	0.16	**186.90**	**186.90**
A5	0.10	**189.54**	**189.55**	0.17	**189.54**	**189.55**	0.10	**189.54**	**189.55**	0.18	**189.54**	**189.55**
A6	0.10	**200.13**	**200.14**	0.15	**200.13**	**200.13**	0.10	**200.13**	**200.14**	0.15	**200.13**	**200.13**
A7	0.10	**225.37**	**225.37**	0.18	**225.37**	**225.37**	0.10	**225.37**	**225.37**	0.18	**225.37**	**225.37**
A8	0.10	**232.05**	**232.05**	0.17	**232.05**	**232.05**	0.10	**232.05**	**232.05**	0.16	**232.05**	**232.05**
A9	0.14	**222.30**	**223.13**	0.21	**222.30**	223.75	0.13	**222.30**	**223.13**	0.20	**222.30**	223.75
A10	0.09	**225.01**	**225.01**	0.12	**225.01**	**225.01**	0.10	**225.01**	**225.01**	0.13	**225.01**	**225.01**
Avg	0.10	**225.20**	**225.30**	0.17	**225.20**	**225.35**	0.10	**225.20**	**225.30**	0.17	**225.20**	**225.35**

Table 9: (Continued) Results of combined intensification components from the literature.

Instance	BRKGA-MP (FD)			BRKGA-MP-MI (FD)			BRKGA-MP-IPR (FD)			BRKGA-MP-MI-IPR (FD)		
	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg
B1	0.40	**428.10**	429.03	0.76	**428.10**	**428.53**	0.40	**428.10**	429.03	0.78	**428.10**	**428.53**
B2	0.45	**476.05**	478.53	0.86	**476.05**	477.24	0.45	**476.05**	478.34	0.87	**476.05**	**476.92**
B3	0.47	**402.80**	412.30	0.95	**402.80**	410.15	0.49	**402.80**	412.49	0.99	**402.80**	**409.29**
B4	0.54	**422.06**	434.86	1.12	**422.06**	**429.54**	0.54	**422.06**	434.06	1.13	**422.06**	430.46
B5	0.51	370.49	376.17	0.98	370.49	**374.63**	0.50	**369.44**	376.43	0.98	**369.44**	375.15
B6	0.59	**470.59**	472.26	1.19	**470.59**	470.80	0.59	**470.59**	472.25	1.22	**470.59**	**470.70**
B7	0.48	**328.67**	328.86	0.96	**328.67**	**328.67**	0.48	**328.67**	328.88	0.93	**328.67**	**328.67**
B8	0.32	358.69	359.60	0.69	358.69	**359.35**	0.33	358.69	359.60	0.70	**357.68**	359.40
B9	0.46	**404.11**	404.76	0.89	**404.11**	**404.29**	0.48	**404.11**	404.74	0.91	**404.11**	**404.29**
B10	0.46	**469.58**	**469.61**	0.83	**469.58**	**469.58**	0.47	**469.58**	**469.61**	0.85	**469.58**	**469.58**
Avg	0.47	413.11	416.60	0.92	413.11	**415.28**	0.47	413.01	416.54	0.93	**412.91**	**415.30**
C1	1.45	970.36	980.74	3.24	970.36	**972.34**	1.42	**969.01**	980.20	3.09	969.11	973.87
C2	1.48	584.25	589.52	2.83	584.25	587.42	1.49	584.25	589.73	2.88	**584.18**	**587.00**
C3	1.48	**549.35**	554.66	2.90	**549.35**	**552.26**	1.46	549.63	555.08	2.86	549.63	552.52
C4	1.46	**519.91**	527.11	2.96	**519.91**	524.83	1.54	**519.91**	526.49	2.97	520.13	**524.15**
C5	1.60	674.69	705.11	3.09	674.69	688.88	1.61	679.88	703.80	3.35	**668.65**	**685.92**
C6	1.44	843.73	851.38	2.72	843.73	847.77	1.53	845.82	851.77	2.92	**841.48**	**846.83**
C7	1.66	**533.74**	544.14	3.49	**533.74**	**539.72**	1.69	538.03	544.46	3.37	533.92	541.88
C8	1.69	**475.81**	478.77	3.42	**475.81**	478.44	1.63	476.20	480.19	3.46	475.96	**478.39**
C9	1.91	538.04	**558.05**	3.33	538.04	562.49	1.75	**536.88**	561.61	3.44	545.18	558.54
C10	1.41	**611.03**	617.76	2.77	**611.03**	615.05	1.50	**611.03**	617.49	2.73	**611.03**	**614.59**
Avg	1.56	630.09	640.72	3.07	630.09	636.92	1.56	631.06	641.08	3.10	**629.93**	**636.37**
D1	3.94	1198.16	1218.58	8.18	1198.16	**1210.86**	3.96	1199.55	1219.44	7.99	**1193.21**	1215.79
D2	3.34	685.75	705.39	6.96	685.75	697.31	3.66	686.24	709.32	7.24	**679.58**	**695.99**
D3	3.87	**638.32**	652.23	8.03	**638.32**	**650.10**	4.07	643.77	651.88	8.50	644.16	650.22
D4	3.55	802.84	834.41	6.62	802.84	829.25	3.39	805.58	836.03	7.22	**795.15**	**827.28**
D5	3.76	693.17	708.35	7.34	693.17	**702.23**	3.70	**692.58**	707.27	7.72	693.83	702.68
D6	4.04	735.43	748.01	7.87	735.43	**743.51**	3.85	737.07	749.53	7.91	**731.71**	743.64
D7	3.83	590.19	602.78	8.42	590.19	**596.59**	3.76	591.90	604.78	7.83	**586.10**	597.25
D8	3.94	663.69	677.42	7.53	663.69	670.61	3.93	666.12	681.46	8.10	**658.49**	**669.83**
D9	4.56	700.52	720.21	8.83	700.52	715.10	4.25	693.62	722.10	9.15	**689.83**	**710.32**
D10	3.45	1231.59	1324.11	6.47	1231.59	1287.24	3.10	1236.28	1330.68	6.87	**1189.32**	**1280.92**
Avg	3.83	793.97	819.15	7.62	793.97	810.28	3.76	795.27	821.25	7.85	**786.14**	**809.39**
E1	7.54	1336.46	1351.90	16.90	1336.46	**1338.61**	8.58	1330.70	1348.62	17.19	**1327.72**	1340.36
E2	8.92	**818.69**	**853.75**	18.22	**818.69**	853.93	8.66	833.48	866.48	17.14	829.79	865.05
E3	8.55	**784.89**	808.47	18.08	**784.89**	**803.01**	8.28	793.33	815.41	16.70	789.56	806.53
E4	7.11	719.89	729.39	13.71	719.89	729.28	7.51	**713.59**	**728.82**	14.77	723.87	728.96
E5	8.83	787.01	810.93	17.26	787.01	**799.44**	8.95	781.20	809.80	17.03	**780.04**	817.13
E6	8.66	784.54	800.74	17.67	784.54	800.21	9.02	786.32	802.22	18.28	**779.82**	**793.68**
E7	8.10	706.81	725.04	16.28	706.81	716.00	8.73	**704.42**	720.59	18.02	705.79	**715.46**
E8	7.94	737.83	**750.62**	14.25	737.83	751.94	7.73	746.26	755.86	17.20	**733.90**	**750.59**
E9	8.27	907.58	937.97	16.42	907.58	921.63	9.53	916.17	935.03	16.38	**893.35**	**916.56**
E10	7.77	824.25	845.67	15.28	824.25	843.52	8.12	831.13	846.80	15.99	**822.85**	841.57
Avg	8.17	840.80	861.45	16.40	840.80	**855.76**	8.51	843.66	862.97	16.87	**838.67**	857.59
F1	56.36	1333.99	1367.22	121.40	1333.99	1357.88	63.80	1331.25	1372.64	124.35	**1311.10**	1351.20
F2	55.02	1323.80	1345.16	117.58	1323.80	**1337.45**	67.65	1308.09	1352.35	121.74	**1298.31**	1337.41
F3	54.04	1226.14	1260.75	113.02	1226.14	**1252.67**	61.93	1227.43	1258.71	116.49	**1215.96**	1272.23
F4	57.77	1108.51	1131.85	131.34	1108.51	**1129.72**	68.72	**1094.77**	1133.73	135.95	1100.66	1134.66
F5	53.86	1314.89	1343.67	121.37	1314.89	1332.29	60.71	1326.11	1345.59	119.79	**1298.55**	1331.09
F6	48.34	1311.24	**1348.44**	102.82	1311.24	1356.85	51.12	1310.66	1366.06	109.64	**1292.52**	1368.41
F7	54.66	1110.95	1137.43	113.36	1110.95	1133.37	65.54	1104.15	1134.54	120.48	**1084.57**	1125.37
F8	46.82	1127.27	1148.04	95.26	1127.27	1140.94	55.61	**1120.09**	1150.68	107.70	1123.22	1140.42
F9	55.43	1279.43	1315.39	120.59	1279.43	**1313.67**	66.08	1293.61	1320.42	125.42	**1263.19**	1344.62
F10	51.70	**1374.92**	1429.27	102.23	**1374.92**	1425.94	60.60	1386.55	1424.99	119.55	1383.08	1419.76
Avg	53.40	1251.11	1282.72	113.89	1251.11	**1278.08**	62.17	1250.27	1285.97	120.11	**1237.12**	1282.52

Table 9: (Continued) Results of combined intensification components from the literature.

Instance	BRKGA-MP (FD)			BRKGA-MP-MI (FD)			BRKGA-MP-IPR (FD)			BRKGA-MP-MI-IPR (FD)		
	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg	Time (sec)	Best	Avg
G1	215.48	1759.49	1797.37	465.45	1759.49	**1771.01**	265.46	1760.69	1813.14	439.40	**1744.14**	1824.34
G2	250.01	1736.98	1787.91	499.88	1736.98	**1773.96**	319.97	1738.21	1799.39	519.49	**1709.70**	1799.78
G3	189.77	1480.51	1510.20	457.79	1480.51	**1487.32**	253.78	1474.21	1510.04	461.61	**1464.69**	1511.86
G4	221.62	1513.00	1553.84	496.57	1513.00	**1548.89**	321.27	1511.15	1549.92	529.01	**1508.94**	1569.01
G5	199.71	1671.42	1701.86	423.41	1671.42	**1680.65**	252.90	1658.72	1710.80	466.62	**1652.88**	1681.01
G6	224.13	1700.53	1728.20	501.02	1700.53	**1716.80**	308.38	1698.21	1738.29	570.54	**1681.64**	1719.18
G7	207.47	1584.82	1612.97	451.29	1584.82	**1591.85**	302.10	1591.10	1606.23	522.37	**1536.00**	1604.96
G8	206.47	1515.08	1548.83	431.53	1515.08	**1530.88**	291.82	1508.53	1538.69	531.73	**1498.38**	1535.90
G9	258.00	**1844.29**	1890.54	552.71	**1844.29**	**1883.73**	272.46	1862.58	1943.51	446.57	1850.07	1976.27
G10	214.86	1828.34	1865.95	469.80	1828.34	**1850.58**	263.73	1790.54	1892.33	482.81	**1785.37**	1868.56
Avg	218.75	1663.45	1699.77	474.94	1663.45	**1683.57**	285.18	1659.39	1710.23	497.01	**1643.18**	1709.09
Avg time (sec)		40.89			88.15			51.68			92.29	
# Best		28			28			28			54	
# Best avg		14			41			11			38	

The first thing to notice is that we can sort the algorithms by their average solve times. The BRKGA-MP (FD) was the fastest approach, requiring on average 218.75 seconds to finish on the largest test cases with 300 patients (subset G). This algorithm is 60.7% faster compared to KBA2020 (c.f. Table 8). BRKGA-MP-IPR was the next fastest algorithm, which indicates a relatively small overhead associated with the IPR heuristic, even though the configuration used with the algorithms triggered the IPR quite frequently, between a large number of guide and base solutions (c.f. Subsection 5.3). It is easy to see that multi-population is the most expensive feature within all four intensification components. Compared to BRKGA-MP (FD), the BRKGA-MP-MI (FD) consumed 2.17 more time to finish, mostly because this algorithm had to evolve two isolated populations in parallel; thus, we expect an increase in algorithm runtime by a factor of two, plus some overhead due to the immigration process.

All the three algorithms discussed so far have similar performance, producing the best solution values for 28 out of 70 instances of the Mankowska et al. (2014) dataset–although the best solution by each algorithm refers to distinct instances. Compared to the other algorithms, the slowest yet best-performing BRKGA-MP-MI-IPR (FD) produced the best solutions to 54 out of 70 test cases, which is almost twice the number of best solutions provided by any other algorithm. We can credit its best performance to the IPR heuristic, which seems to perform much better in the presence of the multi-population component; thus, it allows the heuristic to trigger the *implicit path-relinking* between elite individuals from distinct islands. Again, it also has similar solve times to BRKGA-MP-MI (FD), mainly due to the additional overhead of evolving two isolated populations. Both BRKGA-MP-MI (FD) and BRKGA-MP-MI-IPR (FD) have a similar performance regarding # Best avg, but the first algorithm showed a better performance in producing the best average solution values for the subset G, which indicates better stability of results for BRKGA-MP-MI (FD). We think this can be related to the IPR because it is an intensification mechanism that could cause an early loss in population diversity, especially in larger test cases. Still, this would require additional testing and probably new AAC experiments to produce a tailed configuration for each of the four combinations of the intensification components.

As we have a substantial number of experiments, we applied a pairwise Wilcoxon signed-rank test to identify which of the proposed variations of our metaheuristic algorithm are statistically different. According to Figure 5, almost all pairs of algorithm variations behave differently, with two notable exceptions. First, the statistical test identified no difference between BRKGA-MP (FD) and BRKGA-MP-IPR (FD), which indicates that the IPR heuristic has a negligible impact in such a variant without the multiple populations. Secondly, the test indicated no statistical difference between BRKGA (FD) and BRKGA-MP-MI (FD). The statistical test used a confidence level o 5%, and the Bonferroni correction was applied. Roughly speaking, a Wilcoxon test works by comparing the distribution of the solution values by each algorithm, so counting which method produces most of the best solutions per instance has no importance. Nevertheless, the BRKGA (FD) variant still produced the largest number of "best solution" over all the studied variations, as presented in Table 9.

```
> with(AllResults, pairwise.wilcox.test(x=cost, g=algorithm, p.adj="bonf", paired=T))

Pairwise comparisons using Wilcoxon signed rank test

data:  cost and algorithm

                  BRKGA (FD) BRKGA (SD) BRKGA-MP (FD) BRKGA-MP-IPR (FD) BRKGA-MP-MI (FD)
BRKGA (SD)        1.8e-06    -          -             -                  -
BRKGA-MP (FD)     < 2e-16    2.9e-08    -             -                  -
BRKGA-MP-IPR (FD) < 2e-16    2.1e-06    1             -                  -
BRKGA-MP-MI (FD)  1          1.4e-07    < 2e-16       < 2e-16            -
KBA2020           < 2e-16    < 2e-16    < 2e-16       < 2e-16            < 2e-16
```

Fig. 5: Output of pairwise Wilcoxon signed rank test for paired samples.

6. Conclusion

In this paper, we propose a new multi-population multi-parent biased random-key genetic algorithm for solving the daily home health care problem. This problem consists of a vehicle routing problem with time-windows, in which the vehicles represent skilled caregivers, and the nodes represent patients requiring one or more service types. To enable such attendance, a matching of skilled caregivers to patient service type requests is required. Additional route inter-dependency constraints are imposed for patients requiring more than one service type. Our implementation, and the benchmark dataset are publicly available at `https://github.com/afkummer/brkga-mp-ipr-hhcrsp-2021`. This repository also contains additional logfiles regarding our automatic parameter configuration experiment, as well as supplementary material regarding our experiments.

We present a heuristic-powered decoder for the problem, and we use a state-of-art tool to perform the automatic parameter configuration of our proposed metaheuristic algorithm. We use a benchmark dataset from the literature to both configure and benchmark our algorithm. We run an extensive experiment where

we compare our algorithm with previously proposed methods. With respect to previous local search-based methods, we consistently find improvements up to 26.1% from previously published results. Compared to the similar work of Kummer et al. (2020), our algorithm finds improvements from around 0.4% up to 6.36% in the largest test cases.

There are a few possible directions for improvement of our algorithms. We would like to improve the heuristic decoder further improve its capability of exploring the problem's solution space. We also plan to run a more in-depth study of the benchmark dataset characteristics to automatically provide more accurate parameter settings for the metaheuristic through a combined algorithm configuration and selection framework. We are currently in contact with a manager that coordinates a pilot project to implement this type of service in a large Brazilian city. Our goal is to adapt our solution technique to solve a real problem the manager currently deals with every week. As a result of the longer planning period, we also target re-planning the problem due to uncertainties.

Acknowledgements

We would like to thank the reviewers for their valuable considerations. This research has the support of FAPERGS, project PqG 17/2551-0001201-1. The first author would like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES) for his doctoral scholarship. The contributions of Luciana S. Buriol and Mauricio G.C. Resende to this paper are not related to their roles at Amazon.

References

Ait Haddadene, S.R., Labadie, N., Prodhon, C., 2016. A GRASP $\times$ ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. *Expert Systems with Applications* 66, 274–294.

Andrade, C.E., Toso, R.F., Gonçalves, J.F., Resende, M.G., 2021. The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. *European Journal of Operational Research* 289, 1, 17–30.

Bredström, D., Rönnqvist, M., 2008. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research* 191, 1, 19–31.

Bresina, J.L., 1996. Heuristic-biased stochastic sampling. In *AAAI/IAAI, Vol. 1*, pp. 271–278.

Chaves, A.A., Lorena, L.A.N., Senne, E.L.F., Resende, M.G., 2016. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research* 67, 174–183.

Cissé, M., Yalçındağ, S., Kergosien, Y., Şahin, E., Lenté, C., Matta, A., 2017. Or problems related to home health care: A review of relevant routing and scheduling problems. *Operations Research for Health Care* 13, 1–22.

Decerle, J., Grunder, O., Hajjam El Hassani, A., Barakat, O., 2018. A memetic algorithm for a home health care routing and scheduling problem. *Operations Research for Health Care* 16, 59–71.

Desaulniers, G., Desrosiers, J., Solomon, M.M., 2006. *Column generation*, Vol. 5. Springer Science & Business Media.

Drexl, M., 2012. Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science* 46, 3, 297–316.

Eggensperger, K., Lindauer, M., Hutter, F., 2019. Pitfalls and best practices in algorithm configuration. *arXiv.* Available at https://arxiv.org/abs/1705.06058v3.

Eiben, A.E., Smith, J.E., et al., 2003. *Introduction to evolutionary computing*, Vol. 53. Springer.

Eveborn, P., Flisberg, P., Rönnqvist, M., 2006. LAPS CARE–an operational system for staff planning of home care. *European Journal of Operational Research* 171, 3, 962–976.

Fernandez, A., Gregory, G., Hindle, A., Lee, A., 1974. A model for community nursing in a rural county. *Journal of the Operational Research Society* 25, 2, 231–239.

Fikar, C., Hirsch, P., 2017. Home health care routing and scheduling: A review. *Computers & Operations Research* 77, 86–95.

Frifita, S., Masmoudi, M., Euchi, J., 2017. General variable neighborhood search for home healthcare routing and scheduling problem with time windows and synchronized visits. *Electronic Notes in Discrete Mathematics* 58, 63–70.

Gendreau, M., Potvin, J.Y., et al., 2010. *Handbook of metaheuristics*, Vol. 2. Springer.

Glover, F., 1997. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges, Springer US, Boston, MA, chapter 1. pp. 1–75.

Gonçalves, J.F., Resende, M.G., 2011. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics* 17, 5, 487–525.

Gonçalves, J.F., Resende, M.G., 2012. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research* 39, 2, 179–190.

Gonçalves, J.F., Resende, M.G., Mendes, J.J., 2011. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics* 17, 5, 467–486.

Grenouilleau, F., Legrain, A., Lahrichi, N., Rousseau, L.M., 2019. A set partitioning heuristic for the home health care routing and scheduling problem. *European Journal of Operational Research* 275, 1, 295–303.

Grieco, L., Utley, M., Crowe, S., 2020. Operational research applied to decisions in home health care: A systematic literature review. *Journal of the Operational Research Society* , 1–32.

Gutiérrez, E.V., Vidal, C.J., 2014. Home health care logistics management problems: A critical review of models and methods. *Revista Facultad de Ingeniería Universidad de Antioquia* 1, 160–175.

IBM, 2020. IBM ILOG CPLEX Optimization Studio 20.1.0.0. https://community.ibm.com/community/user/datascience/blogs/xavier-nodet1/2020/12/11/cos-201-is-available. Accessed on March 31, 2021.

Kummer, A.F., Buriol, L.S., de Araújo, O.C., 2020. A biased random key genetic algorithm applied to the VRPTW with skill requirements and synchronization constraints. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 717–724.

Kummer, A.F. Neto, Buriol, L.S., de Araújo, O.C.B., 2019. A matheuristic algorithm applied to the home health care problem. In *Eletronic Proceedings of the LI Brazilian Operational Research Symposium*, Vol. 2. Available at https://proceedings.science/sbpo-2019/papers/a-matheuristic-algorithm-applied-to-the-home-health-care-problem?lang=en.

Landers, S., Madigan, E., Leff, B., Rosati, R.J., McCann, B.A., Hornbake, R., MacMillan, R., Jones, K., Bowles, K., Dowding, D., Lee, T., Moorhead, T., Rodriguez, S., Breese, E., 2016. The future of home health care: A strategic framework for optimizing value. *Home Health Care Management and Practice* 28, 4, 262–278.

Lasfargeas, S., Gagné, C., Sioud, A., 2019. Solving the home health care problem with temporal precedence and synchronization. In *Bioinspired Heuristics for Optimization*. Springer, pp. 251–267.

Lucena, M.L., Andrade, C.E., Resende, M.G., Miyazawa, F.K., 2014. Some extensions of biased random-key genetic algorithms. In *Proceedings of the 46th Brazilian Symposium of Operational Research*, pp. 1–12.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M., 2016. The *irace* package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3, 43–58.

Mankowska, D.S., Meisel, F., Bierwirth, C., 2014. The home health care routing and scheduling problem with interdependent services. *Health Care Management Science* 17, 1, 15–30.

Maya Duque, P., Castro, M., Sörensen, K., Goos, P., 2015. Home care service planning. the case of Landelijke Thuiszorg. *European Journal of Operational Research* 243, 1, 292–301.

PassMark, S., 2021. Passmark cpu benchmark. https://www.cpubenchmark.net/.

Rasmussen, M.S., Justesen, T., Dohn, A., Larsen, J., 2012. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research* 219, 3, 598–610.

Resende, M.G., Ribeiro, C.C., Glover, F., Martí, R., 2010. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of metaheuristics*. Springer, pp. 87–107.

Resende, M.G.C., Ribeiro, C.C., 2016. Path-relinking, Springer New York, New York, NY, chapter 8. pp. 167–188.

Ruiz, E., Soto-Mendoza, V., Barbosa, A.E.R., Reyes, R., 2019. Solving the open vehicle routing problem with capacity and distance constraints with a biased random key genetic algorithm. *Computers & Industrial Engineering* 133, 207–219.

Toso, R.F., Resende, M.G., 2015. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software* 30, 1, 81–93.