Natural Computing (2023) 22:671–684
https://doi.org/10.1007/s11047-023-09968-9

Evolving ensembles of heuristics for the travelling salesman problem

Francisco J. Gil-Gala[1] · Marko Durasević[2] · María R. Sierra[1] · Ramiro Varela[1]

Accepted: 4 October 2023 / Published online: 25 October 2023

Abstract
The Travelling Salesman Problem (TSP) is a well-known optimisation problem that has been widely studied over the last century. As a result, a variety of exact and approximate algorithms have been proposed in the literature. When it comes to solving large instances in real-time, greedy algorithms guided by priority rules represent the most common approach, being the nearest neighbour (NN) heuristic one of the most popular rules. NN is quite general but it is too simple and so it may not be the best choice in some cases. Alternatively, we may design more sophisticated heuristics considering the particular features of families of instances. To do that, we have to consider problem attributes other than the proximity of the next city to build priority rules. However, this process may not be easy for humans and so it is often addressed by some learning procedure. In this regard, hyper-heuristics as Genetic Programming (GP) stands as one of the most popular approaches. Furthermore, a single heuristic, even being good in average, may not be good for a number of instances of a given set. For this reason, the use of ensembles of heuristics is often a good alternative, which raises the problem of building ensembles from a given set of heuristic rules. In this paper, we study the application of two kinds of ensembles to the TSP. Given a set of TSP instances having similar characteristics, we firstly exploit a GP to build a set of heuristics involving a number of problem attributes, and then we build ensembles combining these heuristics by means of a Genetic Algorithm (GA). The experimental study provided valuable insights into the construction and utilisation of single rules and ensembles. It clearly demonstrated that the performance of ensembles justifies the time invested when compared to using individual heuristics.

Keywords Travelling salesman problem · Hyper-heuristic · Greedy algorithm · Ensemble learning · Evolutionary algorithms

1 Introduction

The Travel Salesman Problem (TSP) is one of the most studied combinatorial optimisation problems with many real-world applications in a number of fields such as logistics and planning or communications (Punnen 2007; Mavrovouniotis et al. 2017). Consequently, a lot of exact and approximate algorithms were proposed in the literature over the last decades, such as the Lin-Kernighan heuristic proposed by Link and Kernighan (1973), Christofides heuristic proposed by Christofides (1976), or Genetic Local Search (GLS) proposed by Freisleben and Merz (1996), among others. These approaches have demonstrated good performance, but in situations when the problem instance is large, and the solutions must be completed in a limited time, or when not all information is available at the beginning, the best, if not the only one, solution is a greedy algorithm preferably guided by some kind of efficient heuristic rule.

This kind of heuristics may be designed manually by experts in the problem domain. This is the case of the simple and well-known Nearest Neighbour (NN) heuristic. But such a simple heuristic often fails to build a good solution due to the low amount of knowledge it can exploits. Therefore, more sophisticated heuristics are normally required to obtain outstanding solutions, but this is usually a hard and time consuming task (Branke et al. 2016) for experts. Alternatively, some hyper-heuristic may be used to search in a given space of heuristics. In this context, Genetic Programming (GP) stands out as one of the most common approaches (Burke et al. 2019), which

✉ Francisco J. Gil-Gala
giljavier@uniovi.es

[1] Department of Computer Science, University of Oviedo, Campus de Viesques s/n, Gijón 33271, Spain

[2] Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb 10000, Croatia

was already applied to a large variety of hard optimisation problems (Burke et al. 2012; Durasević et al. 2016; Gil-Gala et al. 2019; Nguyen et al. 2019; Zhang et al. 2021).

Nevertheless, a single heuristic, although performing well on average for a large set of instances, may not be good for a number of them individually. For this reason, several approaches based on ensembles (sets of heuristics) have been recently proposed for some optimisation problems, such as the One Machine Scheduling (Gil-Gala et al. 2020), the Unrelated Machines Scheduling (Durasević and Jakobović 2019), the Job Shop Scheduling (Hart and Sim 2016; Park et al. 2018), the Resource Project Scheduling (Dumić and Jakobović 2021), or the Capacitated Arc Routing Problem (Wang et al. 2019), among others.

In this paper, we investigate two types of ensembles, which are termed *collaborative* and *competitive*, respectively. A collaborative ensemble builds a solution in such a way that all the rules contribute to take the next decision in each iteration; while in a competitive ensemble each rule is exploited to build an independent solution and then the best of these solutions is considered as the solution of the ensemble. Each type of ensemble has its own weak and strong points. For example, collaborative ensembles may be exploited in online settings where the solution is being implemented at the same time it is being built (Durasević and Jakobović 2019), while competitive ensembles require all the problem data before starting to build a solution.

The ensembles we consider herein are aimed to solve the Dynamic TSP (DTSP) viewed as a sequence of static TSP over a time horizon. Therefore, they are exploited to solve TSP instances by a limited time. This time should be much lower than the time interval between every two consecutive TSP instances, which, of course, it will depend on the particular DTSP setting.

To establish the extent to which the ensembles are viable for DTSP, we performed an experimental study on the set of instances proposed in Duflo et al. (2019). They are instances with different sizes that are taken from the TSPLIB (2022). To create ensembles, we exploit a Genetic Program (GP) to evolve a set of heuristics, which are then combined into ensembles by means of a Genetic Algorithm (GA). The results of this study show that the quality of the solutions produced by the ensembles makes up for the larger time they require with respect to that of single rules, and that competitive ensembles perform much better than collaborative ones.

The remainder of the paper is organised as follows. In the next section, we give the formulation of the (Dynamic) TSP. The proposed solving method is described in Sect. 3. Then, in Sect. 5, we detail the combined approach of GP and GA to evolve heuristics and ensembles. In Sect. 6, we report the results of the experimental study. Finally, in Sect. 7, we summarise the main conclusions and outline some ideas for future work.

2 The travelling salesman problem

In the classical version of the Travel Salesman Problem (TSP), we are given a symmetric matrix $D_{N\times N}$, in which $D_{i,j}$ indicates the distance between the cities i and j. The goal is to obtain an optimal tour, i.e., the shortest path for visiting all cities and returning to the starting city. Figure 1 shows an instance with 5 cities and one of its solutions.

The dynamic version of the TSP, i.e., the DTSP, was introduced by Psaraftis in Psaraftis (1998). Since then, a number of variants were considered but there is not still a unified framework. In some cases, the distances between cities may change, and, in other cases, some cities may be removed or added. A review and taxonomy of the models proposed over the last three decades are given in Psaraftis et al. (2016). In general, in an instance of the DTSP, the distances between the cities, $M_{i,j}(t)$, may change over time following some temporal pattern that depends on the underlying problem. In this way, the DTSP is a continuous problem, but in practical settings, it is usually considered as a sequence of static TSP instances over a sequence of time points $t_i, i = 1, ...T$, each time interval $(t_i, t_{i+1}]$ being sufficiently short so that the instance at time t_i must be solved in real-time, indeed by taking a time much lower than $t_{i+1} - t_i$. Therefore, a particular solution may be viewed as a permutation of the N cities $s = [s_1, ..., s_N]$, which is evaluated as:

$$f(s,t) = D_{S_N,S_1}(t) + \sum_{n=1}^{N-1} D_{S_n,S_{n+1}}(t) \quad (1)$$

3 Solving the TSP in real-time

In accordance with the previous definition, solving an instance of the DTSP amounts to solving a sequence of static TSP instances at time points $t_i, i = 1, ..., T$. It often happens that the instances at times t_i and t_{i+1} are very similar. In these cases, repairing a previous solution may be better than generating some new one from scratch; therefore, some population based metaheuristics as Ant Colony Algorithms (ACO) or Evolutionary Algorithms (EA) may

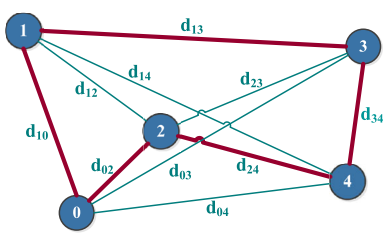

Fig. 1 A TSP instance with 5 cities and one of its solutions represented by the permutation (0,1,3,4,2)

be a good choice Mavrovouniotis et al. (2017). However, if there are abrupt changes from t_i to t_{i+1}, solving the new instance making a fresh start may be better. This is the option we take here; specifically, we propose to use some greedy algorithm guided by problem domain priority rules, under the assumption that the time intervals $t_{i+1} - t_i$ between consecutive solutions may be too short to use exact methods or even population based metaheuristics.

Data: A TSP instance.
Result: A feasible route R.
$R \leftarrow$ starting city;
$UVC \leftarrow$ all unvisited cities;
while $UVC \neq \emptyset$ **do**
  A city $u \in UVC$ is selected heuristically;
  Add u to the route R;
  Remove u from UVC;
end
return *The route* R;

In the TSP context, greedy algorithms are usually termed *route generation schemes*, as in each iteration they select the next city applying some heuristic until a complete tour is built. The procedure we use here is given in Algorithm 1; the heuristic is used as a priority rule, meaning that it assigns a priority value to each unvisited city, and the city with the highest priority is selected to be visited next. An example of such a rule is the well-known Nearest Neighbour (NN) heuristic: the priority of the candidate city j after i is calculated from only one problem attribute, the distance D_{ij}, as $1/D_{ij}$. Figure 2 illustrates the use of this heuristic; in the example, the next city to be visited after A will be B as it is the closest city to A among the unvisited cities.

Because of NN actually exploits too few information on the problem, it often happens that it builds a tour that seems good on the first cities, but that is really bad for the last ones due to the unvisited cities being quite dispersed and far from each others in the last iterations. To avoid the NN's low performance, we may consider attributes other than the distances to the next candidate cities. Specifically, we could consider, for example, some measure of the dispersion of the remaining unvisited cities. But a large number of attributes makes it difficult the problem of devising new heuristics, so that an automatic procedure may be the best option.

In Duflo et al. (2019), the authors consider 7 attributes and exploit GP to evolve priority rules, which are evaluated taking quadratic time complexity. Those attributes were also exploited in Singh and Pillay (2022) with a novel hyper-heuristic based on ant colony optimisation (HACO). Both works show that the evolved heuristics actually outperform NN and some other well-known classic heuristic algorithms for the TSP as nearest insertion or the Christofides heuristic Christofides (1976).

We conjecture that exploiting a low number of simple attributes could be enough to achieve reasonable heuristics that in turn could be evaluated taking less time. This is the rationale of the GP approach proposed in Sect. 5.1. In addition to single rules, we also explore here the use of ensembles of rules (see Sect. 4); the rationale is that combining the recommendations from a set of rules we may take wiser decisions than that from single rules.

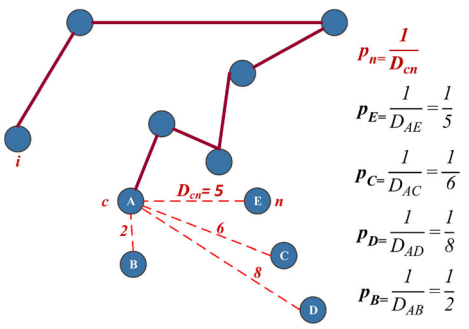

Fig. 2 Application of the Nearest Neighbour (NN) heuristic

4 Ensembles of rules

Under the assumption that a single rule may not be robust enough to produce good solutions for all instances in a given set, we explore here the use of ensembles. An ensemble is just a set of rules. Figure 3 shows an ensemble composed by 3 rules.

From previous experience on some problems as, for example, the one machine sequencing with variable capacity (Gil-Gala et al. 2020), or the unrelated parallel machines scheduling Durasević and Jakobović (2019), we propose to use two kinds of ensembles, which are termed *collaborative* and *competitive*, respectively.

Collaborative ensembles are indeed like the classic ensembles used in other contexts as classification or recommendation systems. The rationale of these ensembles is

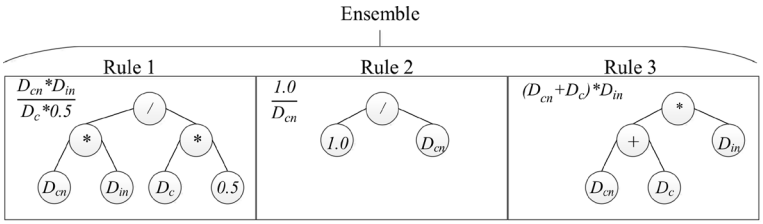

Fig. 3 An example of an ensemble composed of three rules. D_{cn} denotes the distance from city c to n; D_{in} denotes the distance from city i to n; and D_c denotes the distance from the centroid of the unvisited cities to the city c

that good rules take the right decisions in most of the situations and fail in a low number of them. Therefore, the decision on the next city to visit may be taken by aggregating the recommendations of each rule in the ensemble. In Durasević and Jakobović (2019), the authors analysed the two classic aggregation methods, namely *sum* and *vote* and they opted for the second one to avoid the issue of normalising the priorities of the individual rules, which is not an easy problem in general. This is the approach we consider here as well. In the voting method, each rule assigns the value 1 to the city with the largest priority and 0 to the remaining ones. Then, these values are summed up and the city with the largest sum value is chosen, breaking ties at random.

In turn, the rules in competitive ensembles work independently from each other to build a different solution each. Then, the best of these solutions is taken as the solution produced by the ensemble. The rationale of this kind of ensembles is that a good rule produces good solutions to some instances but it may produce bad solutions to others; therefore taking different rules, one can *cover* reasonably well all instances in a given set.

In both cases, competitive and collaborative, the ensembles may be built from a given set of heuristics as it was proposed in Durasević and Jakobović (2019), where the authors analysed 5 methods to create collaborative ensembles, namely random selection, probabilistic selection, grow, grow-destroy and instance based. In all cases, the ensemble starts from just a random rule and then new rules are added iteratively up to a given limit. Each time a new rule is added, the partial ensemble must be evaluated on a training set of instances. As an alternative, we propose to use here a Genetic Algorithm (GA) to build ensembles of both types (see Sect. 5.2).

5 Evolving heuristics and ensembles

In this work, we use the same methodology as in Durasević and Jakobović (2019), Gil-Gala et al. (2022). Therefore, a large set of heuristics (priority rules in this context) is previously evolved by Genetic Programming (GP), and then these rules are used to build ensembles by a Genetic Algorithm (GA).

5.1 GP to evolving priority rules

Priority rules are simple arithmetic expressions that may be naturally represented by trees. For this reason, the framework of GP proposed by John Koza (1992) is widely used to evolve new heuristic rules. To use GP, the first issue is to establish a set of symbols and some grammar. The grammar restricts the set of expression trees that can be built from the symbols, so that it fixes the search space of GP. The set of symbols must include a number of attributes of the problem, some constants and a set of operators. In this work, we consider three problem attributes, namely

- D_{cn}: Distance from c to n.
- D_{in}: Distance from i to n.
- D_c: Distance from the centroid of the unvisited cities to c.

where c denotes the current city in the partial tour built so far, i is the initial city, and n is a candidate city to be visited next. D_c is calculated as the distance between c and the point $c\bar{n}$ (centroid of the unvisited cities excluding n) defined by the coordinates $x = \frac{X-x_n}{N_{rm}-1}$ and $y = \frac{Y-y_n}{N_{rm}-1}$ where N_{rn} is the number of remaining cities to visit, X and Y are the summation of x-values and y-values of the unvisited cities and x_n and y_n are the coordinates of n. Figure 4 shows an example of these terminals.

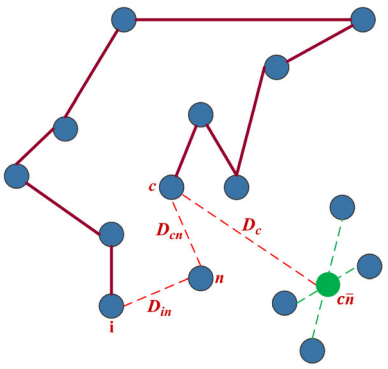

Fig. 4 Illustration of the three terminal symbols used

Table 1 Function and terminal sets used to build expression trees. Symbol "–' is considered in unary and binary versions

Binary functions	–	+	/	×	*max*	*min*	
Unitary functions	–	pow_2	*sqrt*	*exp*	*ln*	max_0	min_0
Terminals	*Dcn*	*Din*	*Dc*				
Numeric constants	0.1	0.2	...	0.8	0.9	1.0	

max_0 and min_0 return the maximum and minimum of an expression and 0

We also include 10 constants and a number of unitary and binary arithmetic functions in the set of symbols. The whole set is given in Table 1. The set of attributes D_{cn}, D_{in}, D_c is indeed a subset of the 7 attributes considered in Duflo et al. (2019). As mentioned, the rationale of this selection of attributes is to consider a small number of them and that they are meaningful and easy to evaluate at the same time.

The GP strategy is rather conventional and it is quite similar as in other studies (Durasević et al. 2016; Gil-Gala et al. 2021; Nguyen et al. 2019; Zhang et al. 2021; Duflo et al. 2019). GP starts from an initial population generated by the well-known ramped half-and-half method (Koza 1992). Then, GP follows an evolutionary scheme in which parents are randomly selected into pairs at the beginning of each generation; each pair of parents is combined, and their offspring are mutated with a given probability. The genetic operators are the well-known one-point crossover and subtree mutation (Koza 1992). Finally, in the replacement phase, from each two parents and their offspring, the best child is selected unconditionally and the second selection comes from tournament between the parents and the other offspring. The evaluation is the same as in Duflo et al. (2019), Gil-Gala et al. (2022), each candidate rule is evaluated on a set of TSP instances (the training set), and the fitness value of the rule is given by the inverse of the average tour of all instances.

5.2 GA for building ensembles

To build ensembles, either collaborative or competitive, we are given a set of rules $\mathcal{R}$ and the goal is to come up with a subset of maximum size P of rules, so that the ensemble composed by these rules performs as well as possible on a given (training) set of TSP instances. In this work, we adapted the GA proposed to build competitive ensembles in Gil-Gala et al. (2022). This GA may be used to build collaborative ensembles just by changing the evaluation function. As proposed in Gil-Gala et al. (2020, 2023), the encoding schema is variations with repetition from $\mathcal{R}$ taken P by P. Figure 5 depicts an example of ensemble encoding. In this illustration, $\mathcal{R}$ consists of five rules, and the ensemble is represented as an array containing three rules, each encoded by corresponding indices: 3, 1, and 4. This allows for representing subsets with maximum size P and for classic genetic operators as one-point crossover and single mutation. The evolutionary schema is quite similar to that of GP described in Sect. 5.1, and the population is randomly generated.

Regarding the evaluation of candidate ensembles, there are substantial differences depending on collaborative and competitive ensembles. In the first case, each of the instances in the training set must be solved by each candidate ensemble, in similar way as done by GP to evaluate a candidate rule. In a good collaborative ensemble it is expected that most of the rules take the right decision in each iteration of the routing generation scheme (see Algorithm 1) when it solves every instance in the training set; only in this way the ensemble will produce eventually a good solution.

However, the evaluation of competitive ensembles can be done much more efficiently. If the results of each rule in $\mathcal{R}$ on each instance of the training set were known in advance, we would not need to obtain a new solution from the candidate competitive ensemble, as this solution is just that from the best rule in the ensemble. However, for the sake of fair comparison to collaborative ensembles, in the experimental study (see Sect. 6) we consider that the above results are not known in advance. Therefore, each rule in the competitive ensemble must be evaluated on the training set, but only when it appears in an ensemble for the first time, as this result may be kept to be used in the same or further generations of the GA. In a good competitive ensemble, it is expected that at least one of the rules produces a good solution to each problem instance in the training set. In other words, the fittest collaborative ensembles evolved by GA should provide a good covering of the training set, i.e., for each instance in the training set, they should include one of the rules that perform the best for this instance.

6 Experimental study

We performed an experimental study to assess the viability of the ensembles and to compare their performance with respect to that of individual rules.

6.1 Experimental setup

We implemented prototypes of GP and GA in Java 8 and ran a series of experiments distributed into a Linux machine: a Dell Power Edge R740 with 2 x Intel Xeon Gold 6132 (2.6GHz, 28 cores) and 128GB.

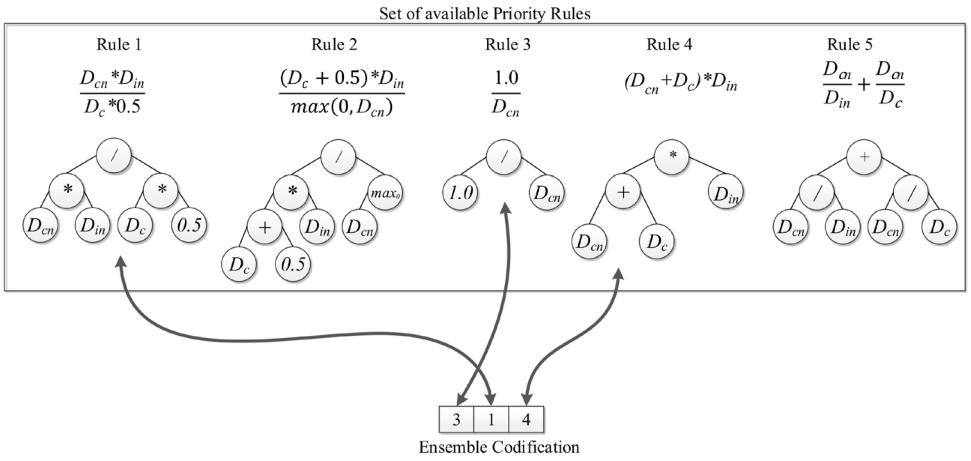

Fig. 5 An example of an ensemble with three heuristics encoding in GA

The test bed is composed of the set of 70 TSP instances considered in Duflo et al. (2019); Gil-Gala et al. (2022). They are the euclidean instances (EDGE-WEIGHT_TYPE=EUC_2D) from the TSPLIB (2022) having less than 4000 cities. As in the above works, the same 21 instances were used for testing and the remaining 49 instances were used for training, with a number of cities between 52 (berlin52) and 3795 (fl3795),

To analyse the effect that the size of the instances may have on the quality of the ensembles, we used training sets composed out of the N smallest instances, N taking different values as it is showed in Table 2.

The set of rules $\mathcal{R}$ was calculated by GP. This set is composed of 42 000 rules out of which 35 296 are syntactically different. They were recorded from the last population in each GP execution. Specifically, 6 000 rules (200 individuals and 30 executions) were collected by training the GP with each training set in Table 2.

The parameters used for GP and GA are summarised in Table 3. These values were taken from some previous experiments reported in Gil-Gala et al. (2022). We considered sizes 3, 5 and 7 for both types of ensembles, collaborative and competitive. For each configuration of parameters, GP and GA were executed 30 times, and the best, average, and standard deviation of the 30 solutions (heuristics or ensembles) were recorded on both the training and the test sets.

As mentioned, we have only used the vote combination method for collaborative ensembles (Durasević and Jakobović 2019).

GP was firstly run starting from random initial population of rules and then from a population built from rules evolved in previous executions of GP. This was done for the sake of a fair comparison between ensembles and rules.

In all cases, the stopping condition of the algorithms was given by a number of generations, but we also established a time limit of 1440 min. Thus, the executions where the field "Time(min)" is 1440 min mean that the algorithm terminated before reaching the maximum number of generations. In addition, we report the number of chromosomes syntactically different (the field "Unique"), which denotes the average number of unique chromosomes (rules or ensembles) per configuration.

Table 2 Description of the Training and Test sets of TSP instances used in the experimental study

Size of the set	Number of cities
Training (N)	in all
7	574
14	1391
21	2595
28	4722
35	10454
42	19756
49	37303
Test (21)	11757

Each training set is composed by the N smallest instances out of the 49 training instances

Table 3 Parameters used by GP and GA

Parameter	GP	GA
Population size	200	100
Crossover ratio	1.0	0.8
Mutation ratio	0.02	0.2
Chromosome length	$2^8 - 1$	3, 5, 7
Generations	100	50

6.2 Analysis of GP and GA

In this section, we analyse the results of the rules and ensembles produced by GP and GA, respectively, with different settings. In particular, we will try to assess their generalisation capability.

6.2.1 Priority rules evolved by GP

Table 4 summarises the results obtained by the rules evolved by GP. For each setting (Init./*N*), the best and average tardiness, and the standard deviations, of the 30 rules are reported for both the training and test sets; for the test set, the best value refers to the best rule in training. We can observe that starting from a heuristic initial population, GP is more stable and it is able to reach slightly better rules on average than when it starts from random populations. However, this difference vanishes on the test set. The time taken, as expected, is in direct ration with the size of the training set. And the number of different chromosomes evaluated along each execution is about 1/4 of the maximum theoretical value ($300 \times 200 = 60\ 000$), with the only exception for the largest training set when GP did not reach 200 generations. The differences between heuristic and random initial populations for different sizes of the training set may be better observed in the box-plots from the results on the test set given in Fig. 6. As it could be expected, the lowest value of *N* produces the worst results. Besides, there

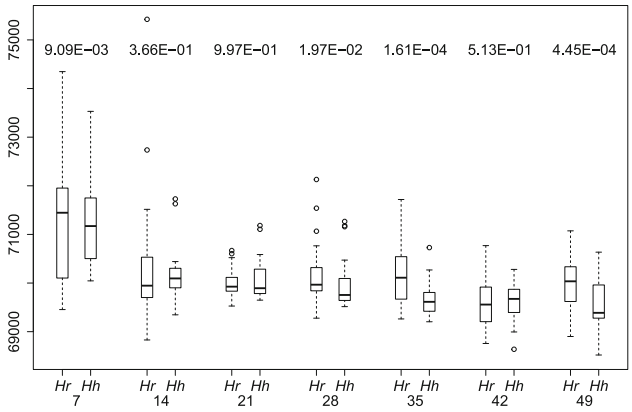

Fig. 6 Box plots from the results achieved by the **priority rules** for each test set (*N*) and initialisation method (*Hr* random or *Hh* heuristic) (see Table 4). On the top are the p-value produced by the Wilcoxon signed-rank test

are significant statistical differences between random and heuristic initialisation for only 4 of the 7 values of *N*.

6.2.2 Collaborative ensembles evolved by GA

Table 5 and Fig. 7 summarise the results obtained by the collaborative ensembles evolved by GA. In this case, the main observation we may draw is that the performance of the ensembles on the test set improves with the size of the training set, and that there are no significant differences between the three sizes of the ensembles for each training set. Besides, the time taken by GA grows exponentially, so

Table 4 Tardiness values of the solutions reached by the **priority rules** evolved by GP on both the training and test sets

Init	*N*	Training			Test			Unique	Time (min)
		Best	Avg	SD	Best	Avg	SD		
Random	7	27993.56	28350.12	5038.23	69454.45	71322.78	13087.45	15434.67	2.80
	14	35672.09	36262.86	6532.08	68829.59	70295.88	12660.79	15815.00	8.62
	21	34398.35	34900.26	6372.18	69527.13	69993.02	12550.59	15468.60	19.02
	28	31901.50	32332.86	5756.63	69277.89	70149.60	12841.70	15499.37	55.42
	35	52672.88	53312.92	9550.57	69262.11	70190.18	12772.45	15262.13	317.24
	42	67907.92	68480.39	12275.50	68757.23	69579.94	12422.91	15377.30	976.42
	49	88598.57	89176.15	15966.83	68900.89	70035.29	12605.37	7178.10	1440.00
Heuristic	7	27687.53	27992.59	91.81	70044.92	71357.78	897.42	16394.10	3.72
	14	35525.94	35645.62	66.61	69346.70	70165.78	478.14	16155.70	10.88
	21	34339.52	34449.29	72.40	69648.37	70063.90	394.31	16172.00	25.56
	28	31721.55	31860.89	45.47	69515.10	69940.53	475.22	15384.60	65.47
	35	52342.04	52617.48	108.33	69203.81	69652.06	317.32	17482.97	544.81
	42	67788.02	67939.04	90.09	68638.03	69653.03	351.12	14654.13	1434.95
	49	88005.64	88160.02	98.81	68520.16	69516.31	471.74	4972.37	1440.00

The last column shows the time taken by GP in one execution, and the next-to-last column shows the number of different candidate rules that were evaluated

Table 5 Tardiness values reached by the **collaborative ensembles** calculated via GA from different training sets (see Table 2) with maximum ensemble sizes P of 3, 5 and 7, on the training and test sets

Ensembles size	N	Training			Test			Unique	Time (min)
		Best	Avg	SD	Best	Avg	SD		
3	7	27873.02	28014.24	47.48	69373.19	70445.39	711.62	2804.17	2.05
	14	35433.68	35575.04	63.88	69361.87	69940.64	204.93	2972.60	6.19
	21	34192.74	34324.42	60.16	69516.05	69848.49	234.67	2965.13	14.61
	28	31677.43	31787.73	60.21	69171.82	69831.51	403.42	3040.57	44.41
	35	52295.07	52430.72	83.70	68830.68	69835.40	524.95	3041.53	253.81
	42	67475.08	67672.47	100.91	68966.14	69567.59	343.50	2992.97	778.44
	49	87679.24	87984.75	114.64	68587.06	69493.56	429.87	1639.20	1440.00
5	7	27659.92	27878.15	90.79	68636.93	70009.78	748.22	7475.17	8.49
	14	35381.98	35515.80	62.51	69365.78	69822.33	344.14	7509.23	25.77
	21	34176.03	34247.26	53.63	69324.13	69836.75	293.44	7562.43	60.63
	28	31576.70	31704.55	65.31	68764.52	69631.65	321.84	7544.53	175.00
	35	52065.30	52294.11	95.30	69084.28	69652.65	372.89	7584.40	1043.92
	42	67417.96	67675.33	101.33	68646.48	69360.25	434.75	3486.60	1440.00
	49	87796.99	88052.49	119.91	68767.62	69341.27	330.48	1035.03	1440.00
7	7	27708.04	27914.45	85.61	68636.26	70340.12	1396.05	8063.63	12.80
	14	35357.52	35491.00	60.48	69060.45	69998.35	810.03	8071.70	37.52
	21	34083.22	34232.77	77.10	68776.47	69701.41	484.94	8080.20	90.37
	28	31531.14	31678.09	74.11	68958.73	69690.43	352.97	8077.63	266.30
	35	51991.21	52283.78	92.46	68864.30	69689.73	493.69	7631.20	1434.52
	42	67511.10	67720.83	81.39	68714.03	69487.71	442.92	2577.30	1440.00
	49	87790.67	88054.16	120.66	68509.95	69403.14	372.92	772.03	1440.00

The last column shows the time taken by GA in one execution, and the next-to-last column shows the number of different ensembles that were evaluated

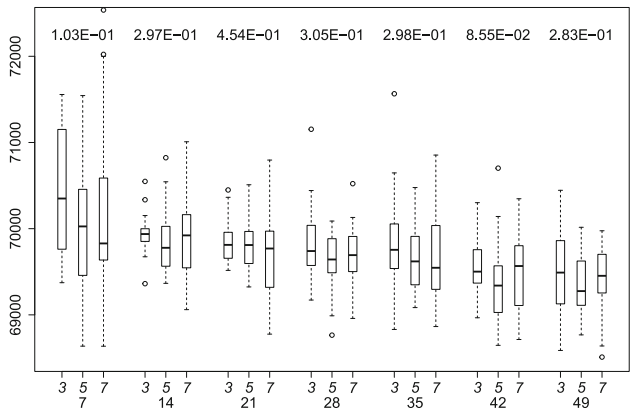

Fig. 7 Box plots of the results reported in Table 5 obtained by **collaborative ensembles** on the test set. The p-values produced by the Kruskal-Wallis test for the three sizes of the ensembles and each training set in the X-axis are given at the top of the figure

that it is unable to complete the 200 generations for $N = 49$ in all cases and even for $N = 42$ for ensembles of sizes 5 and 7. This is not surprising as GA must build a new solution for each instance in the training set to evaluate each candidate ensemble, and the average size of the instances grows with the value of N (see Table 2).

6.2.3 Competitive ensembles evolved by GA

The results achieved by competitive ensembles are reported in Table 6 and Fig. 8 in a similar way as it was done for collaborative ensembles. As in that case, we can observe that the performance of the competitive ensembles slightly improves on average with the size of the training set. But, at difference with collaborative ensembles, the performance of competitive ensembles strongly improves with the size of the ensemble, as it is shown by the Kruskal-Wallis test (Fig. 8). Besides, the time taken by GA grows smoothly with the size of the training set so that it is able to complete the 200 generations much earlier than the time limit in all experiments. This is not surprising as to evaluate a competitive ensemble GA does not need to build a new solution for the rules that took part in previous ensembles when searching for the best solution from the compounding rules. Here, we have to be aware that the efficiency of competitive ensembles could be further

Table 6 Tardiness values reached by the **competitive ensembles** calculated via GA from different training sets (see Table 2) with maximum ensemble sizes P of 3, 5 and 7, on the training and test sets

Ensembles size	N	Training			Test			Unique	Time (min)
		Best	Avg	SD	Best	Avg	SD		
3	7	27353.42	27506.00	86.79	68463.28	69497.97	604.78	4940.87	0.13
	14	35045.90	35109.66	41.12	68263.02	68866.88	313.27	4939.57	0.39
	21	33781.69	33867.67	35.67	68304.36	68883.44	334.60	4938.83	0.93
	28	31257.39	31294.31	24.06	68485.40	68764.59	238.63	4943.57	2.72
	35	51634.35	51742.80	54.46	68276.43	68698.23	205.00	4936.87	15.32
	42	66773.68	66879.59	70.23	68016.01	68628.00	197.91	4940.57	46.81
	49	85495.15	85664.19	163.50	68669.05	68949.68	189.36	4940.03	166.14
5	7	27216.97	27293.61	62.50	67780.03	68809.25	512.60	4942.10	0.26
	14	34782.60	34867.67	62.47	67677.34	68546.51	348.41	4939.13	0.77
	21	33511.84	33576.51	39.01	67806.67	68602.87	331.05	4938.13	1.89
	28	30976.03	31028.65	37.39	67816.30	68555.99	271.66	4937.10	5.51
	35	51217.01	51347.24	48.44	67996.22	68457.20	186.38	4937.93	31.92
	42	66380.41	66459.25	44.27	67451.11	68184.61	361.76	4941.77	95.50
	49	84887.95	85049.35	82.14	67609.02	68313.06	290.02	4935.00	341.99
7	7	27200.60	27251.14	34.59	67361.56	68582.18	469.02	4942.03	0.40
	14	34683.95	34739.64	40.13	67488.50	68215.83	300.75	4940.27	1.17
	21	33271.50	33395.13	57.79	67548.75	68274.20	324.06	4938.53	2.85
	28	30818.36	30897.42	34.40	67636.12	68159.56	264.63	4942.20	8.29
	35	51084.28	51183.36	42.15	67549.85	68184.96	251.13	4941.03	47.67
	42	66020.20	66207.18	79.34	67332.82	67825.75	275.14	4939.13	142.25
	49	84614.00	84781.66	63.14	67295.09	67968.94	312.64	4942.53	501.69

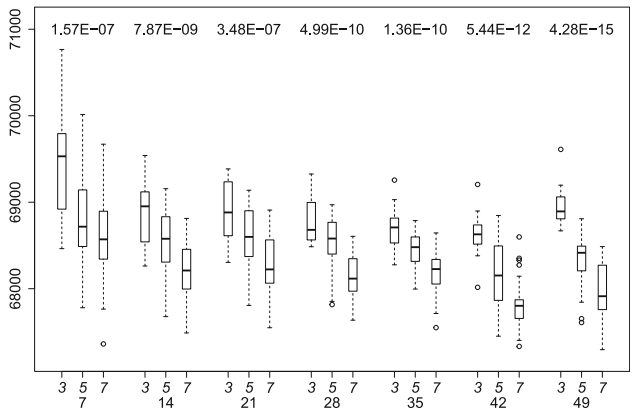

Fig. 8 Box plots of the results achieved by **competitive ensembles** solving the test set that are reported in Table 6. The numbers at the top are the p-values produced by the Kruskal-Wallis test

improved if the results of the rules on the training set were available beforehand, which may be a reasonable assumption. In this case, the time taken to evaluate a competitive ensemble would be independent of the size of the instances, and so GA would run in linear time on the number of instances in the training set.

6.3 Comparison

In this section, we show a comparison between ensembles and single rules. We also provide a comparison against classical heuristics like the nearest neighbour and the best-known solutions in the literature obtained by exact methods and metaheuristics.

6.3.1 Single rules versus ensembles

Table 7 summarises the differences between ensembles and single rules with regards to the quality of the solutions reached. Collaborative ensembles do not always produce better results than single rules and there are statistical differences in only 15 out of the 21 configurations (N,P). In turn, competitive ensembles are always better than single rules, with only one exception in the best solutions reached by the best ensemble with 3 rules and the best rule, and they show significant statistical differences in all configurations.

Table 7 Summary of the comparison between rules and ensembles on the test set

Ensembles	N	Ensemble size (P)								
		Best solution			On average			Stat. Dif.		
		3	5	7	3	5	7	3	5	7
Collaborative	7	✔	✔	✔	✔	✔	✔	✔	✔	✔
	14			✔	✔	✔	✔			
	21	✔	✔	✔	✔	✔	✔	✔	✔	✔
	28	✔	✔	✔	✔	✔	✔	✔	✔	✔
	35	✔	✔	✔				✔	✔	✔
	42				✔	✔	✔			
	49			✔	✔	✔	✔	✔	✔	✔
Competitive	7	✔	✔	✔	✔	✔	✔	✔	✔	✔
	14	✔	✔	✔	✔	✔	✔	✔	✔	✔
	21	✔	✔	✔	✔	✔	✔	✔	✔	✔
	28	✔	✔	✔	✔	✔	✔	✔	✔	✔
	35	✔	✔	✔	✔	✔	✔	✔	✔	✔
	42	✔	✔	✔	✔	✔	✔	✔	✔	✔
	49		✔	✔	✔	✔	✔	✔	✔	✔

The symbol ✔means that the ensemble produces better results than single rules, or that the Wilcoxon signed-rank test shows statistical differences

6.3.2 Collaborative versus competitive ensembles

Figure 9 shows a comparison between collaborative and competitive ensembles by means of a series of box plots and p-values from the Wilcoxon signed-rank test, one for each configuration (N,P). We can observe that competitive ensembles perform better than collaborative ensembles in all cases. Even for some configurations, the worst result from competitive ensembles is better than the best result reached by collaborative ensembles.

6.3.3 Run-time analysis

Since the routing generation scheme given by Algorithm 1 guided by heuristics (rules or ensembles) is aimed to solve DTSP, we have to analyse the time taken by the algorithms to assess its suitability for the dynamic changes in each particular setting. From the algorithm structure, it is clear that the execution time will depend on both the size of the static TSP instance and the size of the rule or ensemble exploited.

To that purpose, we generated 1 000 random rules and 1 000 random ensembles of size $P = 5$. All the instances in the test set were solved by these rules and by these ensembles, in this case considering them as collaborative and competitive, and the time taken in each run was registered. We evaluated them independently, without the presence of any other ensembles. Consequently, we intentionally excluded the reuse of previously calculated results to conduct an unbiased investigation into runtime performance.

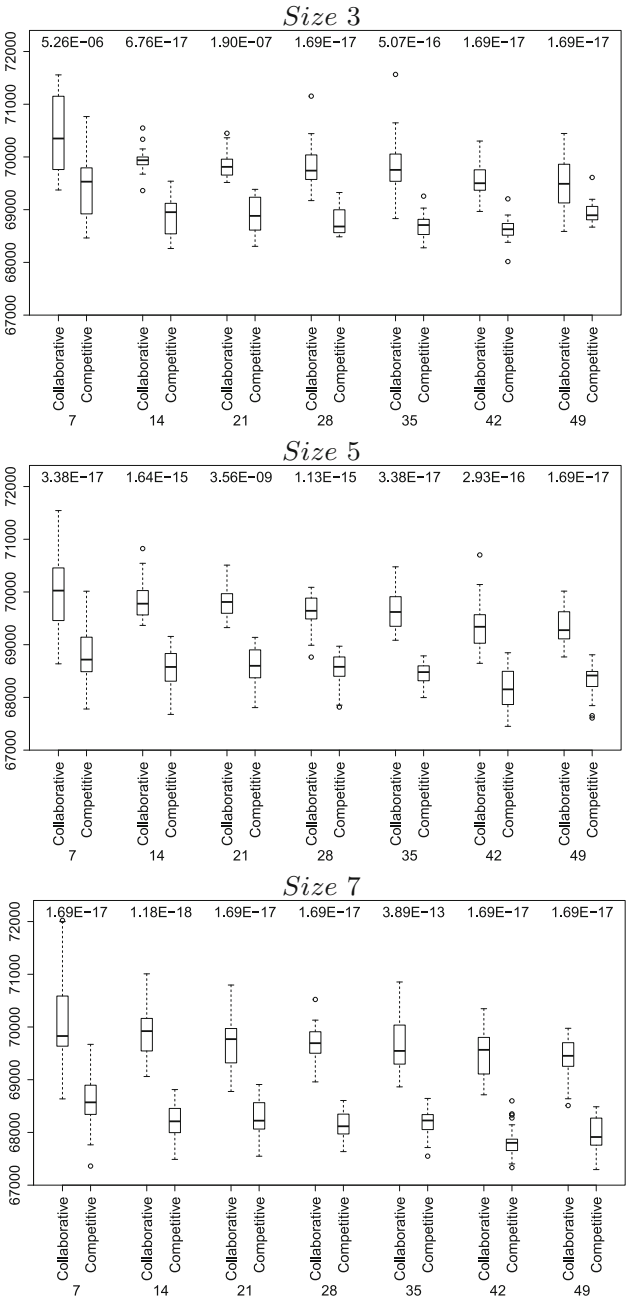

Fig. 9 Box plots from the results achieved by collaborative and competitive ensembles on the test set (detailed in Tables 5 and 6). For each ensemble size P, the box plots are organised by increasing values of the size of the training set N. The numbers at the top are the p-values from Wilcoxon signed-rank tests

Figure 10 shows the box plots of these experiments. The average times required to solve all instances in the test set were 1.62 and 1.63 s for competitive and collaborative ensembles respectively, and 0.3 s for the single rules. This

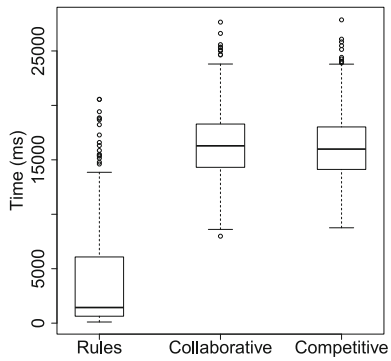

Fig. 10 Box-plot with the time required (in milliseconds) for solving the test set with each heuristic type

means that the expected times to solve one instance are about 7 ms for a rule and 35 ms for an ensemble.

In regards to the size of the heuristics, i.e., the number of symbols in a rule or ensemble, one may expect it be strongly correlated with the time taken by the algorithms. This is rather clear in Fig. 11, which show the dispersion plots of the time taken versus the size of the heuristics. The correlation coefficients in the three plots showcase high correlation between them.

Finally, we have to analyse the influence of the problem size on the time taken by the algorithms. To this end, Fig. 12 shows the bar plot of time versus instance size with the best rule and ensembles obtained. We can see that the number of cities and the time taken is directly related.

6.3.4 Comparison against the state-of-art

The TSP is an extensively studied problem, and numerous algorithms have been proposed to solve it, including the Lin-Kernighan heuristic (Link and Kernighan 1973), Christofides heuristic (Christofides 1976), and Genetic Local Search (GLS) (Freisleben and Merz 1996). While these algorithms demonstrate good performance, they face

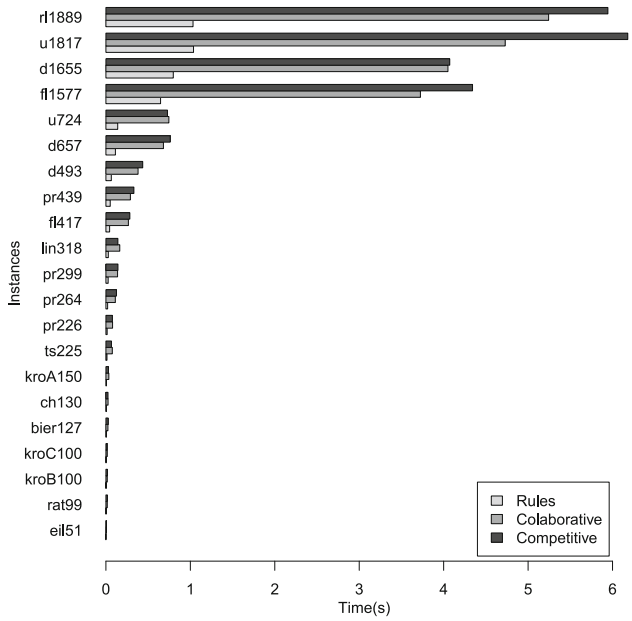

Fig. 12 Bar-plot with the time required (in seconds) for achieve the best solutions with each heuristic type

challenges when dealing with large problem instances, time constraints, or incomplete information. In such situations, a greedy algorithm guided by an efficient heuristic rule is often the preferred solution. Having said that, we consider classic heuristics as NN or NI, as well as the priority rules evolved by GPHH (Duflo et al. 2019) and HACO (Singh and Pillay 2022), as suitable references in the context of hyper-heuristics.

For our comparative study, we exploited NN and NI in combination with Algorithm 1 and considered the results of the best rules evolved by GPHH, which are presented in Duflo et al. (2019). We also include the results obtained by the best configuration of HACO from Singh and Pillay (2022). The results obtained by all the mentioned methods, detailed for each instance of the test set, are reported in

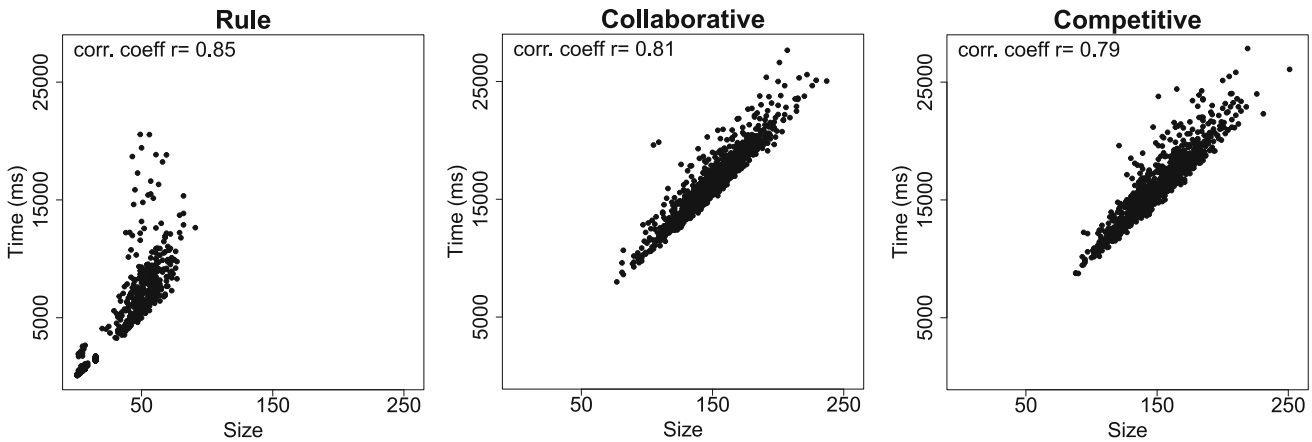

Fig. 11 Dispersion plots of the time taken versus the size of the heuristics. The values at the top are the Kendall rank correlation coefficients

Table 8 Comparison of the best rule and ensembles achieved by GP and GA against the Nearest Neighbour (NN) and Nearest Insertion (NI) heuristics, Genetic Programming Hyper-heuristic (GPHH) proposed in Duflo et al. (2019), Hyper-heuristic Ant Colony Optimisation (HACO) proposed in Singh and Pillay (2022), and the best-known (BK) solutions (BK 2022) solving the whole test set

Instance	BK	Best rule			Best rule	Best ensemble		HACO
		GPHH	NN	NI	GP	Collaborative	Competitive	
ts225	126643	136412.40	147941.80	151884.60	**139766.63**	**139766.63**	139978.00	131820.10
rat99	1211	1381.68	1474.92	1465.88	1383.86	1362.84	**1361.88**	1344.50
rl1889	316536	383303.70	391697.00	393573.50	377690.12	369445.48	**368533.90**	375383.20
u1817	57201	69334.72	69901.17	70970.14	66867.66	68765.38	**68643.83**	66247.10
d1655	62128	73740.45	76950.71	75390.58	74108.73	76421.03	**72468.16**	72292.10
bier127	118282	136781.20	145784.90	145544.10	129579.91	130718.90	**127755.71**	128104.10
lin318	42029	48039.78	52865.57	52299.12	47684.21	47987.33	**47197.19**	48190.90
eil51	426	469.46	562.16	494.75	479.06	480.57	**450.92**	452.30
d493	35002	40453.72	43403.90	42140.47	42007.21	41546.29	**39352.09**	40717.70
kroB100	22141	25254.54	27955.27	26908.61	25441.15	25005.23	**24798.05**	24394.50
kroC100	20749	24114.56	26094.22	25780.57	23024.51	22254.85	**22362.36**	22618.70
ch130	6110	7012.58	7677.60	7283.95	6668.41	7139.50	**6616.58**	6778.90
pr299	48191	56980.64	63334.80	60263.85	**55341.71**	**55341.71**	55456.73	55738.30
fl417	11861	14555.84	15706.24	14887.62	15079.67	14300.26	**14214.60**	13537.40
d657	48912	56882.87	63456.26	60081.63	58942.05	58423.95	**57567.52**	58546.70
kroA150	26524	30660.12	33440.39	31588.40	31239.31	29916.93	**29538.72**	30428.40
fl1577	22249	26163.75	27813.25	27625.77	26148.49	26588.93	**25678.45**	25308.50
u724	41910	48423.29	53834.65	52629.51	47955.74	48292.80	**46571.66**	48465.30
pr264	49135	60908.02	57915.59	65978.21	59363.51	60306.69	**57017.63**	54425.30
pr226	80369	92837.77	100178.30	102887.20	86995.73	**84509.99**	84851.38	89027.90
pr439	107217	130114.30	136546.50	133663.80	123155.76	130133.56	**122781.52**	126402.70
Avg	59277.43	69705.97	73549.30	73492.49	68520.16	68509.95	**67295.09**	67629.70

Table 8. The size of the ensembles is $P = 7$, and they were evolved from the largest training set ($N = 49$). The second column of the table shows the best known solution for each instance, which in some cases is optimal.

As may be expected, all methods produce solutions much worse than the best known ones, which were normally obtained by heavy exact or approximate methods that take much more time than greedy algorithms guided by heuristics. With regards to simple rules, it is clear that NN and NI perform worse than the best rules evolved by both GPHH and GP, showcasing the advantage of automatically calculated rules over the classical ones. The best rule evolved by GP is better on average and also in 11 of the 21 instances than that evolved by GPHH. In this case, showcasing that it is possible to obtain good rules considering a small number of problem attributes.

We can see that ensembles produce the best solutions among all the heuristics considered, being competitive ensembles better than collaborative ones in all but 3 instances. Furthermore, we can also observe that competitive ensembles achieve (on average) similar results to HACO. However, the HACO approach has the inconvenience that the evolved heuristic is difficult to interpret for the human eye. In this regard, a rule in HACO is encoded as a pheromone matrix that is much harder to interpret than expression trees.

7 Conclusions and future work

As it was done in some previous works (Duflo et al. 2019; Gil-Gala et al. 2022), we have seen that Genetic Programming is a suitable hyper-heuristic to evolve priority rules to solve the TSP. In our experimental study, these rules outperformed some classic heuristics, such as Nearest Neighbour or Nearest Insertion. From the comparison between GPHH (Duflo et al. 2019) and the GP proposed in this work, we can see that using a small number of problem attributes, the search space of GP is reasonably low. Therefore, it may reach better rules than those obtained from the search space generated from a large set of attributes.

We have also seen that ensembles of rules may produce better results than single rules at the cost of linear increase

of the execution time with the size of the ensembles. From the two kinds of ensembles analysed, competitive ensembles showed much better performance than collaborative ones. Nevertheless, we have to be aware that both of them were evaluated on building solutions for static TSP instances, which is a suitable framework when a DTSP is viewed as a sequence of static TSP instances over time. However, in other situations, the dynamic problems require on-line solutions, i.e., the route is being travelled at the same time as it is being built. In these cases, competitive ensembles may not be used, so collaborative ones may also represent a good alternative to single rules.

In future work, we plan to consider alternative combination methods (Durasević et al. 2023; Park et al. 2018) and multiobjective optimisation (Durasević et al. 2023). Additionally, we are interested in analysing alternative rule representations, such as neural networks (Branke et al. 2015; Jia et al. 2022) or pheromone matrices (Singh and Pillay 2022), to build ensembles.

Author Contributions All authors reviewed the manuscript. The specific contribution of each author is as follows: Francisco Javier Gil Gala: Conceptualisation, Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data curation, Writing-Original Draft, Visualisation. Marko Durasević: Conceptualisation, Methodology, Investigation, Writing–Review & Editing, Supervision. María Sierra: Visualisation, Formal Analysis. Ramiro Varela: Methodology, Writing—Review & Editing, Supervision, Project administration, Funding acquisition.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This research has been supported by the Spanish State Agency for Research (AEI) under research project PID2019-106263RB-I00, and by the Croatian Science Foundation under the project IP-2019-04-4333.

Availability of data and materials The test bed is composed of the euclidean instances (EDGE_WEIGHT_TYPE=EUC_2D) from the TSPLIB (2022).

Declarations

Ethical approval Not applicable.

Conflict of interest The authors declare that they have no conflict of interest.

References

Branke J, Hildebrandt T, Scholz-Reiter B (2015) Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. Evol Comput 23(2):249–277

Branke J, Nguyen S, Pickardt CW, Zhang M (2016) Automated design of production scheduling heuristics: a review. IEEE Trans Evol Comput 20(1):110–124

Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR (2019) A classification of hyper-heuristic approaches: revisited. In: Gendreau M, Potvin JY (eds) Handbook of metaheuristics. International series in operations research & management science vol 272, pp 453–477

Burke EK, Hyde MR, Kendall G, Woodward J (2012) Automating the packing heuristic design process with genetic programming. Evol Comput 20(1):63–89

Christofides N (1976) Worst-case analysis of a new heuristic for the travelling salesman problem Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group

Duflo G, Kieffer E, Brust MR, Danoy G, Bouvry P (2019) A gp hyper-heuristic approach for generating tsp heuristics. In: IPDPSW'19: IEEE international parallel and distributed processing symposium workshops, pp 521–529

Dumić M, Jakobović D (2021) Ensembles of priority rules for resource constrained project scheduling problem. Appl Soft Comput 110:107606

Durasević M, Jakobović D (2019) Creating dispatching rules by simple ensemble combination. J Heurist 25:959–1013

Durasević M, Jakobović D, Knežević K (2016) Adaptive scheduling on unrelated machines with genetic programming. Appl Soft Comput 48:419–430

Durasević M, Gil-Gala FJ, Planinić L, Jakobović D (2023) Collaboration methods for ensembles of dispatching rules for the dynamic unrelated machines environment. Eng Appl Artific Intell 122:106096

Durasević M, Gil-Gala FJ, Jakobović D, Coello-Coello CA (2023) Combining single objective dispatching rules into multi-objective ensembles for the dynamic unrelated machines environment. Swarm Evol Comput 80:101318

Freisleben B, Merz P (1996) A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems international conference on evolutionary computation

Gil-Gala FJ, Durasević M, Sierra MR, Varela R (2022) Building heuristics and ensembles for the travel salesman problem. In: Ferrández Vicente JM et al (eds) Bio-inspired Systems and Applications: from robotics to ambient intelligence. proceedings of IWINAC 2022. Lecture Notes in Computer Science, vol 13259. Springer, Cham

Gil-Gala FJ, Mencía C, Sierra MR, Varela R (2019) Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time. Appl Soft Comput 85:105782

Gil-Gala FJ, Sierra MR, Mencía C, Varela R (2020) Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling. Nat Comput 21:553–563

Gil-Gala FJ, Sierra MR, Mencía C, Varela R (2021) Genetic programming with local search to evolve priority rules for scheduling jobs on a machine with time-varying capacity. Swarm Evol Comput 66:100944

Gil-Gala FJ, Durasević M, Varela R, Jakobović D (2023) Ensembles of priority rules to solve one machine scheduling problem in real-time. Inf Sci 634:340–358

Hart E, Sim K (2016) A hyper-heuristic ensemble method for static job-shop scheduling. Evol Comput 24(4):609–635

Jia YH, Mei Y, Zhang M (2022) Learning heuristics with different representations for stochastic routing. IEEE Trans Cybern 53(5):3205–3219

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press

Link S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. Oper Res 21(2):498–516

Mavrovouniotis M, Müller FM, Yang S (2017) Ant colony optimization with local search for dynamic traveling salesman problems. IEEE Trans Cybern 47(7):1743–1756

Nguyen S, Mei Y, Xue B, Zhang M (2019) A hybrid genetic programming algorithm for automated design of dispatching rules. Evol Comput 27(3):467–496

Optimal TSP LIB Solutions. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html. Accessed 16 June 2022

Park J, Mei Y, Nguyen S, Chen G, Zhang M (2018) An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. Appl Soft Comput 63:72–86

Psaraftis HN (1998) Dynamic vehicle routing problems vehicle routing: methods and studies, pp 223–248 North-Holland

Psaraftis HN, Wen M, Kontovas CA (2016) Dynamic vehicle routing problems: three decades and counting. Networks 67:3–31

Punnen AP (2007) The traveling salesman problem: applications, formulations and variations the traveling salesman problem and its variations. Springer US, pp 1–28

Singh E, Pillay N (2022) A study of ant-based pheromone spaces for generation constructive hyper-heuristics. Swarm Evol Comput 72:101095

TSP Test Data. http://www.math.uwaterloo.ca/tsp/data/index.html. Accessed 1 Feb 2022

Wang S, Mei Y, Zhang M (2019) Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem. In: Proceedings of the genetic and evolutionary computation conference

Zhang F, Mei Y, Nguyen S, Tan KC, Zhang M (2021) Multitask genetic programming-based generative hyperheuristics: a case study in dynamic scheduling. IEEE Trans Cybern 1(3):1–14

Zhang F, Mei Y, Nguyen S, Tan KC, Zhang M (2022) Instance rotation based surrogate in genetic programming with brood recombination for dynamic job shop scheduling. IEEE Trans Evolut Comput 27(5):1192–1206