Building heuristics and ensembles for the travel salesman problem

Francisco J. Gil-Gala[1], Marko Đurasević[2], María R. Sierra[1], and Ramiro Varela[1]

[1] Department of Computer Science, University of Oviedo
[2] Faculty of Electrical Engineering and Computing, University of Zagreb

Abstract. The Travel Salesman Problem (TSP) is one of the most studied optimization problems due to its high difficulty and its practical interest. In some real-life applications of this problem the solution methods must be very efficient to deal with dynamic environments or large problem instances. For this reasons, low time consuming heuristics as priority rules are often used. Even though such a single heuristic may be good to solve many instances, it may not be robust enough to take the best decisions in all situations so, we hypothesise that an ensemble of heuristics could be much better than the best of those heuristic. We view an ensemble as a set of heuristics that collaboratively build a single solution by combining the decisions of each individual heuristic. In this paper, we study the application of single heuristics and ensembles to the TSP. The individual heuristics are evolved by Genetic Programming (GP) and then Genetic Algorithms (GA) are used to build ensembles from a pool of single heuristics. We conducted an experimental study on a set of instances taken from the TSPLIB. The results of this study provided interesting insights about the behaviour of rules and ensembles.

Keywords: Travel Salesman Problem, Heuristics, Ensembles, Hyper-heuristics

1 Introduction

Greedy algorithms guided by single heuristics are usually the best if not the only method suitable to solve large-scale problems or dynamic problems that require solutions in real-time, due to them being able to perform reasonable decisions quickly. In these situations, exact methods or even metaheuristics such as genetic algorithms are not practical because they are very time consuming. Single heuristics as priority rules are often manually designed by experts, but this is also very time consuming and may be a difficult task because the problem features that are relevant for the heuristics are not always evident to the human eye. For this reason, several hyper-heuristic methods have been exploited to automatically design heuristics for various optimization problems, such as the Job Shop Scheduling Problem [16] or the Unrelated Machines Scheduling Problem [7]. In spite of the success of these methods, it is often the case that a single

heuristic is not robust enough to produce good solutions to all instances of a given benchmark set. This fact motivates the consideration of ensembles as an alternative that could outperform single heuristics at the cost of a reasonable increment in the computational burden.

Over the last years, there have been some proposals for learning ensembles. For example, Hart and Sim [13] proposed the NELLI-GP method, which is used to create ensembles for the Job Shop Scheduling Problem, where each heuristic is used to solve a particular subset of instances. A different approach was proposed by Gil-Gala et al. [10] for the scheduling problem of a single machine with time-varying capacity; in this case, each of the heuristics in an ensemble is used to solve an instance, the final result being the best of the schedules. The calculation of ensembles is modelled as a set covering problem and solved by means of a genetic algorithm. However, the most common strategy to exploit ensembles is that all the heuristics build a solution collaboratively. This is the approach considered by Durasević and Jakobović in [5], where they remarked that the main decisions that have to be taken to build ensembles are 1) how heuristics are combined and 2) how the heuristics are selected to compose the ensemble. For the first problem, some classic techniques borrowed from machine learning such as summation and voting strategies are of common use, while some methods such as greedy algorithms [6, 4] or coevolutionary algorithms [5, 17, 4] were used for the second.

In this work, we are interested in developing ensembles for solving the Travel Salesman Problem (TSP). In our proposed method, Genetic Programming (GP) [14] is used to evolve a large pool of heuristics for solving the TSP. Then, a genetic algorithm (GA) is used to build ensembles from this pool. We conducted an experimental study across a set of instances taken from the TSPLIB[3].

The remainder of the paper is organised as follows. Firstly, we describe the method used to solve the TSP. Then, in Section 3 we present the proposed algorithms designed to evolve heuristics and ensembles. Next, in Section 4 we present the experimental analysis and the obtained results. Finally, in Section 5 we summarise the main conclusions and outline some ideas for future work.

2 The travel salesman problem and solving method

We consider the symmetric Travel Salesman Problem (TSP) that is a well-known NP-hard problem. We are given a matrix $D_{N\times N}$, where d_{ij} denotes the distance between cities i and j and the goal is to obtain an optimal tour, i.e., the shortest path visiting each of N cities exactly once and returning to the starting city. Several algorithms were proposed to find tours, for example genetic algorithms or the well-known Lin-Kernighan Heuristic [15]; however, none of them is actually efficient for large instances or in dynamic environments. In these situations, greedy algorithms guided by simple priority rules are the most, if not the only, viable solution method.

[3] http://comopt.ifi.uni-heidelberg.de

In this work, we exploit a greedy algorithm that, starting from the initial city, in each iteration selects the next city by means of a priority rule. An example of such rule is the Nearest Neighbour (NN) heuristic: if i is the current city, the priority of the candidate city j is given by $1/d_{ij}$. In general, a priority rule is an arithmetic expression that assigns a priority to each unvisited city, which is calculated from the problem attributes; in the NN heuristic, the only considered attribute is the distance d_{ij}.

A good priority rule usually produces good solutions for a number of instances, but it may produce bad solutions for other instances as well. At the same time, it is clear that different rules may produce quite different solutions for the same instance. For these reasons, it is often the case that a single rule may not be robust enough to produce good solutions for all the instances in a given set. To deal with this issue, a suitable alternative may be to exploit ensembles of rules so that each decision is taken from the aggregated values of the individual rules instead from just a single rule. In this way, a number of rules work together to produce a single solution.

To aggregate the priorities from the rules in the ensemble, we may use the classic summation or voting methods, each one having their own strong and weak points. In our study, we consider the vote combination method proposed in [6], where the priorities given by the rules are normalized so that each rule assigns 1 to the city with highest priority and 0 to the remaining ones. The city with the most votes is the one chosen by the ensemble, breaking ties uniformly.

3 Building heuristics and ensembles

The methodology used in this paper is similar to that used in [5, 6, 12, 9] for some scheduling problems. It consists of two steps: 1) a sufficiently large pool of heuristics, i.e., priority rules, is evolved by some hyper-heuristic, in this case Genetic Programming (GP), and 2) from this pool of heuristics, a search algorithm, in our case a Genetic Algorithm (GA), is exploited to build ensembles. In the following subsections we explain the main features of the proposed GP and GA.

3.1 Genetic Programming

Genetic Programming (GP) may be viewed as a hyper-heuristic that is widely used to evolve heuristics for optimization problems. In [2], the authors demonstrated that GP based hyper-heuristics may outperform some other machine learning techniques, such as regression or neural networks, in learning priority rules for a scheduling problem. In this study, we use a generational GP similar to the one proposed in [8], which implements the classical genetic operators proposed by John Koza [14]. They are the one-point crossover, the subtree mutation and the generation of the initial population by means of the ramped half-and-half method. In this paradigm, heuristics are encoded as expression trees [2], which are composed by terminal and function symbols. Terminals are relevant

attributes of the problem and function symbols are used to combine the terminals. We have used the following set of terminals for the TSP, which are some of those proposed in [3]:

- 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 n.

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 after 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.

These terminals can be evaluated in $O(1)$, as it is demonstrated in the above work. To do that, X and Y are initialized with the summation of x-values and y-values of the N cities and, when a city is visited, X and Y are updated by subtracting the coordinates of the last visited city.

The function set is the same used in [11] for a scheduling problem of a single machine with time-varying capacity. Additionally, the proposed alphabet includes some numeric constants. The whole set of symbols is summarised in Table 1.

Table 1: Function and terminal sets used to build expression trees. Symbol "-" is considered in unitary and binary versions. max_0 and min_0 return the maximum and minimum of an expression and 0.

Binary functions	-	$+$	$/$	$\times$	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	

3.2 The Genetic Algorithm

To represent ensembles of maximum size P, we encode a chromosome by a permutation with repetition of heuristics taken P at a time from the pool evolved by GP. Figure 1 shows an example with 3 rules, all of them are different in this case. We consider a generational strategy, similar to that used in [12], with random selection and replacement by tournament among every two mated parents and their two offspring. The initial chromosomes are random variations of heuristics taken uniformly from the pool. As in [12], we use one point crossover and a mutation operator that changes randomly a number of heuristics between 1 and $P/2$ in the chromosome.

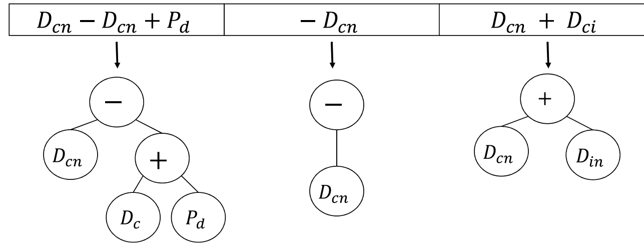

Fig. 1: An example of ensemble composed of three rules. Each rule is represented by the arithmetical expression in each array position.

3.3 The GP and GA fitness functions

In both algorithms, GP and GA, the evaluation of a chromosome involves solving a *training set* composed by a number of TSP instances. These instances are solved by each candidate rule in GP and by each candidate ensemble in GA. Therefore, the fitness function calculates the sum of distances produced by all tours created by a heuristic or an ensemble and returns the inverse of this value.

Note that each time a priority must be calculated, the entire tree encoding the heuristic must be traversed to compute the specific values for each terminal. Moreover, ensembles must perform this process with each heuristic and then compute a single priority for each unvisited city. Therefore, we opted to cache all evaluated chromosomes to prevent GP and GA from repeated evaluation.

4 Experimental study

We conducted an experimental study to analyse the components of the proposed method. To this aim, we implemented a prototype in Java 8 and ran a series of experiments distributed into a Linux machine Dell PowerEdge R740: 2 x Intel Xeon Gold 6132 (2.6GHz, 28 cores) and 128GB. The common termination criterion is given by 100 generations, and each configuration of GP and GA is executed 30 times. Additionally, we establish a day (1440 minutes) as the run-time limit for each execution.

4.1 Preliminaries

The TSPLIB is one of the most used sets of instances to validate solvers for the TSP. We considered the same instances as in [3], 112 in all, but removed those that are not of type EDGE_WEIGTH_TYPE = EUC_2D; so there are 78 left. We select the same 21 instances from them to compose the test set as in [3]. They used another set of 49 instances randomly selected from the remaining instances for training in their genetic program, termed GP-HH, which was parametrised with a population of 200 individuals, 100 generations as termination criterion and a maximum depth of trees of 17. Table 2 summarises the results achieved

by heuristics calculated by the GP-HH when solving the test set. As we can see, heuristics evolved by the GP-HH clearly achieves much better results than simple classical heuristics such as Nearest neighbour and Nearest insertion.

Table 2: Average distances produced when solving the test set by some classical heuristic (Nearest neighbour and Nearest insertion) and evolved by the GP-HH proposed in [3].

GP-HH best	Nearest neighbour	Nearest insertion
69705.97	73549.30	73492.49

In this work, our GP approach uses a population size of 200 individuals and crossover and mutation probabilities of 100% and 2%, respectively, which are similar to those reported in [8], the stopping condition and population size are the same as used by the GP-HH in [3]. However, the maximum depth of trees $\mathcal{D}$ that we use is much smaller, being $\mathcal{D}$=8 instead of $\mathcal{D}$=17. On the other hand, our terminal set comprises of only three instead of the seven terminals used in [3]. Table 3 shows the run-time (in seconds) necessary for solving the instances berlin52, lin318 and pr2392 with 200 random heuristics generated with $\mathcal{D}$ taken values 4, 8 and 12. As it is observed, the execution time of GP is directly related to the number of cities and the number of symbols in the heuristics.

Table 3: Time (in seconds) for solving three instances with three heuristics of different sizes. These intances have 52, 318 and 2392 cities, respectively. It also reports the average sizes of those heuristics.

	Time (sec.)			Avg. Size
$\mathcal{D}$	berlin52	lin318	pr2392	of heuristics
4	0.12	1.24	64.57	6.93
8	0.38	11.42	645.93	51.89
12	10.84	176.34	9980.31	493.38

In view of the results, we note that our target machine is not powerful enough to execute our GP implementation with large $\mathcal{D}$ values and number of cities. Taking into account that the evaluation of ensembles is much more costly than the evaluation of single heuristics, the GA is parametrised with the population size of 100 individuals and the crossover and mutation probabilities of 80% and 20%, respectively, which correspond to the values used in [12]. We restricted these experiments to construct ensembles composed of 3 heuristics and the stopping condition of 50 generations.

For all the above reasons, in the following experiments, we have limited the number of instances in the training set and the maximum sizes of heuristics for obtaining the results in a reasonable time. The training sets proposed were composed by selecting a maximum number of instances N with less number of cities The instances were picked from the set of 78 instances, disregarding the 21 instances (with a total of 11757 cities) used for testing. Additionally, we removed 8 instances with more than 4000 cities, resulting in a total of 49 instances with a number of cities between 52 (berlin52.tsp) and 3795 (fl3795.tsp), which sum up to a total of 37303 cities. The training sets are summarized in Table 4.

Table 4: The seven training sets proposed that are composed by the N instances with less number of cities.

	Number of cities
N	Cities
7	574
14	1391
21	2595
28	4722
35	10454
42	19756
49	37303

4.2 Results

Table 5 shows the results achieved by heuristics evolved by GP when solving the training and test sets. We observe that when the whole training set is used (N=49), the time taken by GP is prohibitive even though it evaluates half different chromosomes than the other combinations, due to the stopping condition is given by 1440 minutes execution, which happens before completing 100 generations. When N takes values 7 and 49, training sets with the smallest and largest number of cities respectively, the results in test are worse than other training sets, such as when N takes values 21 and 42, training sets with the best results in test. Specifically, with N=42 the heuristics are better than those calculated by [3]. Furthermore, our GP approach uses fewer training instances, fewer terminals to compose heuristics and smaller maximum size. Thus, we can conclude that our GP approach is able to achieve similar results as GP-HH, but evolving simpler heuristics. From our point of view, the high standard deviation (SD) in the test set with all settings is indicative of the stochastic nature of GP and provides motivation to use more robust methods like ensembles.

To build ensembles, we recorded all heuristics of the last population in each GP execution. Therefore, a total of 6000 heuristics (200 individuals and 30 executions) were collected from each training set. With the seven sets of heuristics,

Table 5: Results achieved by the best **heuristic** evolved in each execution of GP using seven training sets. Results with N=49 are got before complete 100 generations.

	Training			Test			Time	
N	Best	Avg.	SD	Best	Avg.	SD	(min)	Dif.
7	27993.56	28350.12	5038.23	69454.45	71322.78	13087.45	2.80	15434.67
14	35672.09	36262.86	6532.08	68829.59	70295.88	12660.79	8.62	15815.00
21	34398.35	34900.26	6372.18	69527.13	69993.02	12550.59	19.02	15468.60
28	31901.50	32332.86	5756.63	69277.89	70149.60	12841.70	55.42	15499.37
35	52672.88	53312.92	9550.57	69262.11	70190.18	12772.45	317.24	15262.13
42	67907.92	68480.39	12275.50	**68757.23**	**69579.94**	12422.91	976.42	15377.30
49	88598.57	89176.15	15966.83	68900.89	70035.29	12605.37	1440.00	7178.10

Table 6: Results achieved by the best **ensemble** evolved in each execution of GA using seven training sets and focusing on three as the size of ensembles.

	Training			Test			Time	
N	Best	Avg.	SD	Best	Avg.	SD	(min)	Dif.
7	27873.02	28014.24	47.48	69373.19	70445.39	711.62	2.05	2804.17
14	35433.68	35575.04	63.88	69361.87	69940.64	204.93	6.19	2972.60
21	34192.74	34324.42	60.16	69516.05	69848.49	234.67	14.61	2965.13
28	31677.43	31787.73	60.21	69171.82	69831.51	403.42	44.41	3040.57
35	52295.07	52430.72	83.70	68830.68	69835.40	524.95	253.81	3041.53
42	67475.08	67672.47	100.91	68966.14	69567.59	343.50	778.44	2992.97
49	87679.24	87984.75	114.64	**68587.06**	**69493.56**	429.87	1440.00	1639.20

we generated a large pool of heuristics to build ensembles. After eliminating duplicate heuristics (those syntactically equal), 35 296 heuristics compose the set used for building ensembles.

Finally, to assess the viability of ensembles we executed the GA to build ensembles of size 3. The results are summarized in Table 6 and Figure 2. From our point of view, ensembles are more robust than single heuristics since they are always perform better on average and show lower standard deviation when using the same training set, especially for the smaller sets. However, the best solutions are achieved by single heuristics in some cases. Additionally, there is a considerable gap between the best and average solutions in the test set which motivates further experimentation with alternative combination methods and ensemble sizes, which can help to better predict the behaviour in the test set.

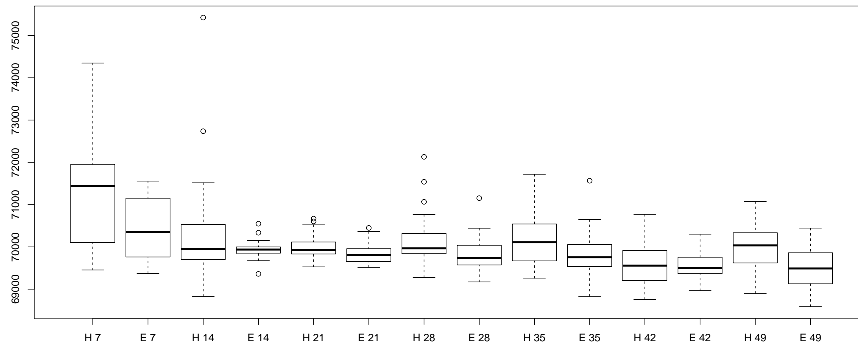

Fig. 2: Boxplots obtained from the results reached on the test set by the heuristics and ensembles evolved from the 7 trainning sets.

5 Conclusions and Future Work

We have seen that Genetic Programming is a suitable hyper-heuristic to evolve priority rules to solve the TSP that perform better than classic heuristics, as Nearest Neighbour or Nearest Insertion, designed by experts. Even though this is not new, in our experiments we obtained heuristics that are simpler than previous heuristics obtained by other GP implementations. But the main conclusion we may draw from our study is that ensembles obtained combining just a few heuristics (3 in our experiments) are able to improve the results from the best heuristic working alone. At the same time, we consider that there is still room for ensembles to improve. To this end, we will try to devise new methods to exploit them more efficiently.

Acknowledgements

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.

References

1. TSP Test Data. `http://www.math.uwaterloo.ca/tsp/data/index.html`, [Online; accessed 1-February-2021]
2. Branke, J., Hildebrandt, T., Scholz-Reiter, B.: Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evolutionary Computation 23(2), 249–277 (2015)

3. Duflo, G., Kieffer, E., Brust, M.R., Danoy, G., Bouvry, P.: A gp hyper-heuristic approach for generating tsp heuristics. In: IPDPSW'19: IEEE International Parallel and Distributed Processing Symposium Workshops. 521–529 (2019)
4. Dumić, M., Jakobović, D.: Ensembles of priority rules for resource constrained project scheduling problem. Applied Soft Computing 110(1) (2021)
5. Durasević, M., Jakobović, D.: Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. Genetic Programming and Evolvable Machines 19(1), 53–92 (2018)
6. Durasević, M., Jakobović, D.: Creating dispatching rules by simple ensemble combination. Journal of Heuristics 25, 959–1013 (5 2019)
7. Durasević, M., Jakobović, D., Knežević, K.: Adaptive scheduling on unrelated machines with genetic programming. Applied Soft Computing 48, 419–430 (2016)
8. Gil-Gala, F.J., Mencía, C., Sierra, M.R., Varela, R.: Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time. Applied Soft Computing 85, 105782 (2019)
9. Gil-Gala, F.J., Mencía, C., Sierra, M.R., Varela, R.: Learning ensembles of priority rules for on-line scheduling by hybrid evolutionary algorithm. Integrated Computer-Aided Engineering 28(1), 65–80 (2021)
10. Gil-Gala, F.J., Sierra, M.R., Mencía, C., Varela, R.: Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling. Natural Computing (2020)
11. Gil-Gala, F.J., Sierra, M.R., Mencía, C., Varela, R.: Genetic programming with local search to evolve priority rules for scheduling jobs on a machine with time-varying capacity. Swarm and Evolutionary Computation (2021)
12. Gil-Gala, F.J., Varela, R.: Genetic algorithm to evolve ensembles of rules for on-line scheduling on single machine with variable capacity. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Toledo Moreo, J., Adeli, H. (eds.) From Bioinspired Systems and Biomedical Applications to Machine Learning. 223–233. Springer International Publishing, Cham (2019)
13. Hart, E., Sim, K.: A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation 24(4), 609–635 (2016)
14. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
15. McMenemy, P., Veerapen, N., Adair, J., Ochoa, G.: Rigorous performance analysis of state-of-the-art tsp heuristic solvers. In: Liefooghe, A., Paquete, L. (eds.) Evolutionary Computation in Combinatorial Optimization. pp. 99–114. Springer International Publishing, Cham (2019)
16. Nguyen, S., Mei, Y., Xue, B., Zhang, M.: A hybrid genetic programming algorithm for automated design of dispatching rules. Evolutionary Computation 27(3), 467–496 (2019)
17. Park, J., Mei, Y., Nguyen, S., Chen, A., Johnston, M., Zhang, M.: Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches. In: GECCO'16: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. 109—110 (2016)
18. Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M.: An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. Applied Soft Computing 63, 72–86 (2018)