Leveraging Symbolic Regression for Heuristic Design in the Traveling Thief Problem

Andrew Ni[1][0000−0003−3480−4536] and Lee Spector[2,3][10000−0001−5299−4797]

[1] Amherst College, Amherst MA 01002, USA `hickory62133@gmail.com`
[2] Amherst College, Amherst, MA 01002, USA
[3] University of Massachusetts Amherst, Amherst, MA 01003, USA
`lspector@amherst.edu`

Abstract. The Traveling Thief Problem is an NP-hard combination of the well known traveling salesman and knapsack packing problems. In this paper, we use symbolic regression to learn useful features of near-optimal packing plans, which we then use to design efficient metaheuristic genetic algorithms for the traveling thief algorithm. By using symbolic regression again to initialize the metaheuristic GA with near-optimal individuals, we are able to design a fast, interpretable, and effective packing initialization scheme. Comparisons against previous initialization schemes validates our algorithm design.

Keywords: Symbolic Regression, Traveling Thief Problem, Combinatorial Optimization, Heuristics, Local Search

1 Introduction

The Traveling Thief Problem (TTP) is an NP-hard combination of the Traveling Salesman (TSP) and 0-1 Knapsack Packing (KP) problems designed to be more representative of real life situations[19, 2]. In a TTP problem, a thief is presented with a set of n cities Cities $= \{i, \cdots, n\}$ and a distance function $d(i, j) : \mathbb{Z}^2 \to \mathbb{Z}$, where each city i except the first one has m items with weight w_{ik} and profit p_{ik}. The thief has to visit every city in the given set, starting and ending at city 1. At each city, they may put any of the available items into their knapsack, which has capacity W. However, the more full their knapsack is, the slower they travel, and a renting cost R is incurred for each time unit spent traveling. Therefore, for a given tour $[c_1, \cdots c_n, c_1]$ and packing plan $\texttt{packed}(i, k) : \{1...n\} \times \{1...m\} \to \{0, 1\}$, where 0 denotes an unpacked item and 1 denotes a packed item, their final objective value will be

$$\sum_{i=1}^{n} \sum_{k=1}^{m} \texttt{packed}(i, k) * p_{ik} - R \times \left(\frac{d(n, 1)}{v_{\max} - \nu W_n} + \sum_{i=1}^{n-1} \frac{d(i, i+1)}{v_{\max} - \nu W_i} \right)$$

where $v_{\max}$ and $v_{\min}$ are the maximum and minimum travel speeds, respectively, $\nu = \frac{v_{\max} - v_{\min}}{W}$ is the decrease in speed per unit weight, and $W_i = \sum_{i'=0}^{i} \sum_{k=1}^{m} \texttt{packed}(i', k) * w_{ik}$ is the total weight of the knapsack at the ith city.

In this paper, we use symbolic regression to design packing heuristics for the traveling thief problem that achieve a better "fit" than previous human-designed heuristics. We first show that there is a smooth, regular relationship between the standardized item profitability ratio (IPR) and the standardized distance to the end of the tour (rDist) of items packed in high-quality packing plans. We then use symbolic regression to discover combinations of the two features IPR and rDist that can closely model the abovementioned relationship. Identification of promising feature combinations then gives rise to a family of metaheuristic genetic algorithms, in which each individual is roughly a vector of the coefficients for these combinations and defines a boundary line separating packed items from unpacked items. By again utilizing symbolic regression to analyze the optimal individuals produced from our metaheuristic genetic algorithm, we are able to directly predict the parameter values of the optimal individual to high precision, greatly accelerating the GA and stabilizing its performance. In this way, we obtain a family of effective and high-performing packing initialization heuristics for the traveling thief problem. We perform experiments to choose the best initialization scheme from this family, and then demonstrate the superior speed and problem-solving performance of our heuristic compared to the previous SOTA.

Our paper is organized as follows: In Section 2, we give a brief overview of related work, including previous TTP solvers, and especially previous initialization heuristics. In Section 3 we gradually infer our initialization heuristic by characterizing optimal packing solutions to the traveling thief problem. Finally, in Section 4, we describe our experiments and results, comparing our heuristics both among themselves and against previous initialization heuristics. Empirical results show that our heuristics are a better "fit" for TTP instances, usually achieving higher objective values in fewer objective value evaluations than previous methods.

2   Background and Related Work

2.1   TTP Benchmark Suite

In the benchmark suite proposed by Polyakovskiy et al.[19], a TTP instance starts off as one of the TSP instances from the TSPLIB library[8, 20]. An item factor F is chosen from the set $\{1, 3, 5, 10\}$, and F items are then placed at each city other than the first one. The weights and profits of these items are determined by the knapsack problem type, which can be one of `uncorr`, `uncorr_similar_weights`, and `bounded_strongly_corr`, in order of increasing difficulty for KP-solvers[13]. In the `uncorr` category, weights and profits are sampled uniformly from $[1, 1000]$. In the `uncorr_similar_weights` category, they are sampled from $[1000, 1010]$ and $[1, 1000]$, respectively. Finally, in the `bounded_strongly_corr` category, the weight of the kth item in the ith city w_{ik} is sampled uniformly from $[1, 1000]$ and the profit is determined based on this weight as $p_{ik} = 100 + w_{ik}$. Based on the near-optimal solutions to the TSP and KP instances generated by the Chained Lin-Kernighan heuristic[22, 1] and the COMBO solver[13] respectively, a renting ratio is calculated as the

optimal profit of the KP subproblem divided by the time taken to traverse the linkern tour while collecting the items in that optimal packing. In other words, the optimal KP packing combined with the linkern tour will give a zero TTP objective value. This prevents one subproblem dominating the other, and forces solvers to focus on both subproblems to achieve high objective values[19]. Finally, the knapsack capacity is determined based on a capacity factor $C \in \{1...10\}$ as $W = \frac{C}{11} \sum_{i=1}^{n} \sum_{k=0}^{n} w_{ik}$. Altogether this gives $4 \times 3 \times 10 = 120$ TTP instances for each TSP instance.

2.2   TTP Solvers

Since the initial proposal of the traveling thief problem[2] and the creation of the TTP benchmark suite[19], many approaches for solving the problem have been developed. Polyakovskiy et al.[19] generate a tour using the chained lin-kernighan heuristic[22] and then apply either a hill-climbing local search (RLS), a (1+1) EA, or a simple heuristic (SH) to generate a good packing plan for that tour. While the EA and RLS methods perform well on smaller instances, SH takes much less time to run and performs the best on the largest TTP instances, illustrating the importance of a good initialization heuristic.

After some time, a large-scale benchmark was conducted of all proposed TTP solvers, and a new algorithm was proposed, which attempts to choose the best previously proposed solver based on information about the specific TTP instance. Besides the aforementioned RLS, EA, and SH algorithms, benchmarked algorithms included DH[3], MATLS[14], S1-S5 and C1-C5[7], CS2SA[?], and M3/M3B and M4/M4B[26].

The density-based heuristic DH and the CoSolver algorithm were proposed by Bonyandi et al.[3]. In the DH heuristic, similarly to SH, items are assigned scores based on the change in objective value obtained when going from a zero packing plan to one which only picks that item. Items are then picked in decreasing order of their scores until no further improvement in objective value can be obtained. On the other hand, CoSolver attempts to decompose the TTP problem into multiple subproblems, interleaving tour improvement for a given packing plan with packing improvement for a given tour. These algorithms were then improved to give the solvers CS2SA[5], an improvement to CoSolver with 2-Opt and Simulated Annealing, MA2B[5] a population-based memetic algorithm with MPX-based crossover[15], and CS2SA* and CS2SA-R[6], two improvements to the CS2SA algorithm.

The Memetic Algorithm with Two-Stage Local Search[14] follows a similar approach, alternating between tour and packing improvement. They also develop a novel packing initialization heuristic `Insertion`, using three different approximations to the change in objective value upon packing an item. The first approximation is the same as that used in DH. The second approximation is a worst-case approximation assuming that the knapsack is already full. Finally, their third approximation is an expected-value approximation assuming that the knapsack weight is distributed evenly along the tour.

Faulkner et al.[7] propose several algorithms mainly based on their new heuristic `packIterative`. In `packIterative`, items are give scores based on their profit, weight, and remaining distance as $\text{score}_{ik} = \frac{p_{ik}^{\alpha}}{w_{ik}^{\alpha} d_i}$, where the parameter α is optimized in an outer loop based on a binary-search-like algorithm. Items are collected in decreasing order of their scores until no further objective increase is possible. However, to decrease the number of costly objective value computations, they propose to pack some fraction of the total items at each step, which in practice is generally 1%[45] If at any point the objective value decreases, they revert their changes and halve the step size. This proceeds until no improvement is possible or the step size reaches 1.

In a different direction, Wagner et al.[26] pursue swarm-intelligence-based TTP solvers with their M3, M3B, M4, and M4B solvers. For a given packing plan, they use an ant colony optimization algorithm to improve the tour. While their algorithm does not scale well to large instances, it finds very good solutions on smaller instances.

Finally, Majid et al.[16] have recently proposed a cooperative coordination heuristic PGCH for simultaneously improving the tour and the packing plan, as well as a directed local search algorithm MBFS for quickly improving packing plans. They show that this approach yields better objective values than previous SOTA algorithms, obtaining record objective values in most large problem instances studied.

Many of these previous approaches often start and restart by generating one or more decent individuals by generating a good tour, either using the lin-kernighan heuristic[22, 1] or something faster but less optimal like the quick-boruvka algorithm[1], and then generating an initial packing for that individual using a human-designed heuristic like `packIterative`[7] or `Insertion`[14]. These heuristics generally compute a utility score for each item using a parameterized function designed to capture human intuition about which items are likely to be packed in an optimal solution. Items are then packed in decreasing order of their utility scores until no improvement in the objective value is possible. However, these human-designed heuristics are not very flexible, as they only have zero or one learnable parameters, and if the "shape" of their score functions do not match the "shape" of high quality packing plans, then their maximum achievable objective score may be reduced. In contrast, we propose directly learning appropriate score function "shape"s from the problem instances themselves by using symbolic regression to analyze near-optimal packing plans. This gives our score functions a better "fit" to the problem at hand and results in higher objective scores.

2.3   Symbolic Regression

Symbolic regression[10] is a subfield of Evolutionary Computation concerned with evolving mathematical functions to fit some dataset. Due to their symbolic

---

nature and simplicity, Symbolic regression methods are more interpretable than neural net-based models[11]. In addition, large-scale testing on the Penn Machine Learning Benchmark (PMLB)[21] has shown that SR-based algorithms are competitive with if not better than other methods in terms of accuracy[11]. Many SR methods[4, 12, 25, 23], including the first system introduced by Koza[10], are based on genetic programming (GP), evolving populations of executable data structures, commonly parse trees, that take a number of inputs and produce a numerical output. In these parse trees, the nodes are usually mathematical functions and the leaves are input features or numerical constants. Other SR approaches include methods inspired by physics[24], based on Bayesian optimization[9], or utilizing reinforcement learning algorithms[18]. For the sake of convenience, we reuse a previous system, the gplearn system[23] with DALex[17] as the selection method, which has been shown to be fast and have good accuracy on SR problems. However, as many of the conclusions we draw are symbolic and qualitative in nature, we believe that the same results can be obtained with other SR systems as well.

3   SR-guided Heuristic Design

In this section we present our proposed packing initialization heuristics through a careful characterization of near-optimal packing plans.

3.1   Analysis of Optimal Packing Plans

We first study the relationship between the standardized item profitability ratio $\text{IPR}_{ik} = \frac{p_{ik}}{w_{ik}}$ and distance-to-go $\text{rDist}_{ik} = d(n, 1) + \sum_{i'=i}^{n-1} d(i', i' + 1)$ of items collected versus uncollected in near-optimal packing plans. Due to the large ourliers in the IPR when items have very low weights, we standardize to zero median M and unit median absolute deviation $\text{MAD}(\{\text{IPR}_{ik}\}) = \text{median}(\{|\text{IPR}_{ik} - M)|\})$ instead of zero mean and unit variance. In addition, as these outliers are always packed due to their light weight and high profit, we limit our attention to IPR and rDist values in the interval $[-2, 2]$. To obtain near-optimal packing plans, we adopt the method proposed in Polyakovskiy et al.[19], generating a tour using the chained lin-kernighan heuristic[22] and using a $(1+1)$ hill-climbing EA to optimize the packing plan, starting from a packing plan that does not pick any items. To get high quality solutions, we run this EA for 10^6 generations. At the end of the algorithm, we record for each item the standardized IPR and rDist, along with whether it was packed in the best packing plan found. Due to space limitations, we only show a few results from selected TTP instances in figure1.

While combinatorial optimization problems like the traveling thief problems are inherently discrete, the plots in figure 1 show a smooth boundary demarcating the packed items from the unpacked items. Of course, the degree of discontinuity is higher for smaller instances, so we hypothesize that the heuristics developed
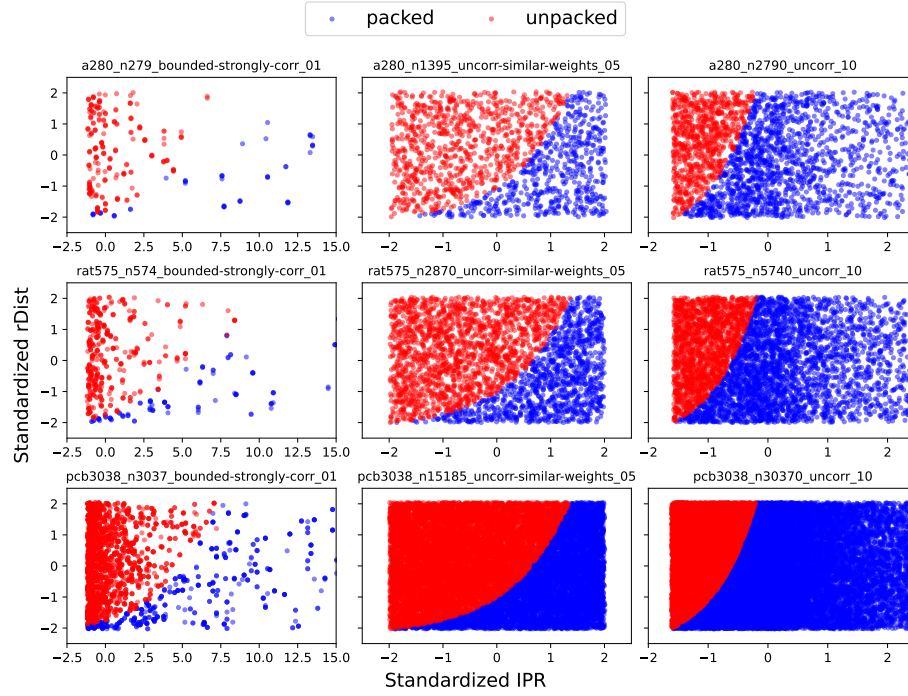
Fig. 1: Plot of the packed and unpacked items in near-optimal tours generated by the (1+1) EA against the normalized item profitability ratio and distance to the end of the tour. A smooth nonlinear boundary can be seen separating packed items from unpacked items. While the specific boundary varies from instance to instance, it keeps the same overall curved shape.

in this paper will work better with larger problem instances. This nicely complements the behavior of packing improvement, which can find optimal solutions to small instances very quickly but may take a long time for larger problems. Heuristic algorithms that assign scores to items based on their profitability ratio and remaining distance essentially draw boundary lines in this space and collect all items on one side of the line. While we do not expect the boundary line that best fits the near-optimal (1+1) EA solutions to also produce the best packing plans as an initialization heuristic, we hypothesize that the shape of the boundary line is a fundamental aspect of the traveling thief problem. Therefore, heuristics that can better match the shape of these near-optimal packing plans have a better "fit" for TTP problems, and are more likely to be able to produce packing plans with very high objectives.

To learn the shape of these boundary lines, we train a nonlinear binary classifier (NLBC) to distinguish betwen packed and unpacked items using symbolic regression. Due to computational constraints, we choose a range of smaller TTP instances with roughly log-spaced sizes: `a280`, `rat575`, `rl1323`, and `pcb3038`. From each category we take 90 problems with capacity factors $C \in \{1...10\}$, knapsack type in {`bounded_strongly_corr`, `uncorr_similar_weights`,`uncorr`}, and item factor $F \in \{1, 5, 10\}$.

Instead of using the absolute error fitness function as in many SR systems[23, 12, 11], we use the binary cross entropy (BCE) loss $\mathcal{L}(y_i, \hat{y}_i) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i)$ as our fitness function, where $y_i \in \{0, 1\}$ is the ground-truth label and $\hat{y}_i = \frac{1}{1+e^{-x_i}}$, where x_i is the output of the SR individual on the ith training example. We use the DALex[17] selection algorithm because it is fast and produces solutions with good accuracy. We keep the other hyperparameters unchanged from the defaults in gplearn[6] except that we reduce the number of generations to 300 and combine the results from 5 independent runs.

At each generation, we take the individual with the best BCE loss, expand it into a sum of terms using `sympy`, remove the constant coefficient from each term to get a set of features, and count the number of features. Due to computational limitations, we only apply this process to the best individual in each generation. We say that an individual dominates another individual if it has a better BCE loss and a number of terms equal to or less than that of the other individual. Throughout each run, we keep a pareto front of non-dominated best-of-generation individuals, and at the end of each run we collect the feature set of each individual in the final pareto front. We group all of the individuals collected this way from all 5 independent runs, and count the number of identical feature sets at each feature set size. To save space, we only report feature sets that appeared more than once. The results are displayed in table 1.

From this table, we select a few promising feature sets. In the size 3, and 4 groups, the most frequent feature set occurs much more often than the runner-up, so we choose the most frequent feature set from each group. Since the two most frequent sets in the size 5 group appear a relatively similar number of times, we choose both of them. Although the feature sets in the size

---

[6] https://github.com/trevorstephens/gplearn

Table 1: Frequency of feature sets in the pareto-fronts of evolved NLBCs, grouped by feature set size.

size 2		size 3		size 4		size 5	
$\{x_0,x_1\}$	108	$\{1,x_0,x_1\}$	213	$\{1,x_0,x_1,x_0x_1\}$	158	$\{1,x_0,x_1,x_0x_1,x_0^2\}$	77
$\{1,x_0\}$	57	$\{x_0,x_1,x_0x_1\}$	10	$\{1,x_0,x_1,x_0^2\}$	53	$\{1,x_0,x_1,x_0x_1,x_1^2\}$	46
$\{1,x_1\}$	5	$\{1,x_0,x_0x_1\}$	10	$\{1,x_0,x_1,x_1^2\}$	6	$\{1,x_0,x_1,x_0^2,x_0^3\}$	19

size 6		size 7	
$\{1,x_0,x_1,x_0x_1,x_1^2,x_0x_1^2\}$	13	$\{1,x_0,x_1,x_0x_1,x_1^2,x_0x_1^2,x_1^3\}$	5
$\{1,x_0,x_1,x_0x_1,x_0^2,x_1^2\}$	11	$\{1,x_0,x_1,x_0x_1,x_1^2,x_0^2x_1,x_0x_1^2\}$	4
$\{1,x_0,x_1,x_0x_1,x_1^2,x_0^3\}$	10	$\{1,x_0,x_1,x_0x_1,x_0^2,x_1^2,x_0x_1^2\}$	3

6 and 7 groups all rarely appear and are all fairly similar, we also choose the $\{1,x_0,x_1,x_0x_1,x_0^2,x_1^2\}$ feature set from that group. In this way, we identify 5 different feature sets, which we name based on the number of terms they contain: 2T $= \{x_0,x_1\}$, 3T $= \{1,x_0,x_1\}$, 4T $= \{1,x_0,x_1,x_0x_1\}$, 5T0 $= \{1,x_0,x_1,x_0x_1,x_0^2\}$, 5T1 $= \{1,x_0,x_1,x_0x_1,x_1^2\}$, and 6T $= \{1,x_0,x_1,x_0x_1,x_0^2,x_1^2\}$.

3.2   Analysis of Optimal Parameter Values

Each feature set $\{f_i\}$ identified in the previous section gives rise to a metaheuristic algorithm, where individuals of the form $[w_0,\cdots,w_n]$ encode boundary lines of the form $\sum_{i=0}^{n} w_i f_i = 0$ and pack all items such that $\sum_{i=0}^{n} w_i f_i > 0$. We use a (1+1) EA, where the initial individual is the 0 vector and mutations are standard normal perturbations $\sim \mathcal{N}(0,1)$. To obtain near-optimal objective values, we conduct 4 independent runs of our EA for 10^5 generations each. In preliminary experiments, we found that the optimal values of the w_i were all relatively stable except for the "bias" term corresponding to the feature 1, which would vary significantly depending on the quality of the tour. Since for a given TTP instance and TSP tour there is a 1-to-1 correspondence between the value of the "bias" term and the total weight packed by the plan, we replace the bias term with a "percent" term describing the packed capacity as a percentage of the knapsack capacity, which we find to be much more stable. Therefore, instead of directly encoding a boundary line, each individual is now of the form $[w_0,\cdots,w_n,p]$ and encodes a score function $\texttt{score} = \sum_{i=0}^{n} w_i f_i$. Items are then collected in decreasing order of their scores until reaching a total weight of pW, where W is the knapsack capacity.

Due to this removal of the 1 term, the 2T and 3T feature sets are no longer distinct, so now our collection of feature sets becomes: 3T $= \{x_0,x_1\}$, 4T $= \{x_0,x_1,x_0x_1\}$, 5T0 $= \{x_0,x_1,x_0x_1,x_0^2\}$, 5T1 $= \{x_0,x_1,x_0x_1,x_1^2\}$, and 6T $= \{x_0,x_1,x_0x_1,x_0^2,x_1^2\}$. Finally, as the percent term has to be in the interval $[0,1]$, we mutate it with noise sampled according to $\mathcal{N}(0,0.1)$ and clamp it to the interval $[0,1]$.

After conducting the 4 metaheuristic algorithm runs for a given feature set and a given TTP instance, we record the best individual across all runs

$[w_0, \cdots, w_n, p]$. However, since scaling all the w_i's of an individual by a constant does not change its performance, we first normalize the genotype so that $\sum_{i=0}^{n} w_i^2 = 1$. We then use symbolic regression again to predict these near-optimal individuals' genotype using 7 features of the TTP instance: the number of cities x_0, the number of items x_1, the renting ratio x_2, the knapsack capacity x_3, the item factor x_4, the maximum speed x_5, the minimum speed x_6, and the capacity factor x_7. Note that the names of these input variables are distinct from the names of the variables in the aformentioned feature sets, which were aliases for the standardized `IPR` and `rDist`.

We use the same TTP problems as in the previous section, comprising 90 instances each from the categories `a280`, `rat575`, `rl1323`, and `pcb3038`. However, we separate them based on their knapsack type, and conduct separate runs for problems in each knapsack type. Since our SR system is only built for single-valued outputs, we conduct a separate run for each index in the meta-heuristic genotype vector. We use the same system as in the previous section, except this time we're interested in eliminating useless input variables rather than constructing useful features. Therefore, instead of using sympy to simplify best-of-generation individuals, we instead track a rough approximation of the pareto front, keeping the best individuals with program lengths less than 20, 30, and 40. We also increase the number of parallel runs to 10, which makes for a total of up to 30 distinct individuals obtained from each set of 10 runs. For each input variable, we then calculate what percentage of these individuals contain that input variable in their programs. However, since $v_{\max}$ and $v_{\min}$ remain the same throughout all instances in the TTP benchmark set[19], we treat them as constants and omit them from consideration. Due to space considerations, we only display the results for the `4T` feature set in table 2, and put the other results in the appendix[7].

Table 2: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `4T` metaheuristic GA. The highest value for each parameter-KP type combination is shown in bold. Almost all solutions contain the x_7 variable, whereas other variables are usually present in less than one third of the solutions.

	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.17	0.37	0.07	0.4	**1.0**	0.27	0.1	0.17	0.17	**1.0**	0.2	0.13	0.07	0.03	**0.97**
$w1$	0.37	0.17	0.03	0.43	**1.0**	0.1	0.0	0.0	0.1	**1.0**	0.07	0.0	0.18	0.0	**1.0**
$w2$	0.1	0.17	0.03	0.9	**1.0**	0.57	0.5	0.07	0.6	**1.0**	0.4	0.1	0.07	0.5	**0.93**
percent	0.1	0.1	0.1	0.13	**1.0**	0.0	0.0	0.1	0.03	**1.0**	0.03	0.03	0.0	0.0	**1.0**

Since the capacity factor variable x_7 was present in almost all solutions, and other variables are only present in a few solutions, we hypothesize that

---

[7] Temporarily at https://github.com/andrewni420/SRHeuristic

the optimal parameter values depend solely on the capacity factor. To verify this hypothesis, evolved solutions using the same symbolic regression setup as before, but now restricting the set of input features to the x_7 variable. We then take the individual with the lowest mean absolute error, and plot it against the capacity factor, along with the rest of our dataset. We encountered two anomalous solutions, one for the w_2 parameter of the 5T0 feature set in the uncorr KP type, and one for the w_3 parameter of the 6T feature set. The graphs of these solutions did not match their high fitness, which we hypothesize is due to differences in handling edge conditions like dividing by zero. So, we replaced them with a linear fit. Due to limited space, we again only show the results for the 4T feature set, and put the rest in the appendix.

From figure 2, we can see that the optimal parameter values are almost entirely dependent on the capacity factor, and that our evolved solutions are a good fit.

3.3   Heuristic Design

By using our SR evolved solutions to predict the optimal value of each parameter in an individual's genotype, we can speed up our metaheuristic algorithm to the point that it becomes a competitive packing initialization heuristic. Our heuristic design is as follows: for each feature set 3T...6T, we predict the value of each parameter $w_0...w_n,p$ using our SR solutions. For each individual with genotype $[w_0, ..., w_n, p]$, we sort the items by their score function, the dot product of the weight vector $[w_0, \cdots, w_n]$ and the feature vector $[f_0, \cdots, f_n]$ from the feature set. For example, when using the 4T feature set, this feature vector would be $[x_0, x_1, x_0 * x_1]$, where x_0 and x_1 are the standardized IPR and rDist respectively. We then pack items in decreasing order of their score function until the objective value reaches a maximum. However, in contrast to algorithms like packIterative[7], which have to start from zero packed items, we already have a good guess of the optimal knapsack weight. Therefore, we start by packing items until reaching a capacity of pW. We then search for the optimal packing in both directions (more items packed vs less items packed) as follows: we first take 1 step forward, packing/unpacking a single item depending on the direction. As long as the objective value increases, we double the step size. However, if the objective value decreases, we revert the changes, unpacking/packing the items, and decrease the step size by a factor of 8. The constants 2 and 8, as well as the initial step size of 1 were chosen through a preliminary hyperparameter search to minimize the number of costly objective value calculations performed by our heuristic.

After determining the maximum objective value for the individual, we mutate its weights $w_0, ..., w_n$ in a $(1+1)$ EA with gaussian noise sampled from $\mathcal{N}(0, 0.5)$. Again, the mutation strength was chosen through a preliminary search to achieve the highest objective value in the fewest generations. Instead of mutating the percent parameter p, we keep a running estimate of p based on the optimal knapsack weights of evaluated individuals. To stabilize estimates while still adapting quickly, we test whether the new sample p_i differs from the estimate $\hat{p}$ by more
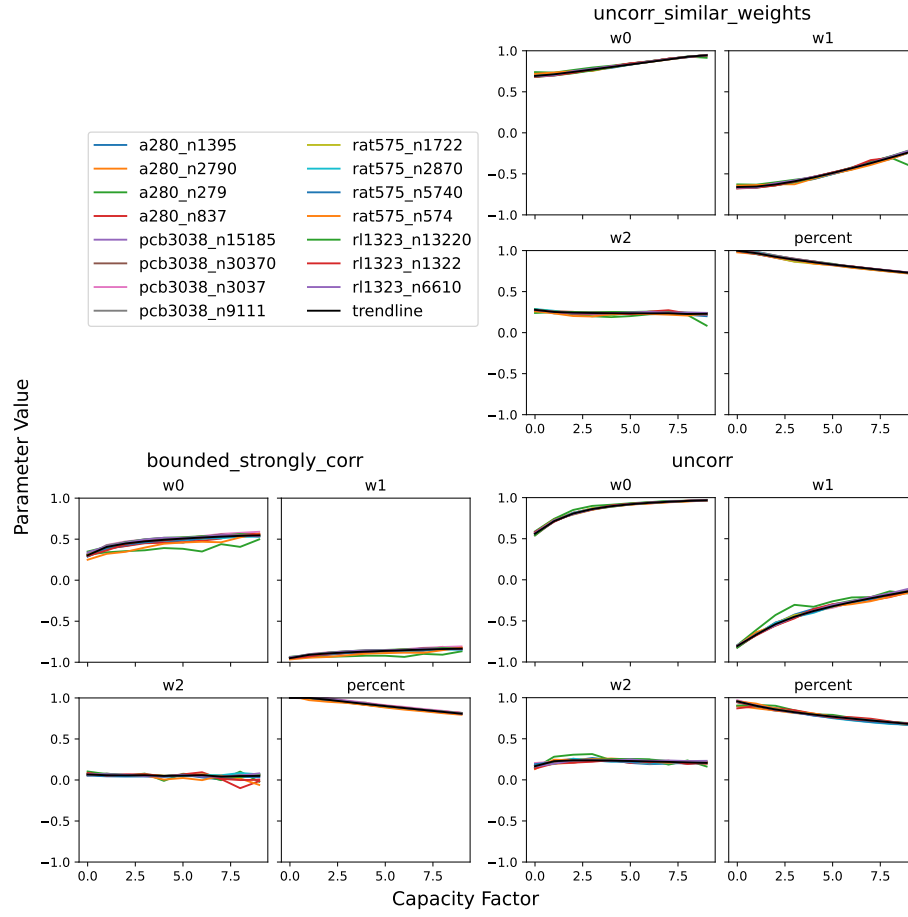
Fig. 2: Plot of the optimal parameter values found by our metaheuristic GA in each training instance against the capacity factor, along with a line of best fit. Parameter values were normalized such that $w_0^2 + w_1^2 + w_2^2 = 1$.

than 10%. If so, then we take the average of the two for more stability $\hat{p} \leftarrow \frac{p_i + \hat{p}}{2}$. If not, then we simply replace the old estimate with the new sample $\hat{p} \leftarrow p_i$. This scheme was also found to reduce the number of objective value calculations in preliminary experiments.

4 Experiments and Results

Having designed a family of initialization heuristics, one for each feature set `3T...6T`, we now compare the heuristics both against each other and against previous SOTA heuristics. For these experiments, we choose 5 previously unseen TSP instances of roughly log-spaced size, including both the smallest and largest instances: `eil51`, `lin318`, `d2103`, `usa13059`, and `pla85900`. As before, we take 90 TTP instances from each category, with capacity factors $C \in [1...10]$, KP type in `bounded_strongly_corr`, `uncorr_similar_weights`, and `uncorr`, and item factor $F \in \{1, 5, 10\}$ for a total of 450 TTP instances. We report both the final objective value obtained by the heuristic and the number of objective value calculations, which is a proxy for the computational cost that is independent of programming language and computer hardware. We first compare our heuristics against each other to determine the best one, and then rerun the best heuristic on all benchmark instances and compare to previous initialization schemes.

4.1 Internal Comparisons

In this section we compare our proposed heuristics against each other to determine which one performs the best. We evaluate each heuristic algorithm 30 times on each TTP instance, and each time we rank the algorithms in terms of their objective values and number of evaluations. For an equal comparison, we share the same starting tour between all heuristics. The results are shown in figure 3

The results show that the `6T` heuristic has the highest frequency of best objective value, but the `4T` heuristic often requires the fewest number of evaluations. However, as the computational cost of the later optimization schemes often outweighs the cost of the initialization heuristic in practical TTP solvers, we choose the `6T` heuristic to compare against SOTA heuristics.

4.2 External Comparisons

In this section we compare the `6T` heuristic against initialization heuristics from the literature. Specifically, since `Insertion`[14] is an improvement upon the similar `DH`[3] and `SH`[19] heuristics, we compare against `Insertion` and `packIterative`[7] . As before, we run our heuristic algorithm 30 times on each TTP instance and share the same tour between all algorithms. Since the other three heuristics are deterministic, they are run only once to determine their performance across all 30 trials. At each trial we rank the algorithms in terms of the objective value and number of evaluations. The results are shown in figure 4
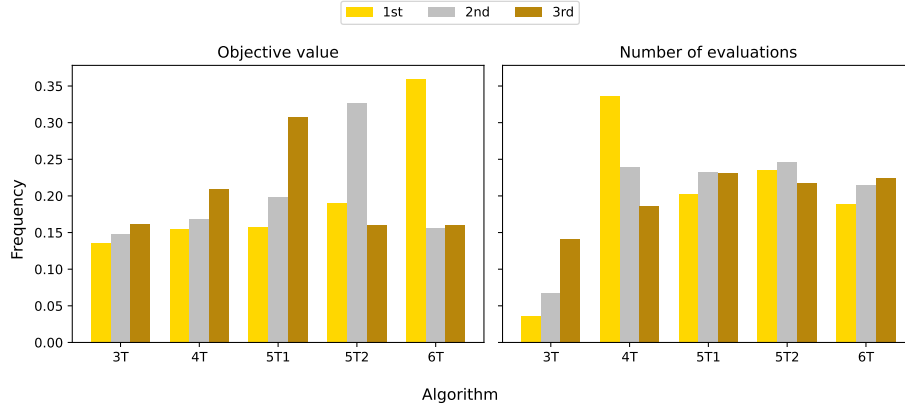
Fig. 3: Ranking results for all heuristics on all 450 instances. The `6T` heuristic is has the best objective most frequently, but the `4T` algorithm generally requires fewer objective value computations.
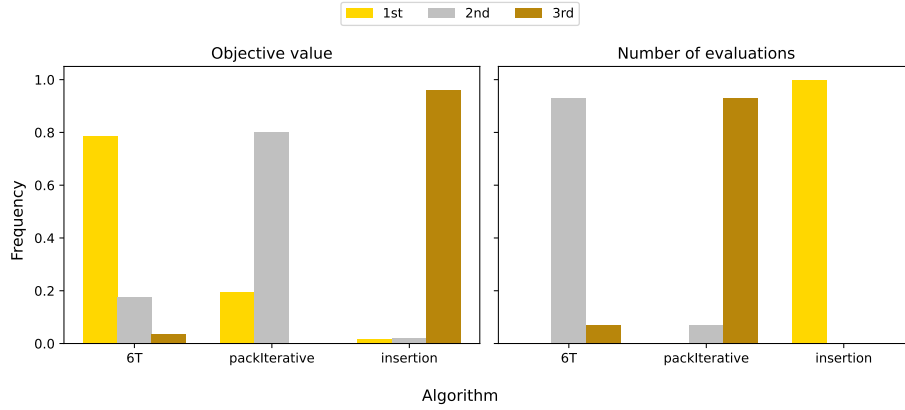
Fig. 4: Ranking results for all heuristics on all 450 instances. `6T` is often the best in terms of objective value, and generally requires fewer objective value computations than `packIterative`, but is slower than `Insertion`.

5   Conclusion and Future Work

In this work, we presented a set of 5 novel packing initialization heuristics for the traveling thief problem. Rather than being designed by human intuition, these heuristics were derived from the interpretable modeling of near-optimal packing plans using symbolic regression. Experimental results on a large range of problem instances validate the effectiveness of our approach, and we demonstrate improved objective values compared to previous SOTA initialization heuristics. While we have focused on comparisons between heuristics in this paper, we do not anticipate that the best algorithms will utilize only a single initialization heuristic. In fact, even though our heuristic almost always achieves a higher objective value than the `Insertion` heuristic, `Insertion` has such a low cost that in practice it makes sense to always run `Insertion` in addition to our heuristic, similarly to Majid et al.[16], on the off chance that `Insertion` will perform better. In a more complicated scheme, we could also learn to predict which of the proposed heuristics will perform the best on a given TTP instance from various features of the instance, similarly to Wagner et al.[27].

In addition, while we have evolved separate functions to predict the optimal parameter values for each KP type, we hypothesize that more general functions can be evolved that use more fundamental properties of the TTP instances, for example, the covariance matrix of the joint distribution of weights and profits. Therefore, future work could expand the current benchmark set of TTP instances to include other distributions of weights and profits, and use this additional training data to evolve a single function for each parameter that will work across all TTP instances.

6   Acknowledgements

References

1. Applegate, D., Cook, W., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. INFORMS Journal on Computing **15**(1), 82–92 (2003). https://doi.org/10.1287/ijoc.15.1.82.15157
2. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: 2013 IEEE Congress on Evolutionary Computation. pp. 1037–1044 (2013). https://doi.org/10.1109/CEC.2013.6557681
3. Bonyadi, M.R., Michalewicz, Z., Przybylek, M.R., Wierzbicki, A.: Socially inspired algorithms for the travelling thief problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. p. 421–428. GECCO'14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2576768.2598367
4. Burlacu, B., Kronberger, G., Kommenda, M.: Operon c++: An efficient genetic programming framework for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. p. 1562–1570. GECCO'20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3377929.3398099
5. El Yafrani, M., Ahiod, B.: Population-based vs. single-solution heuristics for the travelling thief problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. p. 317–324. GECCO'16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2908812.2908847
6. El Yafrani, M., Ahiod, B.: Efficiently solving the traveling thief problem using hill climbing and simulated annealing. Information Sciences **432**, 231–244 (2018). https://doi.org/10.1016/j.ins.2017.12.011
7. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. p. 385–392. GECCO'15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2739480.2754716
8. Gerhard Reinelt: Tsplib, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, [Online; accessed 11-April-2024]
9. Jin, Y., Fu, W., Kang, J., Guo, J., Guo, J.: Bayesian symbolic regression (2020)
10. KOZA, J.: On the programming of computers by means of natural selection. Genetic Programming (1992), https://cir.nii.ac.jp/crid/1573105975012577408
11. La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.: Contemporary symbolic regression methods and their relative performance. In: Vanschoren, J., Yeung, S. (eds.) *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks.* vol. 1. Curran (2021)
12. La Cava, W., Spector, L., Danai, K.: Epsilon-lexicase selection for regression. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016.* p. 741–748. GECCO'16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2908812.2908898
13. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science **45**(3), 414–424 (Mar 1999). https://doi.org/10.1287/mnsc.45.3.414
14. Mei, Y., Li, X., Yao, X.: Improving efficiency of heuristics for the large scale traveling thief problem. In: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui,

L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) Simulated Evolution and Learning. pp. 631–643. Springer International Publishing, Cham (2014)

15. Mühlenbein, H.: Evolution in time and space – the parallel genetic algorithm. Foundations of Genetic Algorithms, vol. 1, pp. 316–337. Elsevier (1991). https://doi.org/https://doi.org/10.1016/B978-0-08-050684-5.50023-9

16. Namazi, M., Newton, M.A.H., Sanderson, C., Sattar, A.: Solving travelling thief problems using coordination based methods. Journal of Heuristics **29**(4), 487–544 (Dec 2023). https://doi.org/10.1007/s10732-023-09518-7

17. Ni, A., Ding, L., Spector, L.: Dalex: Lexicase-like selection via diverse aggregation. In: Giacobini, M., Xue, B., Manzoni, L. (eds.) Genetic Programming. pp. 90–107. Springer Nature Switzerland, Cham (2024)

18. Petersen, B.K., Landajuela, M., Mundhenk, T.N., Santiago, C.P., Kim, S.K., Kim, J.T.: Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients (2021)

19. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. p. 477–484. GECCO'14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2576768.2598249

20. Reinelt, G.: A traveling salesman problem library. ORSA Journal on Computing **3**, 376–384 (1991)

21. Romano, J.D., Le, T.T., Cava, W.L., Gregg, J.T., Goldberg, D.J., Ray, N.L., Chakraborty, P., Himmelstein, D., Fu, W., Moore, J.H.: Pmlb v1.0: An open source dataset collection for benchmarking machine learning methods (2021). https://doi.org/10.48550/arXiv.2012.00058

22. S. Lin, B.W.K.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research **21**, 498–516 (1973)

23. Stephens, T.: gplearn. https://github.com/trevorstephens/gplearn (2023)

24. Udrescu, S.M., Tegmark, M.: Ai feynman: A physics-inspired method for symbolic regression. Science Advances **6**(16), eaay2631 (2020). https://doi.org/10.1126/sciadv.aay2631

25. Virgolin, M., Alderliesten, T., Bosman, P.A.N.: Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 1084–1092. GECCO'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3321707.3321758

26. Wagner, M.: Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In: Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., Stützle, T. (eds.) Swarm Intelligence. pp. 273–281. Springer International Publishing, Cham (2016)

27. Wagner, M., Lindauer, M., Mısır, M., Nallaperuma, S., Hutter, F.: A case study of algorithm selection for the traveling thief problem. Journal of Heuristics **24**(3), 295–320 (Jun 2018). https://doi.org/10.1007/s10732-017-9328-y

7   Frequency of Input Variables in Evolved Solutions

In this section we present the rest of the results of the experiment described in section 3.2. Shown below are the occurrence frequencies of each input variable in

functions evolved to predict the optimal value of a parameter in a metaheuristic GA. The conclusions are the same across the board: the x_7 (capacity factor) variable is essential, and the other input variables are irrelevant, except for maybe the x_4 (item factor) variable, which is occasionally present in a large fraction of solutions.

Table 3: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `3T` metaheuristic GA.

	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.4	0.3	0.17	0.4	**1.0**	0.1	0.47	0.07	0.3	**1.0**	0.02	0.05	0.18	0.05	**1.0**
$w1$	0.27	0.45	0.03	0.38	**1.0**	0.18	0.33	0.1	0.28	**1.0**	0.15	0.13	0.18	0.05	**1.0**
percent	0.03	0.07	0.0	0.05	**1.0**	0.1	0.12	0.18	0.08	**0.93**	0.28	0.12	0.07	0.2	**1.0**

Table 4: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `4T` metaheuristic GA. These results were also shown in section 3.3 of the paper

	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.17	0.37	0.07	0.4	**1.0**	0.27	0.1	0.17	0.17	**1.0**	0.2	0.13	0.07	0.03	**0.97**
$w1$	0.37	0.17	0.03	0.43	**1.0**	0.1	0.0	0.0	0.1	**1.0**	0.07	0.0	0.18	0.0	**1.0**
$w2$	0.1	0.17	0.03	0.9	**1.0**	0.57	0.5	0.07	0.6	**1.0**	0.4	0.1	0.07	0.5	**0.93**
percent	0.1	0.1	0.1	0.13	**1.0**	0.0	0.0	0.1	0.03	**1.0**	0.03	0.03	0.0	0.0	**1.0**

Table 5: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `5T1` metaheuristic GA.

	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.2	0.57	0.0	0.53	**1.0**	0.1	0.0	0.0	0.13	**1.0**	0.1	0.1	0.1	0.3	**1.0**
$w1$	0.17	0.5	0.07	0.5	**1.0**	0.0	0.03	0.17	0.17	**1.0**	0.13	0.1	0.07	0.2	**1.0**
$w2$	0.63	0.8	0.1	0.77	**1.0**	0.17	0.43	0.17	0.57	**0.9**	0.47	0.8	0.13	0.7	**0.97**
$w3$	0.03	0.3	0.1	0.83	**1.0**	0.0	0.1	0.0	0.3	**1.0**	0.17	0.3	0.13	0.53	**1.0**
percent	0.37	0.1	0.2	0.17	**0.9**	0.07	0.11	0.07	0.04	**0.93**	0.07	0.03	0.0	0.24	**1.0**

Table 6: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `5T2` metaheuristic GA.

	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.53	0.33	0.13	0.33	**1.0**	0.03	0.07	0.1	0.13	**1.0**	0.03	0.1	0.0	0.13	**1.0**
$w1$	0.57	0.53	0.3	0.43	**1.0**	0.03	0.03	0.1	0.1	**1.0**	0.23	0.07	0.33	0.17	**0.9**
$w2$	0.4	0.4	0.07	0.77	**1.0**	0.17	0.83	0.07	0.8	**0.97**	0.2	0.07	0.13	0.03	**0.97**
$w3$	0.5	0.67	0.1	0.47	**1.0**	0.1	0.77	0.0	0.73	**1.0**	0.5	0.1	0.2	0.4	**1.0**
percent	0.23	0.13	0.03	0.1	**0.97**	0.33	0.03	0.07	0.33	**1.0**	0.33	0.03	0.03	0.3	**1.0**

Table 7: Percent of evolved solutions containing each input variable for each parameter in the genotype of an individual from the `6T` metaheuristic GA.

|	bounded-strongly-corr					uncorr-similar-weights					uncorr				
	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7	x_1	x_2	x_3	x_4	x_7
$w0$	0.23	0.4	0.07	0.57	**1.0**	0.13	0.03	0.1	0.13	**1.0**	0.0	0.23	0.1	0.03	**1.0**
$w1$	0.1	0.4	0.03	0.3	**1.0**	0.0	0.03	0.1	0.13	**1.0**	0.1	0.1	0.07	0.17	**1.0**
$w2$	0.37	0.07	0.1	0.13	**1.0**	0.23	0.3	0.07	0.27	**1.0**	0.17	0.9	0.1	0.87	**1.0**
$w3$	0.13	0.17	0.07	0.43	**1.0**	0.13	0.57	0.03	0.3	**1.0**	0.1	0.93	0.2	0.83	**0.97**
$w4$	0.07	0.27	0.0	0.37	**1.0**	0.33	0.43	0.0	0.4	**0.97**	0.17	0.8	0.0	0.83	**0.97**
percent	0.03	0.0	0.1	0.0	**1.0**	0.03	0.1	0.03	0.07	**0.97**	0.2	0.03	0.13	0.47	**1.0**

8   Optimal Parameter Value Plots

In this section we display the plots of the optimal parameter values found by our metaheuristic genetic algorithm against the capacity factor of each TTP instance. In each case, the weights were normalized so that the sum of their squares equals 1. In each case, the optimal parameter value depends almost entirely on the capacity factor, and can often be approximated with a simple linear function with the notable exception of w_0 in problems of the `uncorr` knapsack type. In addition, as the number of terms increases from the `2T` to the `6T` algorithm, the optimal values of the weights becomes more noisy due to the larger dimensionality and the higher complexity.
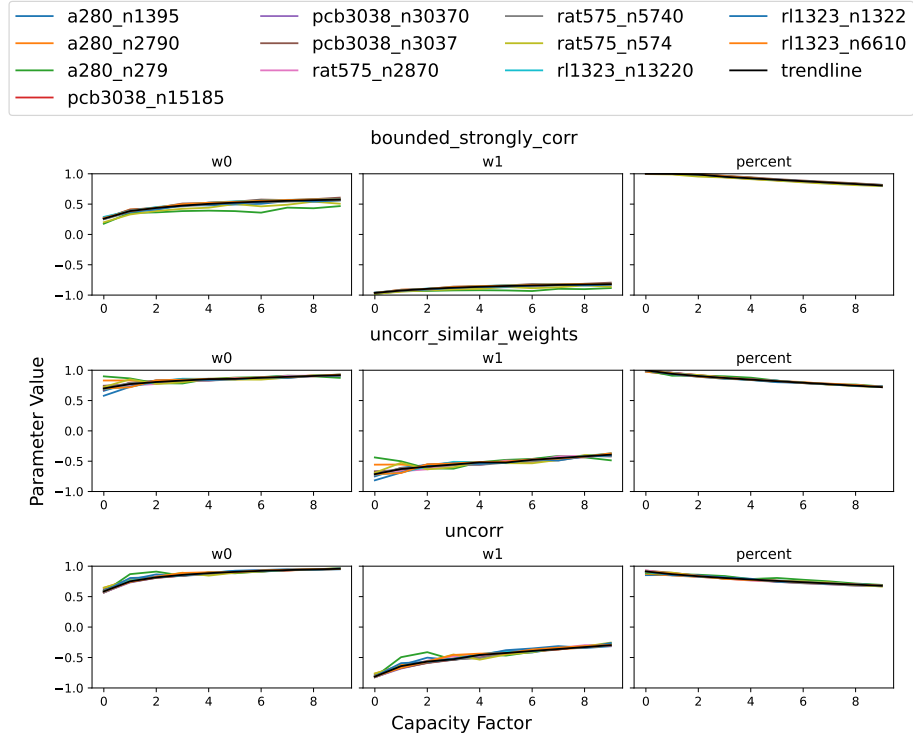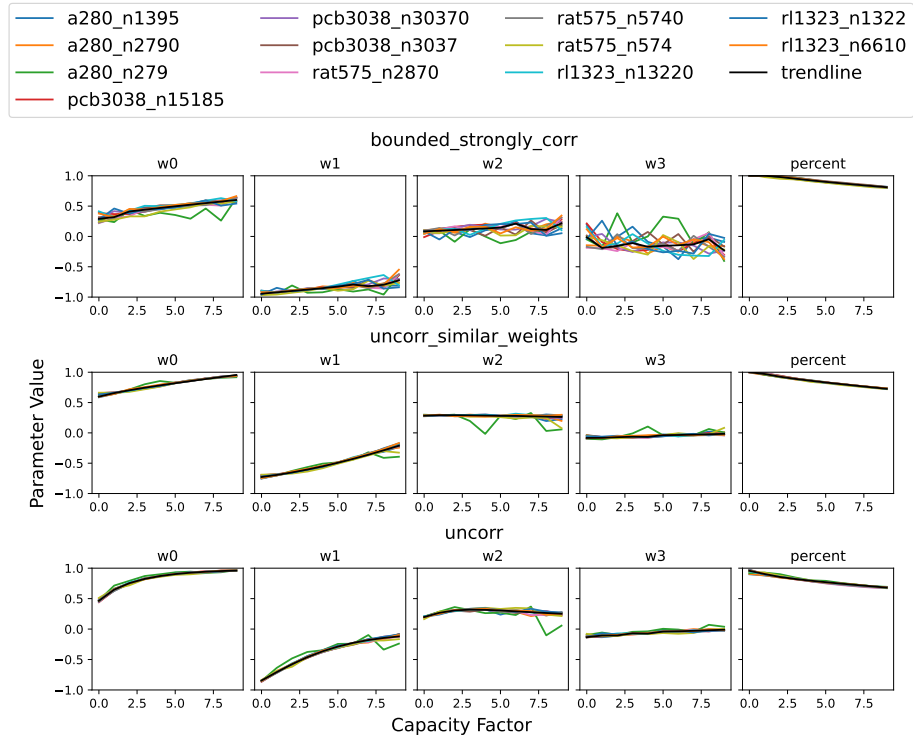
Fig. 5: Plot of the optimal parameter values against the capacity factor for our 2T metaheuristic algorithm. The solution evolved by SR is plotted in black.
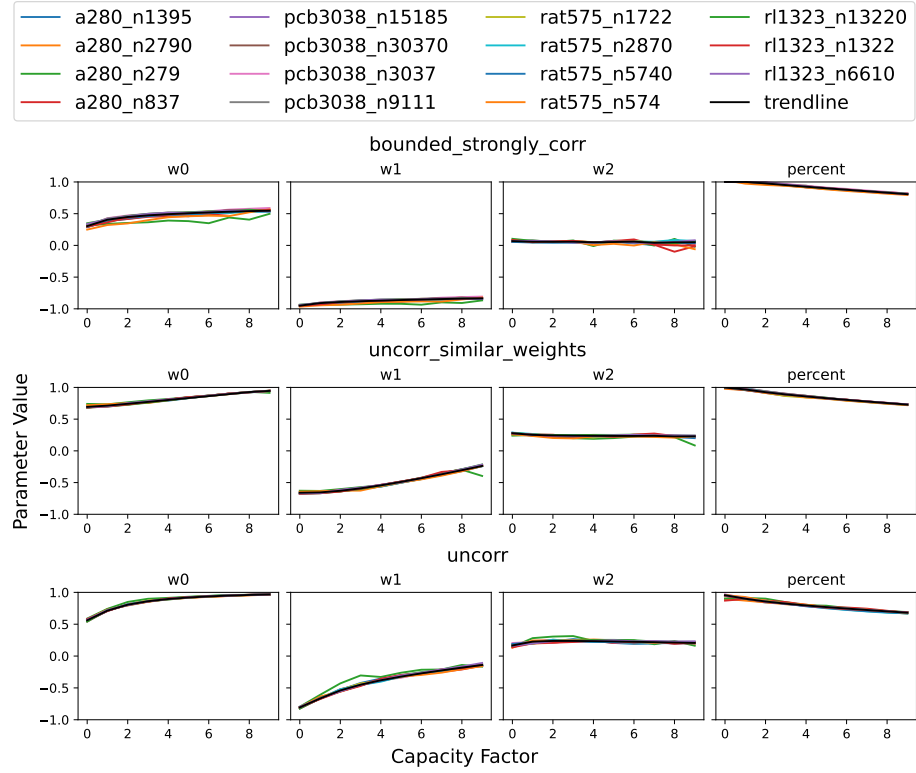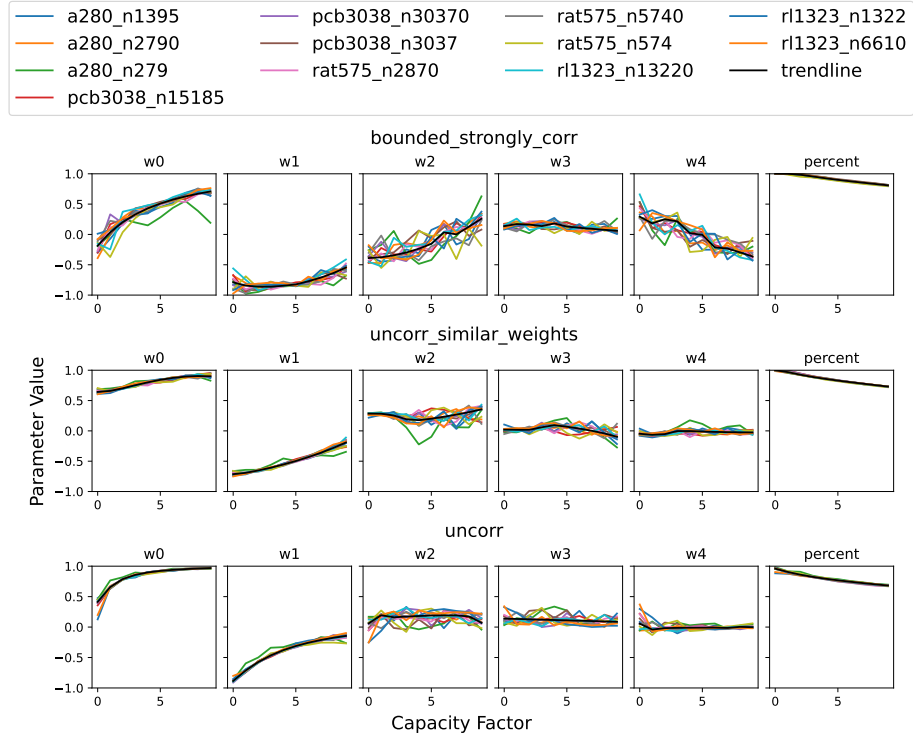
Fig. 6: Plot of the optimal parameter values against the capacity factor for our 3T metaheuristic algorithm. The same plot is shown in a different format in section 3.2 of the paper. The solution evolved by SR is plotted in black.
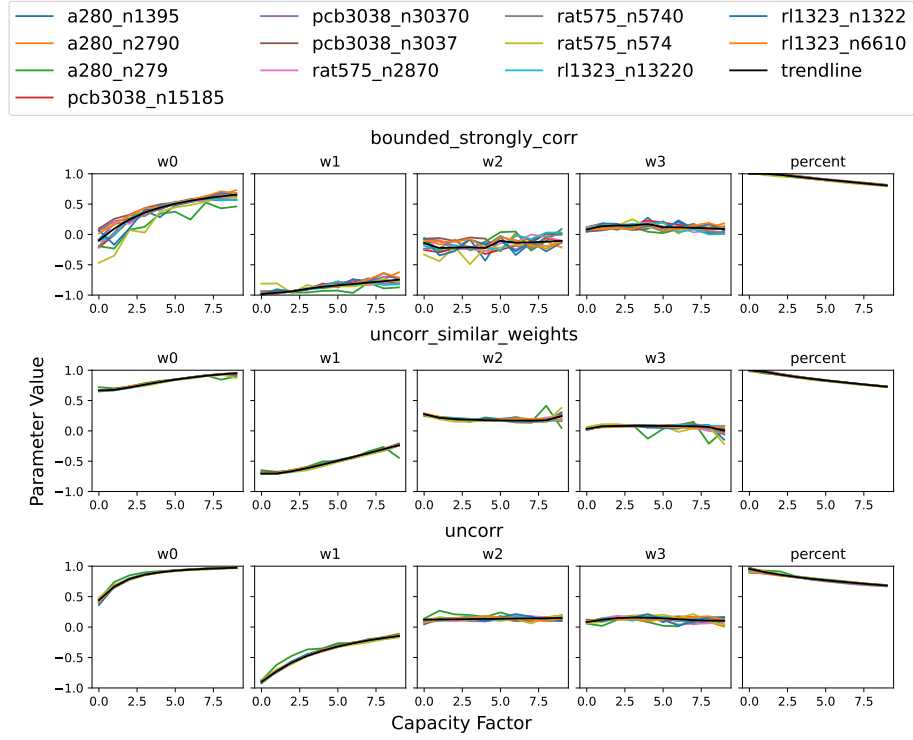
Fig. 7: Plot of the optimal parameter values against the capacity factor for our 5T1 metaheuristic algorithm. The solution evolved by SR is plotted in black.

Fig. 8: Plot of the optimal parameter values against the capacity factor for our 5T2 metaheuristic algorithm. The solution evolved by SR is plotted in black.

Fig. 9: Plot of the optimal parameter values against the capacity factor for our 6T metaheuristic algorithm. The solution evolved by SR is plotted in black.