The Dynamic Travelling Thief Problem: Benchmarks and Performance of Evolutionary Algorithms

Ragav Sachdeva, Frank Neumann, Markus Wagner

School of Computer Science, University of Adelaide, Adelaide, Australia
`ragav.sachdeva@student.adelaide.edu.au`
`frank.neumann@adelaide.edu.au`, `markus.wagner@adelaide.edu.au`

Abstract. Many real-world optimisation problems involve dynamic and stochastic components. While problems with multiple interacting components are omnipresent in inherently dynamic domains like supply-chain optimisation and logistics, most research on dynamic problems focuses on single-component problems. With this article, we define a number of scenarios based on the Travelling Thief Problem to enable research on the effect of dynamic changes to sub-components. Our investigations of 72 scenarios and seven algorithms show that – depending on the instance, the magnitude of the change, and the algorithms in the portfolio – it is preferable to either restart the optimisation from scratch or to continue with the previously valid solutions.

Keywords: dynamic optimisation · multi-component problems

1 Introduction

Real-world optimisation problems often involve dynamic and stochastic components, and evolutionary algorithms have shown to be very successful for dealing with such problems.This is due to the very nature of evolutionary computing techniques that allows them to adapt to changing environments without having to restart the algorithm when the problem characteristics change. In terms of dynamic problems, the main focus has been one studying the behaviour of evolutionary algorithms for dealing with dynamic fitness functions. Here the dynamic setting is usually specified by altering the objective function dynamically and the goal is to re-track a moving optimal peak.

Recently, problems with dynamically changing constraints have been studied in the evolutionary computation literature both from a theoretical [25,26] and experimental perspective [1,2,23,24]. Here the objective function stays fixed and constraints change over time. This reflects scenarios where the goal is to maximise profit but the resources to achieve this goal can vary over time. A classical example are supply chain management problems where trucks and trains can become (un)available due to maintenance and break downs. Studies include important problems such as the knapsack problem [23] and more generally the optimisation of submodular functions [24] under dynamically changing constraints.

In addition, for the knapsack problem, the optimisation under dynamic and stochastic constraints has also been investigated [4].

In this paper, we investigate dynamic variants of the travelling thief problem (TTP). The TTP has been introduced in the evolutionary computation literature in [6] as a multi-component problem combining the classical travelling salesperson problem (TSP) and the knapsack problem (KP). The TTP comprises a thief stealing items with weights and profits from a number of cities. The thief has to visit all cities once and collect items such that the overall profit is maximised. The thief uses a knapsack of limited capacity and pays rent for it proportional to the overall travel duration. To make the two components (TSP and KP) interdependent, the speed of the thief is made non-linearly dependent on the weight of the items picked so far. The interactions of the TSP and the KP in the TTP result in a complex problem that is hard to solve by tackling the components separately. The TTP has gained significant attention in the evolutionary computation literature and several competitions organised to solve this problem have led to significant progress improving the performance of solvers.

In order to further extend the impact and significance of studies around the TTP, and to support research on the effects of dynamic changes to sub-components of problems, we design two types of dynamic benchmarks. The first one relates to the KP where items in cities can become available and unavailable. This inevitably changes the options of items that can be collected. The second one relates to the TSP component of the problem where cities can be made available and unavailable modelling the change in availability of locations.

We design different scenarios for these dynamic settings and examine how evolutionary algorithms based on popular algorithmic components for the TTP can deal with such changes. In particular, we are interested in learning when one should continue with the optimisation by recovering from the previous solution, or when one should create a new solution from scratch as a starting point for the next iterative improvement.

2   The Travelling Thief Problem and its Variants

The TTP is a combinatorial optimisation problem that aims to provide testbeds for solving problems with multiple interdependent components [6,22]. In the following, we first introduce the TTP. Then, we put this present research into the context of existing works by reviewing existing TTP variants.

2.1   Definition

The TTP combines two well-known problems, namely, the Travelling Salesman Problem and the Knapsack Problem.

The problem is defined as follows: we are given a set of n cities, the associated matrix of distances d_{ij}, and a set of m items distributed among these cities. Each item k is defined by a profit p_k and a weight w_k. A thief must visit all the cities exactly once, stealing some items on the road, and return to the starting city.

The knapsack has a capacity limit of W, i.e. the total weight of the collected items must not exceed W. In addition, we consider a renting rate R that the thief must pay at the end of the travel, and the maximum and minimum velocities denoted v_{max} and v_{min} respectively. Furthermore, each item is available in only one city, and $A_i \in \{1, \ldots, n\}$ denotes the availability vector. A_i contains the reference to the city that contains the item i.

A TTP solution is typically coded in two parts: the tour $X = (x_1, \ldots, x_n)$, a vector containing the ordered list of cities, and the picking plan $Z = (z_1, \ldots, z_m)$, a binary vector representing the states of items (1 for packed, 0 for unpacked).

To establish a dependency between the sub-problems, the TTP was designed such that the speed of the thief changes according to the knapsack weight. To achieve this, the thief's velocity at city c is defined as $v_x = v_{max} - C \times w_x$, where $C = \frac{v_{max} - v_{min}}{W}$ is a constant value, and w_x is the weight of the knapsack at city x.

The total value of items is $g(Z) = \sum_m p_m \times z_m$, such that $\sum_m w_m \times z_m \leq W$. The total travel time is $f(X, Z) = \sum_{i=1}^{n-1} t_{x_i, x_{i+1}} + t_{x_n, x_1}$, where $t_{x_i, x_{i+1}} = \frac{d_{x_i, x_{i+1}}}{v_{x_i}}$ is the travel time from x_i to x_{i+1}.

The TTP's objective is to maximise the total travel gain function, which is the total profit of all items minus the travel time multiplied with the renting rate: $F(X, Z) = g(Z) - f(X, Z) \times R$.

For a worked example, we refer the interested reader to the initial TTP article by Bonyadi et al. [7].

2.2   TTP variants to date

Since the inception of the TTP by Bonyadi et al. [7], research has focused not only on the original formulations, but also on a few variants have been created to either address its shortcomings or to investigate the interaction of the components. In the following we provide a brief overview.

Thus far, most research has considered the single-objective TTP formulation (TTP1) from [7], which is typically the TTP variant that is referred to as *the TTP* (see Section 2.1). The other inaugural variant (TTP2) considers two objectives and additionally, a value drop of items over time.

For the single-objective TTP, a wide range of approaches has been considered, ranging from general-purpose iterative search-heuristics [22], co-evolutionary approaches [8,20] and memetic approaches [19] to swarm-intelligence based approaches [27,34], to approaches with problem specific search operators [14], customised estimation of distribution approaches [18] and to hyper-heuristics [13]. Wagner et al. [28] provide a comparison of 21 algorithms for the purpose of portfolio creation. To better understand the effect of operators on a more fundamental level, [33] and [31] present, through fitness-landscape analyses, correlations and characteristics that are potentially exploitable.

For a variant with fixed tours, named Packing-While-Travelling (PWT), Neumann et al. [21] showed that it can be solved in pseudo-polynomial time via dynamic programming, by exploiting the fact that the weights are integer. [30] ex-

tended this to optimal approaches for the entire TTP, however, their approaches are only practical for very small instances.

As the TTP's components are interlinked, multi-objective considerations that investigate the interactions via the idea of "trade-off"-relationships are becoming increasingly popular as of lately. For example, [32] created an approach where the objectives were the travel time and total profit of items. [29] considered a bi-objective version of the TTP, where the objectives were the total weight and the TTP objective score. At two recent competitions[1], an intermediate version of TTP1 and TTP2 was proposed, which can be seen as a TTP1 with two objectives or TTP2 without a value drop of items over time. The same TTP variant was investigated by [5]. to build the bridge from TTP1 to TTP2 by having two objectives but not adding another level of complexity to the problem. Recently, Chagas et al. [10] proposed a NSGA-II and customised random-key genetic algorithm for this problem.

Lastly, it is often argued if the TTP is realistic enough because of its formulation – not because it is static, which we are addressing in this article – but because it requires a single thief to travel across hundreds or thousands of cities to steal items. Chand et al. [11]'s Multiple Travelling Thieves Problem relaxes several constraints so that a group of thieves can maximise the group's collective profit by going on different tours and without the need to visit all cities.

A more general discussion of a multi-objective approach to interconnected problems can be found in [17], and a more general discussion on the opportunities of multi-component problems can be found in [9].
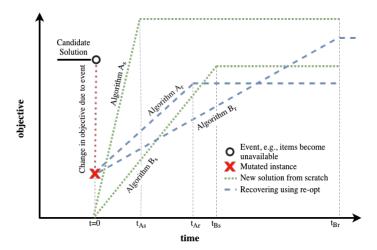
Note that our research in this article has been conducted in parallel to and independently of that presented by Herring et al. [16] on multi-objective dynamic TTP variants. There, the authors considered dynamic city locations, dynamic item availability and dynamic item values. Experimentally, they considered one dynamic setting for several small TTP instances, and they did not investigate the effects of disruptions in isolation.

3   Dynamic Travelling Thief Problem (DynTTP)

Broadly speaking, DynTTP is an extension of the classic TTP where the problem constraints can change during run-time. In this study, we investigate the introduction of dynamism in two different ways by flipping the availability status of $d\%$ of the (1) items and (2) cities uniformly at random, i.e. if an item/city is available, it is made unavailable and vice-versa. From here on, we will call the event when availabilities change a *disruption*, and the time span between two disruptions is called an *epoch* which is of duration z.

To allow for a meaningful investigation, both these dynamic changes are studied independently of one another. Furthermore, while "Toggling Items" the tour is kept fixed and similarly while "Toggling Cities" the packing list is not

---
[1] *EMO-2019*,   https://www.egr.msu.edu/coinlab/blankjul/emo19-thief/   and *GECCO-2019* https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

Fig. 1. Simplified DynTTP scenario: as time $t = 0$, an instance gets disrupted, and a decision maker has to decide (1) whether to employ algorithm A or algorithm B and (2) whether to recover from the previous solution R (marked ✗) or to optimise from scratch S. $t_{AS} \ldots t_{BR}$ denote the times needed by the different choices to converge.

re-optimised. In addition, to guarantee that the solutions remain valid and to permit a fair analysis, we make the following design decisions:

Toggling Items: If an item is made unavailable, it is no longer in the current packing plan (if already picked) and cannot be picked again by the thief. On the other hand, if an item is made available again, it is available for the thief to steal but it is not automatically added to the current packing list (to prevent unintentionally exceeding the knapsack capacity).

Toggling Cities: If a city is made unavailable, it is removed from the current tour while maintaining the order of the remaining cities. Of course, this also implies that the items in this city are no longer available either. On the other hand, if a city is made available again, it is inserted back into the tour at the same position as it was before. Furthermore, toggling a city back in, also updates the knapsack to include the items from this city that were previously in the packing list. This is an acceptable thing to do because even if all the cities are toggled 'on', the knapsack will not exceed capacity as the packing plan would revert back to the initial solution at $t = 0$.

We show a simplified DynTTP scenario in Figure 1. As it is often the case with dynamic optimisation, it depends on the magnitude of the change, on the available computational budget, and on the available algorithms whether it is preferable to restart the optimisation from scratch or to recover based on the previous solution.

Note that a dedicated random number generator is responsible for modifying the instances. This way, different algorithms observe the same set of conditions

for a given instances, and thus we ensure that sequences of disruptions remain comparable, as otherwise instances would drift apart and the algorithms' performances become incomparable.

4   Heuristics for the DynTTP

As the items/cities are toggled ensuing from the dynamic change, the objective value of the known TTP solution changes. At this point there are two choices: Rebuild a TTP solution from scratch for, what is effectively a new TTP instance; or re-optimise the packing/tour for the previously known best solution.

To achieve this, we make use of a number of generic as well as problem-specific algorithmic building blocks.

BITFLIP [15] is a simple greedy hill-climber with a low runtime. It iteratively evaluates the outcome of flipping each bit position corresponding to item i in the packing plan Z. If a flip improves the objective value, the change is kept, otherwise Z remains unchanged. A single iteration of the operator ends when all bit flips have been attempted once. Multiple passes can result in further improvement of the objective value.

PACKITERATIVE [15] is a greedy packing heuristic with a scoring function that computes a trade-off between a distance that item i has to be carried over, its weight and its profit. As it might be beneficial to change the influence of the distance depending on the particular instance, PACKITERATIVE performs an interval search over one parameter that balances the respective influences.

INSERTION [15] is another simple greedy hill-climber, and very similar to BITFLIP. It operates on tours and takes advantage of the situation where a valuable item at a particular city is picked up early and it is worth trying to delay the item pickup by modifying the tour.

CLK (Chained Lin-Kernighan heuristic [3][2]) is a fast TSP solver. While research optimal TTP tours are not necessarily TSP-optimal, it is nevertheless a popular starting point in TTP research.

REA_m is a variant of the simple population-based $(\gamma+1)$EA outlined by Doerr et al. [12] that utilises a diversity mechanism to re-optimise the solution to a perturbed instance. It works with a diverse set of solutions at Hamming distance of at most γ from a previously known solution x^{old}. In every iteration, the algorithm first selects a parent individual x, from the diverse set of solutions along with the current best solution, via a biased random selection. Following this, one offspring y is generated using a standard bit mutation with the rate of $1/n$. If the Hamming distance $i = H(y, x^{old})$ is at most γ, this offspring y replaces the previous best individual x^i at distance i if it is at least as good. It is imperative to point out that the "change event" studied by Doerr et al. is conceptually different to what we are investigating: in their work the objective function f^{old} is perturbed resulting in a new objective function f and their problem definition imposes no condition where their known solution becomes

---

[2] As available at http://www.tsp.gatech.edu/concorde/downloads/downloads.htm

invalid after the change. To adapt to these fundamental differences, we employ this algorithm where the initial known solution i.e. x^{old} is the resulting solution *after* the disruption and γ is set to m (number of items originally available). The merit of using this diversity mechanism is to strike a balance between exploring the neighbourhoods of a previously known good solution vs. hill-climbing the best solution thus far.

For the different scenarios of the dynamically changing availability of items and cities, we combine the building blocks in the following seven ways.

Toggling Items – four approaches:
1. Re-optimise the mutated packing using BITFLIP;
2. Re-optimise the mutated packing using REA_m with the population size of m, which is the number of items originally available.
3. Create a new packing plan from scratch using PACKITERATIVE;
4. Create a new packing plan from scratch using PACKITERATIVE and then optimise it using BITFLIP;

Toggling Cities – three approaches:
1. Re-optimise the mutated tour using INSERTION;
2. Create a new tour from scratch using CLK;
3. Create a new tour from scratch using CLK and then optimise it using INSERTION.

5   Computational Investigation

In the following, we first describe our experimental setup and the 72 scenarios. Then, we present the results and highlight situations that we find interesting.

We have made all the code, the results, the and processing scripts publicly available at *https://github.com/ragavsachdeva/DynTTP*. This includes cross-validated C++ implementations of the algorithms.

5.1   Experimental setup

Polyakovskiy et al. presented a systematically created set of benchmark TTP instances that cover a wide range of features and are based on well known instances of TSP and KP [22]. The following set of TTP instances covers a range of scenarios for the travelling thief, and therefore was selected for the purposes of this study:[3]

280_n279_bounded-strongly-corr_01	a280_n1395_uncorr-similar-weights_05
a280_n2790_uncorr_10	fnl4461_n4460_bounded-strongly-corr_01
fnl4461_n22300_uncorr-similar-weights_05	fnl4461_n44600_uncorr_10
pla33810_n33809_bounded-strongly-corr_01	pla33810_n169045_uncorr-similar-weights_05
pla33810_n338090_uncorr_10	

---

[3] It has also been the subset used at the 2014/2015 and 2017 TTP competitions, e.g., https://cs.adelaide.edu.au/~optlog/TTP2017Comp/

The numbers in each instance name are (from left to right) the total number of cities, the total number of items, and the knapsack size in that instance. For example, instance a280_n2790_uncorr_10 has 280 cities all of which (except the first) have 10 items each, totalling 2790 items, and it has the largest knapsack. In addition the terms uncorrelated, uncorrelated-similar-weights and bounded-strongly-correlated reflect the unique knapsack types as described in [22]. To facilitate readability, we will use the abbreviations A279, A1395, ..., P338090, which include the number of items in their names.

In our computational study of the DynTTP, we investigate disruptions of the following types:

- instance feature $f \in \{$items, cities$\}$;
- amount of disruption $d \in \{1\%, 3\%, 10\%, 30\%\}$;
- period z, which is the time between disruptive events.

As the disruptions to instances are random (modulo the determinism described in Section 3), we perform 30 independent runs of each scenario – a "scenario" is an instance, together with an instance feature f that is to be disrupted, an amount of disruption d, and a period z.

We conduct all experiments on the university's compute cluster, which is equipped with Intel Xeon Gold 6148 processors (2.4 GHz). The initial solutions are created by running CLK first to generate a good TSP tour and then by running PackIterative+Bitflip to converge to a good TTP solution.

5.2   A single disruptive event

In our first study, we investigate how well the different algorithms perform given different scenarios. This study comprises 252 experiments, as we run 7 algorithms on the 9 instances that undergo 4 different amounts of disruption, and as mentioned above, we repeat each experiment 30 times. Following the disruption, each algorithm has a maximum of $z = 10$ minutes, however, it can stop early if it detects convergence based on its internal criterion.

In the following, we limit ourselves to subsets that show interesting situations. In general, we can typically make the following qualitative observations:

1. *Toggling Items:* When recovering from the previous solution, Bitflip makes only a small number of improvements. In Figure 2 Part (a), we can see that there is only a small number of improvements for each of the 30 runs, even though 4460 bits can be flipped in the packing plan. When starting from scratch using PackIterative+Bitflip, we regularly observe a subset of runs that had difficulties reaching the quality of the solution prior to being discarded by PackIterative (see the red crosses in the staircase plots); hence, restarting from scratch can be seen as a risky choice here.
2. *Toggling Cities:* When recovering from the previous solution, Insertion makes only a small number of improvements here as well; moreover, the improvements are very minor. We can see in Figure 2 Part (b), that CLK+Insertion has significantly more difficulties surpassing the previous solutions (red crosses) than the conceptually similar PackIterative+Bitflip has in the *Toggling Items* scenarios.
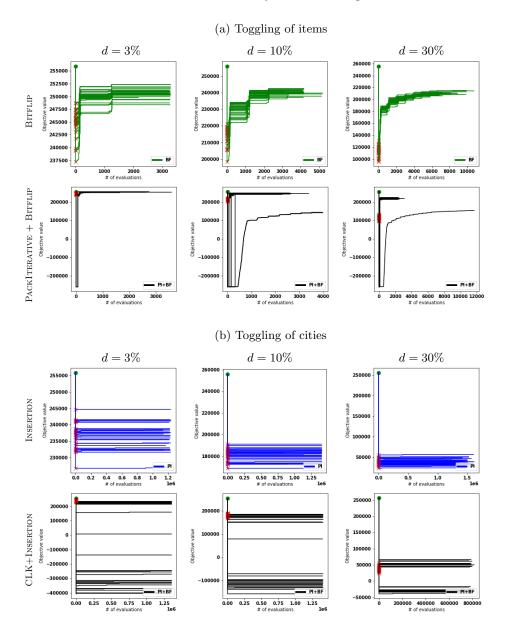
(a) Toggling of items

(b) Toggling of cities

Fig. 2. DynTTP results of four algorithms on six scenarios for F4460. **Part (a)** Toggling of items: recovering from the previous solution (BITFLIP, 1st row), starting from scratch (PACKITERATIVE+BITFLIP, 2nd row). **Part (b)** Toggling of cities: recovering from the previous solution (INSERTION, 1st row), starting from scratch (CLK+INSERTION, 2nd row). Note that the scales have not be normalised on purpose to show the characteristics of the individual algorithms. Red crosses show the solution quality of the last solution given the new situation (see Figure 1), hence they represent the starting points for BITFLIP and INSERTION.
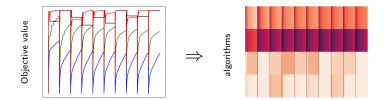
Fig. 3. *Average* performance of four algorithms for 10 epochs (F4460, $d = 30\%$). Time, as measured in the number of evaluations, passes from left to right; each period is of length $z = 4460$ evaluations. Within each epoch, the performance is normalised across all algorithms. The lighter the colour, the higher the performance. Algorithms: BITFLIP (green, 1st row), REA$_m$ (blue, 2nd row), PACKITERATIVE (black, 3rd row), PACKITERATIVE+BITFLIP (red, 4th row).

3. REA$_m$, being a population-based approach for *Toggling Items* scenario, often makes many improvements over the starting points. While it typically hits the time limit of 10 minutes (and amounting, e.g., 70,000 fitness evaluations on F4460 and up to 10 million fitness evaluations on A279 during this time), the trajectories imply that it was typically far from converging.
4. CLK without a subsequent hillclimber typically performs badly for *Toggling Cities*, as it is able to generate a TSP tour only once by a TSP solver, which ignores the fixed KP component. In contrast to this, PACKITERATIVE without a subsequent hillclimber performs significantly better as it performs an internal local search that attempts to maximise the TTP score.

5.3   Multiple disruptive events

Next, we investigate the performance over time for multiple disruptions. In particular, we observe 10 epochs and each lasts for a period of $z = m$ function evaluations (equal to the number of items in the original instance) or for 10 minutes, whichever comes first. Again, we investigate 72 scenarios in total. Due to the high dimensionality of this study, (1) we use heatmaps to allow for a qualitative inspection of the results, and (2) we cut through the data in different ways and use statistical tests to compare the heuristic approaches.

Heatmaps. Before moving on, we briefly explain the creation of each heatmap. First, we record for each of the 30 runs all TTP objective scores. Second, for each scenario and each epoch (as these are identical) we determine the minimum and maximum objective scores across all algorithms, and then linearly normalise the average of the 30 trajectories into $[0, 1]$. These scores are then visualised as coloured bars, from black (lowest value seen by any algorithm in this scenario and epoch), via red to a light peach (highest value seen by any algorithm). Figure 3 shows an example for four algorithms. To facilitate readability, we omit all labels, because the period is fixed (as defined above), and because the order of the four (respectively three) approaches is stated in the captions.[4]

---

[4] The algorithm's order is also identical to the order listed in Section 4.

(a) Toggling of items
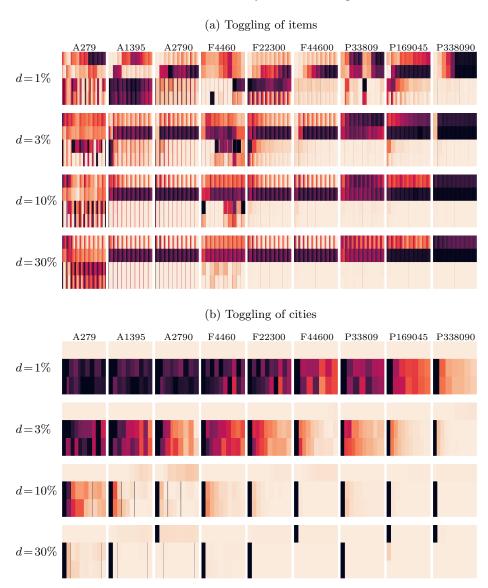

(b) Toggling of cities

Fig. 4. DynTTP results of seven algorithms on 72 scenarios, and for a duration of 10 epochs each. The performance has been normalised across all algorithms (within each individual epoch and w.r.t. to all objective scores seen in this epoch) to allow for a relative performance comparison. **Part (a)** Toggling of items: BITFLIP (1st row), REA$_m$ (2nd row), PACKITERATIVE (3rd row), PACKITERATIVE+BITFLIP (4th row). **Part (b)** Toggling of cities: INSERTION (1st row), CLK (2nd row), CLK+INSERTION (3rd row). The lighter the colour, the higher the performance. **Example:** based on the heatmap in the top right corner of Part (a) (i.e., P338090 and $d = 1\%$), we can see that all algorithms perform comparatively at first, but then PACKITERATIVE and PACKITERATIVE+BITFLIP gradually perform better and better than the other two approaches.

Figure 4 shows all the results. While we will dig deeper in our following statistical analyses, it is obvious that some situations are "very clear". For example, for the *Toggling Items* scenarios and for P338090 (rightmost column of heatmaps), the bottom two approaches always perform best (these are PACK-ITERATIVE and PACKITERATIVE+BITFLIP, which both start from scratch) as they have the brightest colours. In contrast to this, the approaches that start from scratch for the *Toggling Cities* scenarios (with $d = 1\%$) perform the worst. We can also see that, as we move from left to right in this matrix of heatmaps, that patterns change and thus the ranking of the algorithms change – this is important for anyone making decisions: based on this qualitative analysis, it is already clear that different situations require conceptually different approaches.

Statistical Analyses. Given our high-dimensional dataset, we can cut through it in various ways and compare the performance of the different algorithms under various conditions. As inputs to these tests, we use the averaged (normalised) performance in two ways: (1) the final performance at the end of an epoch (END), and (2) the area-under-the-curve of the objective scores in an epoch (AUC), to consider the convergence speed as well as quality.

We employ the pairwise, one-sided Mann-Whitney U test as a ranksum test that does not rely on normal-distributed data. Again, we will limit ourselves to interesting cases, and to those with p-values of less than 5%.

In the following, we first provide a global picture, and then slice through the data along the four amounts of disruption d and along the nine instances. This results in a total of 650 pair-wise statistical tests. Our summary:

1. *Global level (aggregated over all epochs, instances and values of d).* (1) Toggling Items: Starting from scratch is preferred, as PACKITERATIVE+BITFLIP outperforms PACKITERATIVE, which outperforms BITFLIP and that in turn outperforms REA_m. (2) Toggling Cities: Recovering is preferred here, as INSERTION outperforms both CLK and CLK+INSERTION, which perform comparably. One of the reasons for this is that we do not optimise the packing in this case, and when we start from scratch the TSP is changed and the packing no longer is adequate with it. Moreover, the heuristics here are problem-independent, whereas PACKITERATIVE is TTP-specific.

2. *Disruptions d (aggregated over all epochs and instances)*: (1) Toggling Items: We now see a few changes in the ranking when d=1%, i.e., BITFLIP now outperforms PACKITERATIVE as well (in addition to REA_m, but only in case of AUC) possibly because the change to the instance is very minor and recovering is better than risking the start from a solution from scratch that is of unknown quality. However, when $d \geq 3\%$ the disruption seems too big for BITFLIP to recover, and PACKITERATIVE again outperforms BITFLIP.
(2) Toggling Cities: We observe only a single change from the global picture, i.e, CLK+INSERTION outperforms the other two when $d = 30\%$ (END).

3. *Instances (aggregated over all epochs and values of d).* (1) Toggling Items: A279 is a surprise in that REA_m outperforms all three in AUC; for the next four larger instances (up to F22300) this does not hold anymore and many

comparisons are insignificant; for the largest five instances, the global picture holds and PACKITERATIVE+BITFLIP is the always significantly better. (2) Toggling Cities: There is never any deviation from the global picture, except for P338090, where there are not significant differences of all three approaches when the final performance (END) is considered.

To sum up, we observe that disruptive events have a major impact on the performance of the algorithms, which in turn significantly affect the relative rankings.

6 Outlook

Future research can go into many different directions. For example, none of the approaches was properly problem-specific, and only REA_m employed a population. Even though REA_m did not perform well, we intend to explore how to adapt its diversity mechanism to entire TTP solutions. In addition, we envision multi-objective approaches and co-evolutionary approaches that will balance solution quality while preparing for disruptive events of unknown type and magnitude.

References

1. Ameca-Alducin, M.Y., Hasani-Shoreh, M., Blaikie, W., Neumann, F., Mezura-Montes, E.: A comparison of constraint handling techniques for dynamic constrained optimization problems. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2018)
2. Ameca-Alducin, M.Y., Hasani-Shoreh, M., Neumann, F.: On the use of repair methods in differential evolution for dynamic constrained optimization. In: 21st International Conference EvoApplications. vol. 10784, pp. 832–847. Springer (2018)
3. Applegate, D., Cook, W.J., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. INFORMS Journal on Computing **15**(1), 82–92 (2003)
4. Assimi, H., Harper, O., Xie, Y., Neumann, A., Neumann, F.: Evolutionary bi-objective optimization for the dynamic chance-constrained knapsack problem based on tail bound objectives (2020), `https://arxiv.org/abs/2002.06766`
5. Blank, J., Deb, K., Mostaghim, S.: Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms, pp. 46–60. Springer (2017)
6. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The Travelling Thief Problem: The First Step in the Transition from Theoretical Problems to Realistic Problems. In: IEEE Congress on Evolutionary Computation. pp. 1037 – 1044. IEEE (2013)
7. Bonyadi, M.R., Michalewicz, Z., Barone, L.: The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: IEEE Congress on Evolutionary Computation. pp. 1037–1044. IEEE (2013)
8. Bonyadi, M.R., Michalewicz, Z., Przybylek, M.R., Wierzbicki, A.: Socially inspired algorithms for the TTP. In: Genetic and Evolutionary Computation Conference. pp. 421–428. ACM (2014)

9. Bonyadi, M.R., Michalewicz, Z., Wagner, M., Neumann, F.: Evolutionary Computation for Multicomponent Problems: Opportunities and Future Directions, pp. 13–30. Springer (2019)

10. Chagas, J.B.C., Blank, J., Wagner, M., Souza, M.J.F., Deb, K.: A non-dominated sorting based customized random-key genetic algorithm for the bi-objective traveling thief problem. CoRR **abs/2002.04303** (2020), `https://arxiv.org/abs/2002.04303`

11. Chand, S., Wagner, M.: Fast heuristics for the multiple traveling thieves problem. In: Genetic and Evolutionary Computation Conference. pp. 293–300. GECCO '16, ACM (2016)

12. Doerr, B., Doerr, C., Neumann, F.: Fast re-optimization via structural diversity. CoRR **abs/1902.00304** (2019), `http://arxiv.org/abs/1902.00304`

13. El Yafrani, M., Martins, M., Wagner, M., Ahiod, B., Delgado, M., Lüders, R.: A hyperheuristic approach based on low-level heuristics for the travelling thief problem. Genetic Programming and Evolvable Machines **19**(1), 121–150 (2018)

14. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Genetic and Evolutionary Computation Conference. pp. 385–392. ACM (2015)

15. Faulkner, H., Polyakovskiy, S., Schultz, T., Wagner, M.: Approximate approaches to the traveling thief problem. In: Genetic and Evolutionary Computation Conference. pp. 385–392. ACM (2015)

16. Herring, D., Kirley, M., Yao, X.: Dynamic multi-objective optimization of the travelling thief problem. CoRR **abs/2002.02636** (2020)

17. Klamroth, K., Mostaghim, S., Naujoks, B., Poles, S., Purshouse, R., Rudolph, G., Ruzika, S., Sayn, S., Wiecek, M.M., Yao, X.: Multiobjective optimization for interwoven systems. J. of Multi-Criteria Decision Analysis **24**(1-2), 71–81 (2017)

18. Martins, M.S.R., El Yafrani, M., Delgado, M.R.B.S., Wagner, M., Ahiod, B., Lüders, R.: Hseda: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem. In: Genetic and Evolutionary Computation Conference. p. 361368. ACM (2017)

19. Mei, Y., Li, X., Yao, X.: On investigation of interdependence between sub-problems of the TTP. Soft Computing **20**(1), 157–172 (2014)

20. Namazi, M., Sanderson, C., Newton, M.A.H., Sattar, A.: A cooperative coordination solver for travelling thief problems. ArXiv e-prints (2019)

21. Neumann, F., Polyakovskiy, S., Skutella, M., Stougie, L., Wu, J.: A fully polynomial time approximation scheme for packing while traveling. In: Algorithmic Aspects of Cloud Computing. pp. 59–72. Springer (2019)

22. Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F.: A comprehensive benchmark set and heuristics for the traveling thief problem. In: Genetic and Evolutionary Computation Conference. pp. 477–484. ACM (2014)

23. Roostapour, V., Neumann, A., Neumann, F.: On the performance of baseline evolutionary algorithms on the dynamic knapsack problem. In: Parallel Problem Solving from Nature. pp. 158–169 (2018)

24. Roostapour, V., Neumann, A., Neumann, F., Friedrich, T.: Pareto optimization for subset selection with dynamic cost constraints. In: Proc. of AAAI. pp. 2354–2361 (2019)

25. Roostapour, V., Pourhassan, M., Neumann, F.: Analysis of evolutionary algorithms in dynamic and stochastic environments. In: Theory of Evolutionary Computation, pp. 323–357. Springer (2020)

26. Shi, F., Schirneck, M., Friedrich, T., Kötzing, T., Neumann, F.: Reoptimization time analysis of evolutionary algorithms on linear functions under dynamic uniform constraints. Algorithmica **81**(2), 828–857 (2019)
27. Wagner, M.: Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In: Swarm Intelligence. pp. 273–281. Springer (2016)
28. 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 (2018)
29. Wu, J., Polyakovskiy, S., Wagner, M., Neumann, F.: Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In: Genetic and Evolutionary Computation Conference. pp. 777–784. ACM (2018)
30. Wu, J., Wagner, M., Polyakovskiy, S., Neumann, F.: Exact approaches for the travelling thief problem. In: Simulated Evolution and Learning. pp. 110–121. Springer (2017)
31. Wuijts, R.H., Thierens, D.: Investigation of the traveling thief problem. In: Genetic and Evolutionary Computation Conference. p. 329337. ACM (2019)
32. Yafrani, M.E., Chand, S., Neumann, A., Ahiod, B., Wagner, M.: Multi-objectiveness in the single-objective traveling thief problem. In: Genetic and Evolutionary Computation Conference Companion. pp. 107–108. ACM (2017)
33. Yafrani, M.E., Martins, M.S.R., Krari, M.E., Wagner, M., Delgado, M.R.B.S., Ahiod, B., Lüders, R.: A fitness landscape analysis of the travelling thief problem. In: Genetic and Evolutionary Computation Conference. p. 277284. ACM (2018)
34. Zouari, W., Alaya, I., Tagina, M.: A new hybrid ant colony algorithms for the traveling thief problem. In: Genetic and Evolutionary Computation Conference Companion. p. 9596. ACM (2019)