Dynamic programming operators for the bi-objective Traveling Thief Problem

Roberto Santana
University of the Basque Country (UPV/EHU)
San Sebastian, Spain
Visitor Scholar EBTIC (Emirates ICT Innovation Center), Khalifa University
Abu Dhabi, United Arab Emirates
roberto.santana@ehu.eus

Siddhartha Shakya
EBTIC (Emirates ICT Innovation Center),
Khalifa University
Abu Dhabi, United Arab Emirates
sid.shakya@ku.ac.ae

***Abstract*—The traveling thief problem (TTP) has emerged as a realistic multi-component problem that poses a number of challenges to traditional optimizers. In this paper we propose different ways to incorporate dynamic programming (DP) as a local optimization operator of population-based approaches to the bi-objective TTP. The DP operators use different characterizations of the TTP instance to search for packing plans that improve the best current solutions. We evaluate the efficiency of the DP-based operators using TTP instances of up to 33810 cities and 338100 items, and compare the results of the DP operators with state-of-the-art algorithms for these instances. Our results show that DP-based approaches, applied individually and in combination with other types of operators, can produce good approximations of the Pareto sets for these problems.**

***Index Terms*—traveling thief problem, evolutionary optimization, MOEA, TSP, dynamic programming**

I. Introduction

The traveling thief problem (TTP) [2] is a paradigmatic example of multi-component optimization problems, which try to fill the gap between traditional benchmark optimization problems and more realistic ones. In the TTP, a thief has to travel a set of n cities and go back to the origin. In each city he may get one or more items for his knapsack, which has a maximum capacity. Items add value to the bounty collected by the thief but also add a weight which in turn reduces his speed in moving from city to city.

The particular type of interaction between the traveling salesman problem (TSP) and the knapsack problem (KS), which are the two components of TTP, determines the quality of a TTP solution. Therefore, it is not possible to compute the optimal solution by solving each problem independently. Given its suitability to model more realistic and intricate optimization scenarios, TTP has attracted considerable research recently [5], [11], [15], [20].

TTP has also been extended to the multi-objective scenario. For example, in [19], two different objectives are proposed, maximizing the total reward and minimizing the total weight of the collected items. Another bi-objective formulation is presented in [1], where the first objective considered is to maximize the value of the items in the knapsack, subject to the maximum capacity constraint, and the second one is minimizing the time spent travelling. Multi-objective variants of TTP add a number of interesting and challenging research questions, such as how to deal with interactions between the problem objectives and the problem variables, or how to keep a diverse sets of solutions when the dimensionality of the problem grows.

In this paper we address the bi-objective TTP problem introduced in [1]. We propose a fast TTP evaluation strategy combined with the application of different heuristics that incorporate dynamic programming. While DP approaches can provide exact solutions for small instances of the knapsack problem, they become unfeasible for larger instances. However, in this paper we show that, by making changes to the way DP is used, it is possible to find sub-optimal or partial solutions to TTP.

The goal of the paper is to develop effective strategies for adding dynamic programming as an important element of multi-objective approaches to multi-component problems. In addition to investigating DP variants, we introduce an evaluation strategy that allows us to simultaneously evaluate multiple solutions contained within a single TTP representation.

The paper is organized as follows: In the next section, we present the TTP single-objective and bi-objective formulations. Section III reviews related work. Section IV introduces the hybrid evolutionary-DP search algorithm to deal with the bi-objective TTP. The different variants of DP operators are presented and discussed in Section V. A number of operators that can be used together with DP are discussed in Section VI. Section VII introduces the problem benchmark, presents the experimental framework and discusses the results of the experiments. We conclude the paper in Section VIII.

II. TTP problem definition

In the classical formulation of TTP [2], the thief rents a car for travel between cities, and after the tour is completed he should have maximized the revenue which is calculated adding the values of all stolen items and deducting the cost

R. Santana acknowledges the support of the Spanish Ministry of Science, Innovation and Universities (Project TIN2016-78365-R), and the Basque Government (IT1244-19 and ELKARTEK Programs).

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

of renting the car during the time taken to visit the cities. Therefore, it is as important for the thief to collect the most valuable set of items as it is to keep their weight reduced so as to spend a shorter time traveling between the cities. To explain our approach we use the TTP definition introduced in [15].

Let $\pi = (\pi_1 = 1, \pi_2, \ldots, \pi_n)$ represent a tour such that $\pi_i = j$ iff j is the ith visited city of the tour. We assume that there are k items that could be taken in each city except the departure city π_1. The profit for taking object j in city i is represented as $p_{i,j}$. Similarly, the weight of object j in city i is represented as $w_{i,j}$.

The packing plan of the thief is represented as a binary decision vector $\rho \in \{0,1\}^m$ where m is the number of items that could be picked in any of the cities and $\rho = (\rho_{2,1} \ldots \rho_{2,k}, \ldots, \rho_{n,1} \ldots \rho_{n,k})$. $\rho_{i,l} = 1$ iff item l in city i is chosen, and 0 otherwise.

A TTP solution is represented as a pair (π, ρ), as illustrated in Figure 1, where for $n = 8$, $k = 2$ the packing plan has size $m = k * (n-1) = 14$. Notice in the figure that items are not available at the first (departure) city in the tour.

Other parameters that describe the problem are:

- Capacity of the knapsack: C
- Total weight of items sequentially selected in the cities from π_1 to π_i: $W_{\pi_i} = \sum_{j=1}^{i} \sum_{l=1}^{k} w_{j,l}\rho_{j,l}$.
- Velocity of the thief at the moment of quitting the ith city: $v_{\pi_i} = v_{max} - \nu W_{\pi_i}$ where $\nu = \frac{(v_{max} - v_{min})}{C}$, and v_{max} and v_{min} are the maximum and minimum velocity.

The two components in the original TTP objective function are respectively represented by Equation (1) and Equation (2).

$$g(\pi, \rho) = \sum_{i=2}^{n} \sum_{l=1}^{k} p_{i,l}\rho_{i,l}, \; st. \sum_{i=2}^{n} \sum_{l=1}^{k} w_{j,l}\rho_{l,k} < C \quad (1)$$

$$h(\pi, \rho) = \left(\sum_{i=1}^{n-1} \frac{d_{\pi_i, \pi_{i+1}}}{v_{\pi_i}} \right) + \frac{d_{\pi_n, \pi_1}}{v_{\pi_n}} \quad (2)$$

Equation (1) represents the gain obtained from the items taken by the thief during the tour, constrained on the maximum capacity of the knapsack. Equation (2) computes the time spent on the tour. In the original formulation of the problem these two objectives were combined as $f(\pi, \rho) = g(\pi, \rho) - R\, h(\pi, \rho)$, being R the renting rate of a car paid for by the thief to travel between the cities. In the bi-objective formulation proposed in [1] these two objectives are independently optimized. Since they are conflicting, the goal of the optimization problem is to find the Pareto set of trade-off solutions.

III. Related work

Although the TTP problem is relatively new, an increasing number of papers have investigated this problem from different perspectives, and considering a number of variants. In this section we review some of this previous work with a focus on approaches and operators related to our proposal.

The most common approach to solve TTP is to separate the tour improvement stage and the item packing stage. Examples

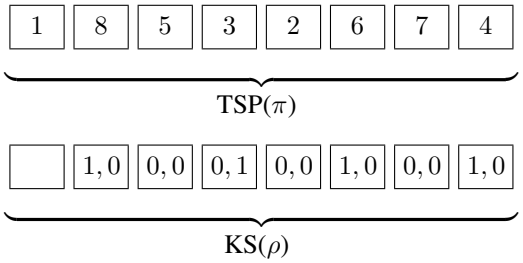

Fig. 1. Example of the TTP candidate solution representation for $n = 8$, $k = 2$.

of previous algorithms that have organized the search of solutions in this way include the CoSolver algorithm [3], the Cooperative Co-evolution approach [13], the Two-Stage Memetic Algorithm (TSMA) [12], and CS2SA [4]. In particular, the CoSolver approach decomposes the complete problem into two subproblems: The Traveling Salesman with Knapsack Problem (TSKP) which consists of finding the best tour when the packing plan is fixed, and the Knapsack on the Route Problem (KRP) which, given a fixed tour tries to find the best packing plan. This approach is extended in [4] by including complexity reduction and performance enhancement techniques. While decomposing the problem is the most common approach, some research also addresses both sub-problems at the same time [8]. The DP operators introduced in this paper modify only the packing plan of the TTP solution. However, some of these operators take into account the characteristics of the TSP tour at the time of optimizing the packing plan.

The packing routine heuristic (PRH) was originally proposed in [6], and has been acknowledged as an important component of successful algorithms [18]. PRH first assigns a score $s_{i,l}$ to item $\rho_{i,l}$ according to Equation (3).

$$s_{i,l} = \frac{p_{i,l}^{\alpha}}{w_{i,l}^{\alpha} d_{\pi_i, \pi_n}} \quad (3)$$

where α is a parameter used to tune the influence of the profit and weight parameters. The PRH as introduced in [6] sorts the items according to $s_{i,l}$ and then constructs a solution by adding items until the capacity of the knapsack is reached.

In [15], a benchmark set of TTP problems is introduced covering different characteristics of the instances. The benchmark is constructed combining TSP instances from the TSP library [16] and knapsack instances introduced by Martello et al [9]. We use a subset of these instances for evaluating the performance of our algorithms. They are discussed in detail in Section VII-A.

Bi-objective TTP variants have been investigated to a lesser extent than their single-objective counterpart. The variant investigated in this paper was proposed by Blank et al. [1] which compare a greedy approach, an independent sub-problem algorithm, and NSGA-II for instances of up to 100 cities and up to 10 items per city. They point to the advantages of MOEAs in comparison to deterministic algorithms although they acknowledge the need for improvement in the performance of the variation operators.

Wu et al. [19] propose the combination of a dynamic programming approach and evolutionary search for a class of bi-objective TTP problems different to the one addressed in this paper. The bi-objective problem investigated in [19] tries to maximize the total reward by minimizing the accumulated weight of the picked items. The DP method applied to that bi-objective problem is the one introduced in [14].

There are important differences between our contribution and the work described in [19]: their DP approach works on the complete set of items, and depends on a given order of the cities, while ours can be applied to any subset of cities selected in an arbitrary order. We apply DP not only using the profit as the input value but also using other metrics that incorporate information about the quality of the TSP tour. Furthermore, applying DP to the whole solution is only feasible for small instances and consequently the problems addressed in [19] have a relatively small number of cities, $n \in \{51, 76, 101\}$. However, we are able to apply our DP approach on problems with $n \in \{280, 4461, 33810\}$. Despite the fact that the differences between the DP-algorithms presented in this paper and those reported in [19] are important, both works corroborate the gains from the combination of DP techniques and population-based search.

Although in-depth analysis of the behavior of problem solvers for the bi-objective variant of TTP are scarce, in the single-objective case such analyses have been reported. A comparison of some of the most efficient TTP solvers for an extensive set of instances is presented in [4]. An overview and comparison of 21 algorithms, including EAs and other heuristics, for the single-objective TTP is presented in [18].

IV. Hybrid evolutionary-DP search

To evaluate the performance of the DP operators introduced in the paper, we use a hybrid multi-objective evolutionary DP algorithm (MO-DP) in which a set of non-dominated solutions simultaneously serve as an archive of the best solutions found so far and as a population from which solutions are selected for local optimization. Algorithm 1 shows the pseudocode of MO-DP.

Algorithm 1 Hybrid evolutionary-DP algorithm (MO-DP)

1: Initialize population of non TTP non-dominated solutions (Pop)
2: **for** $i = 1$ to n_{iter} **do**
3: Select a solution x from Pop
4: Create solution $\bar{x}$ applying DP operator to x
5: Create subset of non-dominated candidate solutions from evaluating x
6: Add non-dominated solutions to Pop
7: Using a pre-defined criterion, remove solutions from Pop so $|Pop| = M$
8: **end for**

A. MO-DP initialization

As is usually the case with TTP approaches, MO-DP starts from TSP tours found using the very efficient Lin-Kernighan method [7]. We use the Lin-Kernighan-Helsgaun (LKH) solver[1] to find a set of (not necessarily optimal) solutions to the TSP problem. The number of TSP solutions generated using LKH is equal to the population size M plus one additional solution.

In order to create initial packing plan solutions, we combined solutions generated by three different methods: 1) Sorting $\rho_{i,k}$ values according to profit, and selecting a number of items according to that order up to a given maximum capacity. 2) Similar to the previous method but sorting $\rho_{i,k}$ values according to the profit/gain ratios. 3) Using the best $\rho_{i,l}$ assignment as computed by a DP algorithm applied to subsets of cities.

The first two initialization methods start by sorting all items according to a given criterion, the first method sorts $\rho_{i,k}$ values according to profit, and the second according to profit/weight values. Then both methods create a packing plan by adding items according to the ordering computed, until a previously defined maximum weight value for the knapsack is reached. This maximum weight value is determined using a parameter $0 \geq R_w \leq 1$ multiplied by the knapsack capacity.

By using different values of R_w, it is possible to design packing plans with a varying number of time durations for the final TTP solution. Notice that these initialization procedures do not depend on the TSP tour. The third method we use to initialize packing plans is an exact DP procedure commonly used for solving small instances of the knapsack problem [10] and which will be explained in more detail in Section V.

Given the initial best TSP tour and a set of candidate packing plans, we pair them before evaluating them as candidates to populate the population (P). Each pair of components can generate different solutions identified during the fitness evaluation process as explained in the following section.

B. Fitness evaluation, selection and archiving strategies

In the bi-objective TTP, some partial solutions of (π, ρ) can be non-dominated and relevant for the search. We define a *partial solution* of (π, ρ) as a pair (π, ρ'), where ρ' is defined according to a parameter $r \in \{1, \dots n\}$ in the following way:

$$\rho'_{\pi(i),j} = \begin{cases} \rho_{\pi(i),j}, \forall j & \text{if } i \geq r \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Starting from a full solution it is possible to create TTP partial solutions that require less time for the thief tour by removing more items from ρ. The tour configuration is the same for all solutions, but at each step i the packing plan is changed, all items from city $\pi(i)$ are removed. All these solutions will have different values of the two objectives. They will also be non-dominated among them since the addition of one item necessarily increases the total profit but also increases the total time.

We implement the fitness function evaluation so that multiple partial solutions contained in the (π, ρ) could be evaluated. Depending on the number of items included in the original ρ

[1] Available from http://www.akira.ruc.dk/~keld/research/LKH/

configuration, we may obtain, as a result of the evaluation, a varying number of solutions; with a maximum number of n. For, large n, getting all possible solutions from one single (π, ρ) evaluation can be inefficient. Therefore, we set a parameter for the evaluation procedure that evaluates only μ solutions. This is done by taking a subset of cities located at regular intervals in the tour. For example, for a tour $\pi = (1, 8, 5, 3, 2, 6, 7, 4)$ we may take $\mu = 2$ solutions, taking solutions determined by starting positions $i = 1$ and $i = 5$. For the first solution no item from the starting ρ is set to zero. For $i = 5$, all items in cities $\{\pi_2 = 8, \pi_3 = 5, \pi_4 = 3\}$ are set to zero.

In order to apply the variation operator, a solution is randomly selected from the population and, as a result of the evaluation, multiple non-dominated solutions can be generated which are added to the archive. Selection implicitly occurs every time the archive reaches its maximum size and non-dominated solutions have to be removed from it. When this occurs, non-dominated solutions are sorted in descending order using the individual hypervolume, and added to the archive one by one, recomputing the individual HV at each step, until the maximum size is reached.

At each generation, MO-DP keeps a set of up to M non-dominated solutions. To focus on the analysis of these operators, MO-DP does not use recombination. The addition of crossover operators is left for future research.

V. Dynamic programming operators for the bi-objective TTP

To discuss the DP algorithm, and to simplify the presentation, we use a notation slightly different from the one presented for TTP in Section II.

Given a sequence $s = (s_1, s_2, \ldots, s_m)$ of m items, we represent any sub-sequence $(s_1, s_2, \ldots, s_k)$ as $s_{\leq k}$. The classical DP approach to knapsack defines a method to recursively compute which is the best possible profit $P(m, w)$ that can be obtained by selecting a subset of items from $s = (s_1, s_2, \ldots, s_m)$ whose accumulated weight does not exceed w. The solution for subproblem $s_{\leq k}$ is defined in terms of the solution for subproblem $s_{\leq k-1}$ using the following recursive formula.

$$P(k, w) = \begin{cases} P(k-1, w), & \text{if } w_k > w \\ \max(P(k-1, w), P(k-1, w-w_k) + p_k), & \text{else} \end{cases} \quad (5)$$

where w_k and p_k respectively represent the weight and profit for item k.

The equation captures the idea that the best subset of items with total weight w is either:

1) The best subset of $s_{\leq k-1}$ with total weight w, or
2) The best subset of $s_{\leq k}$ that has total weight $w - w_k$ plus item k.

The DP approach uses memoization for an efficient storage of partial computations. We used an efficient Python implementation[2] of DP as the kernel for the DP variants investigated in this paper.

A. Using DP for partially improving solutions

For a fixed TSP tour, DP can be used to find solutions to the bi-objective variant of the TTP problem by applying the algorithm for different values on a given maximum weight (lower than or equal to the maximum capacity). For each value of the total weight, the solution would be optimal in terms of the items picked. On the other hand, solutions obtained in this way would be non-dominated since less weight means that the time spent to complete the tour is also smaller.

DP can find an optimal packing plan for small instances of the problem (in our experiments up to 280 items) but it requires a high computational time. Furthermore, as the number of items is increased, the memory and computational time requirements make the DP application infeasible. Nevertheless, it is possible to use DP as a local optimizer if, for a given set of candidate items, some of which are part of the current selected packing plan, it is able to find a different set of items *with the same weight* but a higher total profit. Therefore, such a DP-based optimizer would first identify a subset of candidate items, some of which are present in the current solution, then would compute the weights of those that are in the solution, and finally would apply DP for all candidates using as maximum capacity that corresponding to the sum of weights of those candidate items present in the solution.

We used such a DP-based local optimizer in two phases of the algorithm. 1) For the initialization of the packing plans, and 2) During the evolution as a local optimizer.

When used for initialization, the initial packing plan is empty and we cannot estimate the values of a maximum capacity to optimize as previously explained. In this case we use a variation of the DP algorithm. We split the number of cities of the given solution (and their corresponding items) into subsets of a manageable size. For example, let us suppose the number of subsets to be q, then we apply DP to each of the $r = \left\lfloor \frac{n}{q} \right\rfloor$ problems, using as maximum capacity for each of these problems $c = \left\lfloor \frac{C}{q} \right\rfloor$, where we guarantee that r and c are integer numbers. The TTP solution obtained by combining the partial solutions of all the q problems satisfies the general constraint of $\sum_{j=2}^{n} \sum_{l=1}^{k} w_{j,l} \rho_{j,l} \leq C$.

When applied as a local optimizer, a packing plan for that solution already exists and the key question is how to select the candidate items to be optimized. We introduce two general strategies for this purpose:

1) Methods based on item importance computation.
2) Methods based on item pre-selection.

B. DP methods based on item importance computation

There are four parameters that shape the application of the method.

[2]Available from https://codereview.stackexchange.com/questions/20569/dynamic-programming-knapsack-solution/20581#20581

Window size (WS): Number of items selected as candidates.
Value metric (VM): This can be the profit or a distance-based metric computed from the TSP tour.
Weight proportion (WP): Proportion of weight from the original solution that is set as maximum capacity.
Intensity (In): Number of times DP is applied to the same solution.

Given an input TTP solution defined by the pair (π, ρ), the partial DP operator (PDP) starts by selecting from π a random subset of cities π_A, guaranteeing that there is at least one selected item from any of these cities in ρ. The number of selected cities multiplied by the number of items at each city is equal to WS. All the items in cities from π_A are taken as candidates and the total weight $\widehat{W}$ of those items that are selected in ρ is computed.

For each of the items, a value metric VM is computed. In the simplest variant, this corresponds to the profit associated to the item. We also consider another metric:

$$r_d(\rho_{i,l}) = \frac{d_{\pi_i,\pi_n}}{\sum_{j \in \pi_A} d_{\pi_j,\pi_n}} \tag{6}$$

The idea of $r_d(\rho_{i,l})$ is to bias the selection of the items taking into account the distance between the cities where items are located and the last city of the tour. Since in a TTP solution items from cities that are more distant from the end of the tour will have to be carried on for a longer part of the tour, and therefore their weight will have a higher impact on the time spent by the thief, r_d will prioritize taking items from cities closer to the end. This prioritization is integrated with DP by adjusting the profit of an item $\rho_{i,l}$ according to Equation 7:

$$\widehat{p}_{i,l} = \lceil p_{i,l} \cdot (1 - R_d \cdot r_d(\rho_{i,l})) \rceil \tag{7}$$

where the bias ($0 < R_d < 1$) indicates the strength of the penalization to cities that are far from the last one. R_d is randomly selected in each call to the operator to promote diversity of the solutions in the Pareto set.

The weight proportion parameter WP is used to modify the maximum weight that is passed as a parameter to the DP procedure. We consider only two scenarios, when $WP = 1$, the maximum capacity passed to DP is the same as in the current solution. When $WP = 2$, this maximum capacity is a random value between the weight of the current solution and twice that weight value. The operator is described by Equation 8 and Equation 9:

$$W = \sum_{j \in \pi} \sum_{l \in \{1,\ldots,k\}} \rho_{j,l} w_{j,l} \tag{8}$$

$$\widehat{W} = W \cdot (1 + R_w) \tag{9}$$

where the bias ($0 < R_w < 1$) indicates the amount of expansion of the current weight.

The idea of allowing DP to use a weight above the current weight of the partial solution is to allow the algorithm to explore the addition of new items, increasing the profit of the current solution. If the addition of new items causes the total weight to exceed the maximum capacity, the corresponding (overweighted) partial solutions are discarded in the fitness evaluation step.

The last parameter used by DP is the number of times that it is applied to a given solution. This means the repetition of all previously described phases. Considering that some TTP instances may have thousands of cities, and that WS is necessarily constrained to relatively small values, the multiple application of DP is needed to deal with large TTP instances.

C. Methods based on item pre-selection

Methods based on item pre-selection start by selecting an item from ρ, and then selecting a number of "related" items based on different criteria. For this set of candidate items, the method will identify the items that are already part of the current solution and their total weight. Finally, DP is called with all the candidate items and the current total weight of items in the solution is set as the maximum weight

Let i be the item selected from ρ, from a list of items sorted according to some criterion, being s_i the position of item i in this list, then candidates comprise those solutions in $(\max(s_i - \frac{k}{2}, 0) \ldots, s_i, \ldots, \min(s_i + \frac{k}{2}, m))$, that is, a maximum of k items. As criteria to sort the solutions we used the *weights*, *profits*, and *ratios* between profits and weights. We also used the *PRH* values computed as explained in Section III. After the set of candidate items has been determined, a DP variant that receives the list of weights and profits for the selected candidates as inputs is applied.

By using different criteria to select the candidate items, we expect to substitute the current items by those more likely to be good candidates for substitution. For example, items with closer weights could be good candidates to substitute the selected item.

VI. Complementary operators for the DP approaches

Although some variants of DP take into account information about the TSP structure, they only modify the packing plan part of the solution. Furthermore, for the large TTP instances, the application of DP operators will only modify a fraction of the packing plan variables. Therefore, it is expected that, in order to be competitive with other algorithms, DP operators should be applied together with other operators. In this section, we briefly introduce three operators that were used in order to create TTP optimization approaches competitive with state-of-the-art algorithms.

The PRH operator is defined as a variation of the PRH technique, so each time the operator is called, a random $1 \leq \alpha \leq 6$ is selected and the candidate packing plan ρ is constructed using Equation 3.

The rotation operator (PR) makes a rotation of l positions in the sub-sequence comprised by $\pi_2, \pi_3, \ldots, \pi_n$. It can be applied clockwise or anticlockwise. This operator exploits the fact that only two neighbor distances will be affected as a result of a rotation.

Given a solution to the bi-objective TTP, the profit improver operator searches for a change to the packing plan component that simultaneously improves the two objectives of the problem. This is done by first randomly selecting a pivot item, and then identifying another item that could replace it and produce an improvement in the objectives.

A. Algorithms based on combined operators

As the DP component, we selected DP methods based on item importance computation, with fixed parameters $VM = r_d$ and $WP = 1$. The particular configurations of the combined operator were:

- *DP20*: $In = 20$.
- *Rot_PP_DP10*: First a rotation operator is applied. Then, the packing routine heuristic is used to initialize the packing plan. Finally, DP is applied, $In = 20$.
- *Rot_PP_DP10_PI*: First a rotation operator is applied. Then, the packing routine heuristic is used to initialize the packing plan. Subsequently DP is applied, $In = 20$. Finally, a profit improver algorithm is applied.
- *DP10_PI*: DP is applied, $In = 10$. Then, a profit improver algorithm is applied.
- *DP_30_PI*: DP is applied, $In = 30$. Then, a profit improver algorithm is applied.

VII. Experiments

The main goal of our experiments is to investigate the impact that the proposed DP-based operators have on the behavior of the MO-DP algorithm. To that end, we compare the performance of the different methods. Although our goal is not to propose a new state-of-the-art algorithm for the bi-objective TTP, we do compare the results produced by the best variants of the MO-DP with state-of-the-art methods and analyze the differences in their behavior.

A. Function benchmark

We use the bi-objective function benchmark proposed for the *GECCO-2019* competition[3]. TTP instances in this benchmark were originally introduced, for the single-objective case, in [15]. The benchmark comprises 9 problems whose characteristics are described in Table I. There are three types of knapsack instances according to their difficulty as investigated in [9]: 1) uncorrelated, 2) uncorrelated with similar weights, and 3) bound strongly correlated types. The description of the knapsack type is given in columns *bound*, *type corr.*, *w* and *corr.* of Table I.

To compare the results of the algorithms we use the hypervolume (HV) metric. A higher value of HV indicates that the PF approximation produced by the algorithm is better. The GECCO challenge established that algorithms should produce a PF approximation of fixed size as output. The size was specified for each group of instances as: $(a280, 100)$, $(fnl4461, 50)$, $(pla33810, 20)$.

[3]https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

TABLE I
TTP function benchmark, originally proposed as part of the *GECCO-2019*, and used in the experiments.

Instance	n	k	bound	type corr.	w	corr.
a280-n279	280	1	bd	strong		0.1
a280-n1395	280	5		uncorr.	≈	0.5
a280-n2790	280	10		uncorr.		1.0
fnl4461-n4460	4461	1	bd	strong		0.1
fnl4461-n22300	4461	5		strong	≈	0.5
fnl4461-n44600	4461	10		uncorr.		1.0
pla33810-n33809	33810	1	bd	strong		0.1
pla33810-n169045	33810	5		strong	≈	0.5
pla33810-n338090	33810	10		uncorr.		1.0

1) Parameters of the algorithm: There are a number of general parameters used by the MO-DP algorithm. These include:

- Population size: This was selected according to the PF size defined for the GECCO-2019 challenge.
- Maximum number of generations: $g = 1000$.
- Window size for DP variants: $w = 150$.

B. Analysis of the DP operators

We compare different variants of DP operators, grouped in the two classes introduced in Section V-A 1) Methods based on item importance computation, and 2) Methods based on item pre-selection. For the first class of algorithms, 8 possible configurations of the parameters: $VM \in \{profit, r_d\}$, $WP\{1, 2\}$, $In \in \{1, 10\}$ were evaluated. For the second class of problems, we also evaluate 8 configurations, determined by the parameters $VM \in \{weight, profit, ratio, PRH\}$ and $In \in \{1, 10\}$. The HV results produced by all the algorithms are shown in Table II, where the best results for each class of algorithms are highlighted.

It can be seen in Table II that, for each class of algorithms there is a single configuration whose performance outperforms the other. For the methods based on item importance computation, the best choice is to use a metric that scores the items taking into account the distance from the end of the tour to the the cities they belong to. This configuration keeps the original weight of the solution being modified ($WP = 1$) and executes 10 repetitions of the DP operator. For the second class of methods, the best combination includes the PRH metric, 10 repetitions of the DP operator, and in this case a maximum capacity higher than that of the original solution. Considering the two classes of algorithms, the best configuration of the methods based on item importance computation is the absolute winner for all instances, although in some cases the difference between the HV of the first and second ranked algorithms is small.

In general, the results indicate that scoring the items with metrics that incorporate information about the distances between the cities is more effective that using the original profits as the traditional DP operator does. Results also reveal that more applications of the DP operator improve the quality of the solutions. Finally, the effect of allowing DP to increase the

TABLE II
HV RESULTS FOR DIFFERENT VARIANTS OF DP OPERATORS.

VM	In	WP		n279	n1395	n2790	n4460	n22300	n44600	n33809	n169045	n338090
r_d	1	1		0.9175	0.8001	0.8676	0.8774	0.7866	0.8589	0.8145	0.7676	0.8512
r_d	1	2		0.9012	0.7967	0.8664	0.8860	0.7851	0.8587	0.8303	0.7673	0.8510
profit	1	1		0.8527	0.7869	0.8630	0.8185	0.7835	0.8582	0.7911	0.7670	0.8510
profit	1	2		0.8785	0.7886	0.8637	0.8347	0.7838	0.8583	0.7950	0.7669	0.8509
r_d	10	1		**0.9177**	**0.8052**	**0.8759**	**0.9094**	**0.7988**	**0.8639**	**0.8769**	**0.7729**	**0.8527**
r_d	10	2		0.8985	0.8015	0.8716	0.9003	0.7933	0.8606	0.8714	0.7691	0.8518
profit	10	1		0.8236	0.7858	0.8636	0.8287	0.7836	0.8583	0.8051	0.7671	0.8509
profit	10	2		0.8570	0.7863	0.8632	0.8410	0.7840	0.8583	0.8099	0.7671	0.8509
weights	10	1		0.8638	0.7861	0.8635	0.8020	0.7842	0.8584	0.7893	0.7673	0.8510
weights	10	2		0.8726	0.7859	0.8632	0.8231	0.7841	0.8583	0.7938	0.7671	0.8509
profit	10	1		0.8532	0.7850	0.8631	0.8015	0.7832	0.8582	0.7873	0.7669	0.8509
profit	10	2		0.8560	0.7960	0.8635	0.8221	0.7849	0.8583	0.7913	0.7670	0.8510
ratios	10	1		0.8171	0.7852	0.8626	0.7995	0.7833	0.8582	0.7878	0.7669	0.8509
ratios	10	2		0.8691	0.7983	0.8650	0.8287	**0.7850**	0.8583	0.7952	0.7671	0.8509
PRH	10	1		0.8513	0.7858	0.8635	0.8094	0.7849	0.8584	0.8145	0.7680	0.8509
PRH	10	2		**0.8862**	**0.8026**	**0.8689**	**0.8770**	0.7839	**0.8584**	**0.8479**	**0.7706**	**0.8522**

total weight of the optimized solution depends on the type of DP operator strategy selected.

C. Comparison with state-of-the-art algorithms

We compare the DP-based algorithms described in Section VII-B with the three best contenders in an open challenge held at GECCO-2019[4]. Open challenges provide a unique chance to test a variety of approaches. Each team had to submit the Pareto sets found by their algorithm, where the number of solutions in the Pareto set approximations were constrained as done in this paper. Algorithms HPI, Jomar, and NTGA, identified by the names of the teams that submitted them to the challenge, reached the first, second and third places of the challenge, respectively.

Due to the complexity of the problems, there were no particular restrictions in terms of the computational time or number of evaluations used by each team or in the number of runs of the algorithms used to find the best PS approximation. Therefore, our comparison focuses on the quality of the obtained solutions.

For each of the DP-based algorithms, we ran 5 executions of the algorithm with a maximum of 1500 evaluations. Each algorithm produces an output of exactly $l \in \{100, 50, 20\}$ solutions. We then compute the set of non-dominated solutions from the joint set of $l \times 5$ outputs, and further refine this set to only l solutions by removing those with lower individual HV contribution. This is the set of solutions which is compared to the state-of-the-art algorithms. Following the GECCO challenge evaluation pipeline, the HV of all algorithms was computed. The results are shown in Table III.

The analysis of Table III reveals that among the DP-based algorithms, the best contender, for 7 of the 9 instances, is Algorithm DP_30_PI, although the differences with algorithm DP20, which does not incorporate any non DP-based operator, are very small. For instances pla33810-n169045 and pla33810-n338090, the best DP-based contender is Algorithm Rot_PP_DP10_PI. These results suggest that for this large instance, a component that modifies the TSP tour is indeed important. Another remarkable observation is that algorithms DP_30_PI, DP20, and Rot_PP_DP10_PI outperform algorithms Jomar and NTGA for all pla33810 instances. Only algorithm HPI beats them for this class of high-dimensional problems.

To obtain a general assessment of the performance for the complete set of instances, a ranking of all algorithms was computed by assigning three points to the algorithm with the best HV for each of the instances, 2 points to the second ranked algorithm and 1 to the one ranked third. This ranking is shown in Table IV. The first position in the rank is HPI, which is the top performing method for all instances. In the second position is Jomar. The best DP-based variant, DP_30_PI, ranks third, above NTGA, the algorithm that was the third contender for the GECCO competition. In summary, the algorithms are competitive with jomar and NTGA, particularly for the hardest instances, although they are not able to beat the best algorithm.

VIII. CONCLUSIONS

In this paper we have investigated the use of dynamic programming for designing variation operators for the bi-objective TTP problem. We have also presented a way to implement an efficient evaluation of multiple TTP solutions. The introduced variants of DP incorporate more information about the TTP characteristics. Some of the introduced DP variants exploit information about the TSP tour, effectively combining information about the two TTP components. Our results show that the use of DP as a variation operator can lead to more accurate approximations of the Pareto front. The comparison with state-of-the-art results has shown that some of the DP variants are competitive with the best contenders. However, this comparison can not be taken as definitive since information about the number of evaluations or the computational resources used to compute the best contenders is not available. While our emphasis has been on the knapsack component of TTP, further improvements are expected from

[4]https://www.egr.msu.edu/coinlab/blankjul/gecco19-thief/

TABLE III
RESULTS OF THE COMPARISON BETWEEN ALGORITHMS THAT INCORPORATE THE DP VARIANTS AND THE STATE-OF-THE-ART ALGORITHMS

Algorithm		n279	n1395	n2790	n4460	n22300	n44600	n33809	n169045	n338090
HPI		0.9404	0.8284	0.8860	0.9339	0.8229	0.8825	0.9229	0.8107	0.8748
Jomar		0.9387	0.8241	0.8864	0.9327	0.8187	0.8743	0.8356	0.7279	0.8521
NTGA		0.9307	0.8141	0.8809	0.9140	0.8076	0.8242	0.8819	0.7638	0.7788
DP20		**0.9257**	0.8064	0.8785	0.9150	0.8064	0.8697	0.8960	0.7795	0.8575
Rot_PP_DP10		0.9201	0.7949	0.8682	0.8348	0.7831	0.8581	0.8458	0.7949	0.8664
Rot_PP_DP10_PI		0.9205	0.7967	0.8682	0.8356	0.7831	0.8581	0.8464	**0.7954**	**0.8671**
DP10_PI		0.9262	0.8061	0.8780	0.9130	0.8038	0.8663	0.8896	0.7762	0.8556
DP_30_PI		0.9256	**0.8065**	**0.8787**	**0.9153**	**0.8079**	**0.8716**	**0.8984**	0.7831	0.8594

TABLE IV
JOINT RANKING OF THE DP-BASED AND THE STATE-OF-THE-ART ALGORITHMS.

Algorithm	
HPI	26
Jomar	13
DP_30_PI	5
Rot_PP_DP10_PI	4
NTGA	3
%hline Rot_PP_DP10	2
DP20	1
DP10_PI	0

exploring other ways to combine information from the two TTP components.

A. Future work

There are several ways in which the work presented in this paper could be extended. We have provided evidence that different DP operators have different impact on the solution of different instances, opening the door for their combined application, a path that we intend to follow in the future. Therefore, adapting the application of the operators to the characteristics of the solutions seems a promising approach since different areas of the Pareto set may require different ways to explore them. Similarly, use of probabilistic modeling strategies especially suited for permutation representations [17], [21] could lead to more effective search operators for the TSP component. Another possibility would be to design TSP operators that, when applied together with the introduced DP operators, could amplify their effect.

REFERENCES

[1] J. Blank, K. Deb, and S. Mostaghim. Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 46–60. Springer, 2017.

[2] M. R. Bonyadi, Z. Michalewicz, and L. Barone. The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1037–1044. IEEE, 2013.

[3] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki. Socially inspired algorithms for the travelling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 421–428. ACM, 2014.

[4] M. El Yafrani and B. Ahiod. Population-based vs. single-solution heuristics for the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2016)*, pages 317–324. ACM, 2016.

[5] M. El-Yafrani and B. Ahiod. Efficiently solving the traveling thief problem using hill climbing and simulated annealing. *Information Sciences*, 432:231–244, 2018.

[6] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner. Approximate approaches to the traveling thief problem. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 385–392. ACM, 2015.

[7] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[8] N. Lourenço, F. B. Pereira, and E. Costa. An evolutionary approach to the full optimization of the traveling thief problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 34–45. Springer, 2016.

[9] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

[10] S. Martello and P. Toth. *Knapsack problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd., 1990.

[11] Y. Mei, X. Li, F. Salim, and X. Yao. Heuristic evolution with genetic programming for traveling thief problem. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2753–2760. IEEE, 2015.

[12] Y. Mei, X. Li, and X. Yao. Improving efficiency of heuristics for the large scale traveling thief problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 631–643. Springer, 2014.

[13] Y. Mei, X. Li, and X. Yao. On investigation of interdependence between sub-problems of the travelling thief problem. *Soft Computing*, 20(1):157–172, 2016.

[14] F. Neumann, S. Polyakovskiy, M. Skutella, L. Stougie, and J. Wu. A fully polynomial time approximation scheme for packing while traveling. *CoRR*, abs/1702.05217, 2017.

[15] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 477–484. ACM, 2014.

[16] G. Reinelt. TSPLIB-A traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[17] R. Santana, G. Sirbiladze, B. Ghvaberidze, and B. Matsaberidze. A comparison of probabilistic-based optimization approaches for vehicle routing problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2606–2613. IEEE, 2017.

[18] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, 24(3):295–320, 2018.

[19] J. Wu, S. Polyakovskiy, M. Wagner, and F. Neumann. Evolutionary computation plus dynamic programming for the bi-objective travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 777–784. ACM, 2018.

[20] M. E. Yafrani, M. S. Martins, M. E. Krari, M. Wagner, M. R. Delgado, B. Ahiod, and R. Lüders. A fitness landscape analysis of the travelling thief problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 277–284. ACM, 2018.

[21] M. Zangari-de Souza, A. Mendiburu, R. Santana, and A. Pozo. Multiobjective decomposition-based Mallows models estimation of distribution algorithm. A case of study for permutation flowshop scheduling problem. *Information Sciences*, 397–398:137–154, 2017.